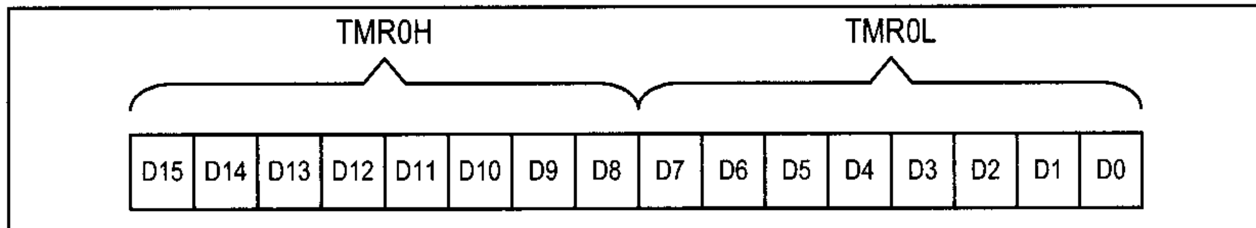
# Unit 3: Facilities in PIC 18 Part I

**Programming timer 0 and 1:-**

Every timer needs a clock pulse to tick. The clock source can be internal or external. If we use the internal clock source, then 1/4th of the frequency of the crystal oscillator on the OSC1 and OSC2 pins (Fosc/4) is fed into the timer. Therefore, it is used for time delay generation and for that reason is called a timer. By choosing the external clock option, we feed pulses through one of the PIC18's pins: this is called a counter.

## Timer0 registers and programming

Timer0 can be used as an 8-bit or a 16-bit timer. The 16-bit register of Timer0 is accessed as low byte and high byte, as shown in Figure 9-1. The low-byte register is called TMR0L (Timer0 low byte) and the high-byte register is referred to as TMR0H (Timer0 high byte). These registers can be accessed like any other special function registers. For example, the instruction "`MOVWF TMR0L`" moves the value in WREG into TMR0L, the low byte of Timer0. These registers can also be read like any other register. For example, "`MOVFF TMR0L, PORTB`" copies TMR0L (low byte of Timer0) to PORTB.




**T0CON: TIMER0 CONTROL REGISTER:-**

| TMR0ON | T08BIT | T0CS | T0SE | PSA | T0PS2 | T0PS1 | T0PS0 |
|---|---|---|---|---|---|---|---|

bit 7 bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is not assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits

111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

**INTCON: INTERRUPT CONTROL REGISTER**

| GIE/GIEH | PEIE/GIEL | TMR0IE | INT0IE | RBIE | TMR0IF | INT0IF | RBIF |
|---|---|---|---|---|---|---|---|
| bit 7 | | | | | | | bit 0 |

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit

1 = TMR0 register has overflowed
0 = TMR0 register did not overflow

## Timer1 programming

Timer1 is a 16-bit timer, and its 16-bit register is split into two bytes, referred to as TMR1L (Timer1 low byte) and TMR1H (Timer1 high byte). See Figure 9-8. Timer1 can be programmed in 16-bit mode only and unlike Timer0, it does not support 8-bit mode. Timer1 also has the T1CON (Timer 1 control) register in addition to the TMR1IF (Timer1 interrupt flag). The TMR1IF flag bit goes HIGH when TMR1H:TMR1L overflows from FFFF to 0000. Timer1 also has the prescaler option, but it only supports factors of 1:1, 1:2, 1:4, and 1:8. See Figure 9-9 for the Timer1 block diagram and Figure 9-10 for T1CON register options. The PIR1 register contains the TMR1IF flags. See Figure 9-11.

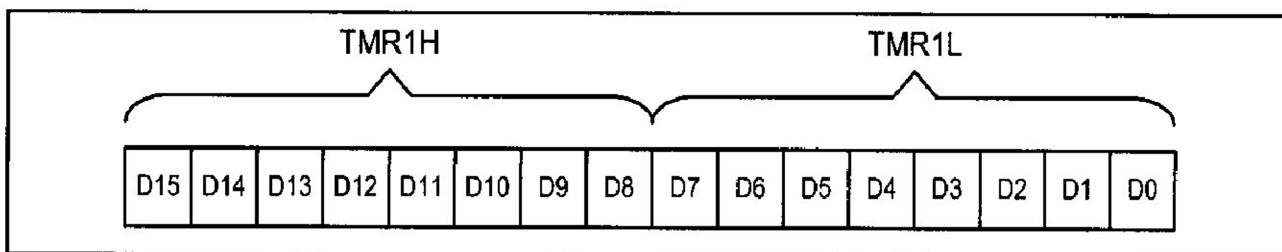



**Figure 9-8. Timer1 High and Low Registers**

**T1CON: TIMER1 CONTROL REGISTER**

| RD16 | — | T1CKPS1 | T1CKPS0 | T1OSCEN | T1SYNC | TMR1CS | TMR1ON |
|---|---|---|---|---|---|---|---|

bit 7 bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer1 in one 16-bit operation
0 = Enables register read/write of Timer1 in two 8-bit operations
bit 6 **Unimplemented:** Read as ‘0’
bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value
bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 oscillator is enabled
0 = Timer1 oscillator is shut-off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.
bit 2 **T1SYNC:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.
bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (FOSC/4)
bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

**PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1**

| PSPIF**(1)** | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF |
|---|---|---|---|---|---|---|---|

bit 7 bit 0

bit 0 **TMR1IF:** TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed
0 = TMR1 register did not overflow

## 16-bit timer programming

The following are the characteristics and operations of 16-bit mode:

1. It is a 16-bit timer; therefore, it allows values of 0000 to FFFFH to be loaded into the registers TMR0H and TMR0L.
2. After TMR0H and TMR0L are loaded with a 16-bit initial value, the timer must be started. This is done by "`BSF T0CON,TMR0ON`" for Timer0.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets HIGH a flag bit called TMR0IF (timer interrupt flag, which is part of the INTCON register). This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer.
4. After the timer reaches its limit and rolls over, in order to repeat the process, the registers TMR0H and TMR0L must be reloaded with the original value, and the TMR0IF flag must be reset to 0 for the next round.

## Steps to program Timer0 in 16-bit mode

To generate a time delay using the Timer0 mode 16, the following steps are taken:

1. Load the value into the T0CON register indicating which mode (8-bit or 16-bit) is to be used and the selected prescaler option.
2. Load register TMR0H followed by register TMR0L with initial count values.
3. Start the timer with the instruction "`BSF T0CON, TMR0ON`".
4. Keep monitoring the timer flag (TMR0IF) to see if it is raised. Get out of the loop when TMR0IF becomes high.
5. Stop the timer with the instruction "`BCF T0CON, TMR0ON`".
6. Clear the TMR0IF flag for the next round.
7. Go back to Step 2 to load TMR0H and TMR0L again.

To clarify the above steps, see Example 9-3. To calculate the exact time delay and the square wave frequency generated on pin PB5, we need to know the XTAL frequency. See Example 9-4 and Example 9-5.

Notice in Figure 9-5 that we should load **TMR0H first**, and **then load TMR0L**, because the value for TMR0H is kept in a temporary register and written to TMR0H when TMR0L is loaded. This will prevent any error in counting if the TMR0ON flag is set HIGH.

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay. Analyze the program.

```
      BCF   TRISB,5           ;PB5 as an output
      MOVLW 0x08              ;Timer0,16-bit,int clk,no prescale
      MOVWF T0CON             ;load T0CON reg.
HERE  MOVLW 0xFF              ;TMR0H = FFH, the high byte
      MOVWF TMR0H             ;load Timer0 high byte
      MOVLW 0xF2              ;TMR0L = F2H, the low byte
      MOVWF TMR0L             ;load Timer0 low byte
      BCF   INTCON, TMR0IF    ;clear timer interrupt flag bit
      BTG   PORTB,5           ;toggle PB5
      BSF   T0CON, TMR0ON     ;start Timer0
AGAIN BTFSS INTCON, TMR0IF    ;monitor Timer0 flag until
      BRA   AGAIN             ;it rolls over
      BCF   T0CON, TMR0ON     ;stop Timer0
      BRA   HERE              ;load TH, TL again
```

**Solution:**

In the above program notice the following steps:

1. T0CON is loaded.
2. FFF2H is loaded into TMR0H–TMR0L.
3. The Timer0 interrupt flag is cleared by the "`BCF INTCON, TMR0IF`" instruction.
4. PORTB.5 is toggled for the high and low portions of the pulse.
5. Timer0 is started by the "`BSF T0CON, TMR0ON`" instruction.
6. Timer0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the Timer0 flag (TMR0IF = 1). At that point, the "`BTFSS INTCON, TMR0IF`" instruction bypasses the "`BRA AGAIN`" instruction.
7. Timer0 is stopped by the instruction "`BCF T0CON, TMR0ON`", and the process is repeated.

Notice that to repeat the process, we must reload the TMR0L and TMR0H registers, and start the timer again.

Calculation the amount of time delay generated by the timer in above example if XTAL=10MHz.

**Solution:**

The timer works with the Fosc/4 clock; therefore, we have 10 MHz / 4 = 2.5 MHz as the timer frequency. As a result, each clock has a period of T = 1 /2.5 MHz = 0.4 μs. In other words, Timer0 counts up each 0.4 μs resulting in delay = number of counts × 0.4 μs.

The number of counts for the rollover is FFFFH − FFF2H = 0DH (13 decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raises the TMR0IF flag. This gives 14 × 0.4 μs = 5.6 μs for half the pulse. For the entire period the time delay generated by the timer is T = 2 × 5.6 μs = 11.2 μs.

## Finding values to be loaded into the timer

Assuming that we know the amount of timer delay we need, the question is how to find the values needed for the TMR0H and TMR0L registers. To calculate the values to be loaded into the TMR0L and TMR0H registers, look at Examples 9-8 and 9-9, where we use a crystal frequency of 10 MHz for the PIC18 system.

Assuming XTAL = 10 MHz and no prescaler we can use the following steps for finding the TMR0H and TMR0L registers' values:

1. Divide the desired time delay by 0.4 μs.
2. Perform 65,536 − $n$, where $n$ is the decimal value we got in Step 1.
3. Convert the result of Step 2 to hex, where *yyxx* is the initial hex value to be loaded into the timer's registers.
4. Set TMR0L = *xx* and TMR0H = *yy*.

Assuming that XTAL = 10 MHz, write a program to generate a square wave with a period of 10 ms on pin PORTB.3.

**Solution:**

For a square wave with T = 10 ms we must have a time delay of 5 ms. Because XTAL = 10 MHz, the counter counts up every 0.4 μs. This means that we need 5 ms / 0.4 μs = 12,500 clocks. 65,536 – 12,500 = 53,036 = CF2CH. Therefore, we have TMR0H = CF and TMR0L = 2C.

```
      BCF   TRISB,3          ;PB3 as an output
      MOVLW 0x08             ;Timer0,16-bit,int clk,no prescale
      MOVWF T0CON            ;load T0CON reg
HERE  MOVLW 0xCF             ;TMR0H = CFH, the high byte
      MOVWF TMR0H            ;load Timer0 high byte
      MOVLW 0x2C             ;TMR0L = 2CH, the low byte
      MOVWF TMR0L            ;load Timer0 low byte
      BCF   INTCON,TMR0IF    ;clear timer interrupt flag bit
      CALL  DELAY
      BTG   PORTB,3          ;toggle PB3
      BRA   HERE             ;load TH, TL again
;-----------delay using Timer0
DELAY BSF   T0CON,TMR0ON     ;start Timer0
AGAIN BTFSS INTCON,TMR0IF    ;monitor Timer0 flag until
      BRA   AGAIN            ;it rolls over
      BCF   T0CON,TMR0ON     ;stop Timer0
      RETURN
```

## 8-bit mode programming of Timer0

Timer0 can also be used in 8-bit mode. The 8-bit mode allows only values of 00 to FFH to be loaded into the timer's register TMRL0. After the timer is started, it starts to count up by incrementing the TMR0L register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets HIGH the TMR0IF. See Figure 9-7.

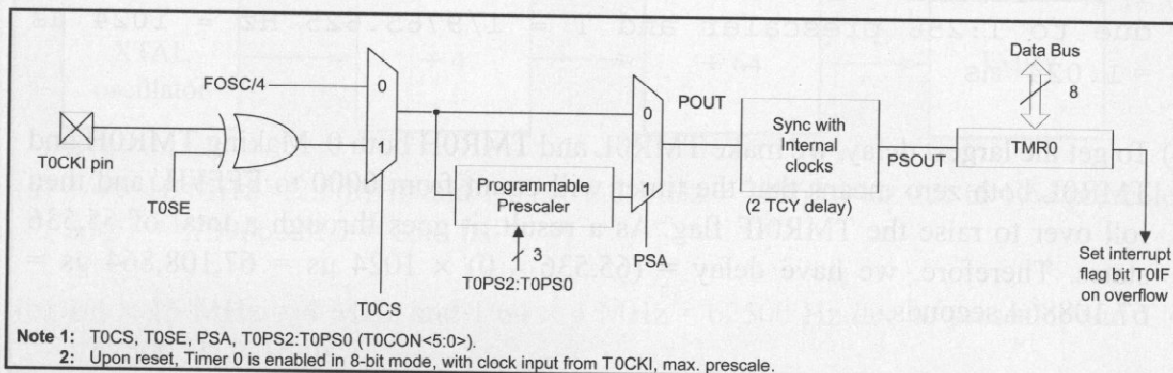



**Figure 9-7. Timer0 8-bit Block Diagram**

## Steps to program 8-bit mode of Timer0

To generate a time delay using Timer0 in 8-bit mode, take the following steps:

1. Load the T0CON value register indicating 8-bit mode is selected.
2. Load the TMR0L registers with the initial count value.
3. Start the timer.
4. Keep monitoring the timer flag (TMR0IF) to see if it is raised. Get out of the loop when TMR0IF becomes HIGH.
5. Stop the timer with the instruction "`BCF T0CON, TMR0ON`".
6. Clear the TMR0IF flag for the next round.
7. Go back to Step 2 to load TMR0L again.

Notice that when we choose the 8-bit option, only the TMR0L register is used and the TMR0H has a zero value during the count up. To clarify the above

Assuming that XTAL = 10 MHz, find (a) the frequency of the square wave generated on pin PORTB.0 in the following program, and (b) the smallest frequency achievable in this program, and the TH value to do that.

```
      BCF    TRISB,0           ;PB0 as an output
      MOVLW  0x48              ;Timer0,8-bit,int clk,no prescaler
      MOVWF  T0CON             ;load T0CON reg.
      BCF    INTCON,TMR0IF     ;clear timer interrupt flag bit
HERE  MOVLW  0x5               ;TMR0L = 5, the low byte
      MOVWF  TMR0L             ;load Timer0 byte
      CALL   DELAY
      BTG    PORTB,0           ;toggle PB0
      BRA    HERE              ;load TL again
;————————delay using Timer0
DELAY BSF    T0CON,TMR0ON      ;start Timer0
AGAIN BTFSS  INTCON,TMR0IF     ;monitor Timer0 flag until
      BRA    AGAIN             ;it rolls over
      BCF    T0CON,TMR0ON      ;stop Timer0
      BCF    INTCON,TMR0IF     ;clear Timer0 interrupt flag bit
      RETURN
```

**Solution:**

(a) Now (256 – 05) = 251 × 0.4 μs = 100.4 μs is the high portion of the pulse. Because it is a 50% duty cycle square wave, the period T is twice that; as a result T = 2 × 100.4 μs = 200.8 μs, and the frequency = 4.98 kHz.

(b) To get the smallest frequency, we need the largest T, and that is achieved when TMR0H = 00. In that case, we have T = 2 × 256 × 0.4 μs = 204.8 μs and the frequency = 1 / 204.8 μs = 4,882.8 Hz.

**T2CON: TIMER2 CONTROL REGISTER**

| — | TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 |
|---|---|---|---|---|---|---|---|
| bit 7 | | | | | | | bit 0 |

bit 7 **Unimplemented:** Read as '0'
bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale
bit 2 **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off
bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

**T3CON:TIMER3 CONTROL REGISTER**

| RD16 | T3ECCP1 | T3CKPS1 | T3CKPS0 | T3CCP1 | T3SYNC | TMR3CS | TMR3ON |
|---|---|---|---|---|---|---|---|
| bit 7 | | | | | | | bit 0 |

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operations
bit 6,3 **T3ECCP1:T3CCP1:** Timer3 and Timer1 to CCP1/ECCP1 Enable bits
1x = Timer3 is the clock source for compare/capture CCP1 and ECCP1 modules
01 = Timer3 is the clock source for compare/capture of ECCP1,
Timer1 is the clock source for compare/capture of CCP1
00 = Timer1 is the clock source for compare/capture CCP1 and ECCP1 modules
bit 5-4 **T3CKPS1:T3CKPS0**: Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value
bit 2 **T3SYNC:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the system clock comes from Timer1/Timer3.)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.
bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T1CKI (on the rising edge after the first falling edge)
0 = Internal clock (FOSC/4)
bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

# PIC18 Interrupts:-

## Interrupt service routine

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the microcontroller runs the interrupt service routine. Generally, in most microprocessors, for every interrupt there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called the *interrupt vector table*. In the case of the PIC18, there are only two locations for the interrupt vector table, locations 0008 and 0018, as shown in Table 11-1. We will discuss the difference between these two in Section 11.6 when we cover interrupt priority.

**Table 11-1: Interrupt Vector Table for the PIC18**

| Interrupt | ROM Location (Hex) |
|---|---|
| Power-on Reset | 0000 |
| High Priority Interrupt | 0008 (Default upon power-on reset) |
| Low Priority Interrupt | 0018 (See Section 11.6) |

## Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

1. It finishes the instruction it is executing and saves the address of the next instruction (program counter) on the stack.
2. It jumps to a fixed location in memory called the interrupt vector table. The interrupt vector table directs the microcontroller to the address of the interrupt service routine (ISR).
3. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETFIE (return from interrupt exit).
4. Upon executing the RETFIE instruction, the microcontroller returns to the place where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then it starts to execute from that address.

## Sources of interrupts in the PIC18

There are many sources of interrupts in the PIC18, depending on which peripheral is incorporated into the chip. The following are some of the most widely used sources of interrupts in the PIC18:

1. There is an interrupt set aside for each of the timers, Timers 0, 1, 2, and so on. See Section 11.2.
2. Three interrupts are set aside for external hardware interrupts. Pins RB0 (PORTB.0), RB1 (PORTB.1), and RB2 (PORTB.2) are for the external hardware interrupts INT0, INT1, and INT2, respectively. See Section 11.3.
3. Serial communication's USART has two interrupts, one for receive and another for transmit. See Section 11.4.
4. The PORTB-Change interrupt. See Section 11.5.
5. The ADC (analog-to-digital converter). See Chapter 13.
6. The CCP (compare capture pulse-width-modulation). See Chapters 15 and 17.

## Enabling and disabling an interrupt

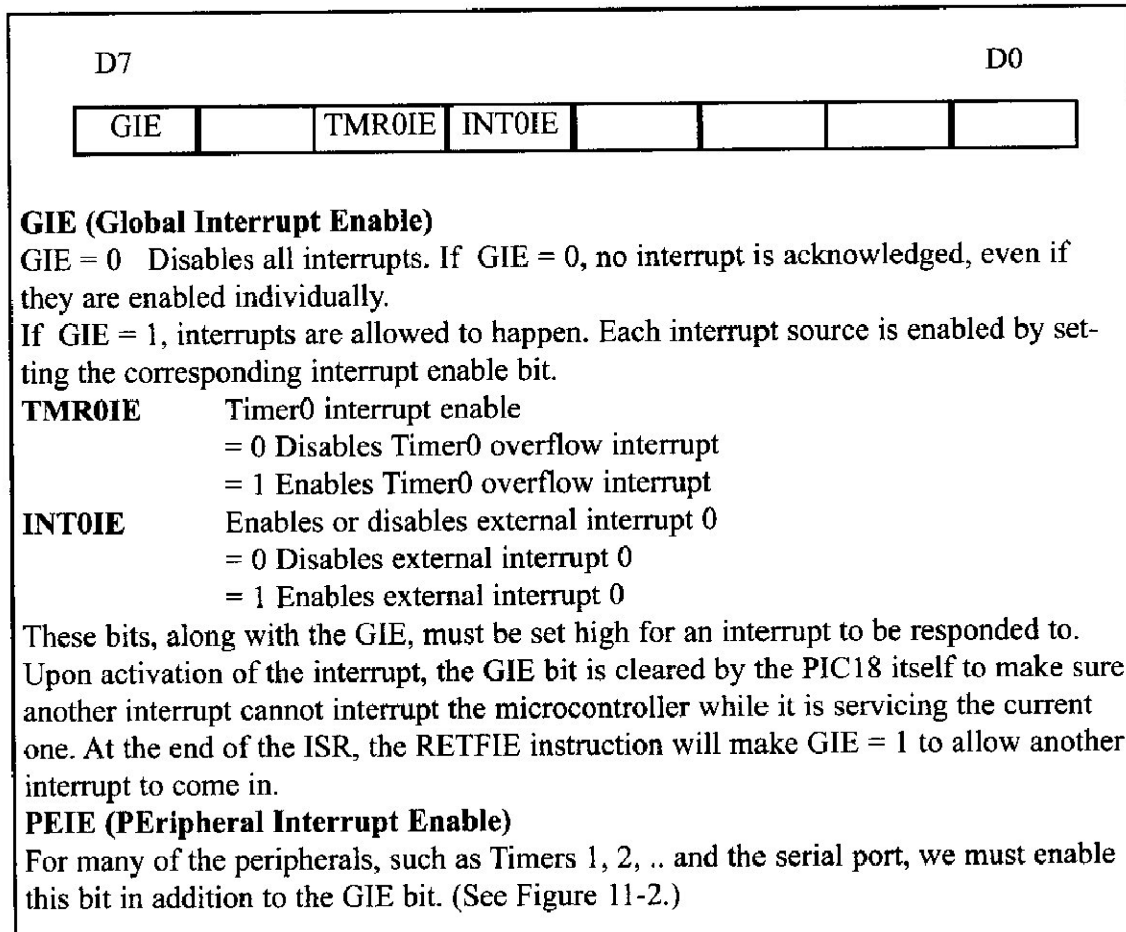
Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated. The interrupts must be enabled (unmasked) by software in order for the microcontroller to respond to them. The D7 bit of the INTCON (Interrupt Control) register is responsible for

enabling and disabling the interrupts globally. Figure 11-3 shows the INTCON register. The GIE bit makes the job of disabling all the interrupts easy. With a single instruction (BCF INTCON,GIE), we can make GIE = 0 during the operation of a critical task. See Figure 11-2.

## Steps in enabling an interrupt

To enable any one of the interrupts, we take the following steps:

1. Bit D7 (GIE) of the INTCON register must be set to HIGH to allow the interrupts to happen. This is done with the "BSF INTCON, GIE" instruction.
2. If GIE = 1, each interrupt is enabled by setting to HIGH the interrupt enable (IE) flag bit for that interrupt. Because there are a large number of interrupts in the PIC18, we have many registers holding the interrupt enable bit. Figure 11-2 shows that the INTCON has interrupt enable bits for Timer0 (TMR0IE) and external interrupt 0 (INT0IE). As we study each of peripherals throughout the book we will examine the registers holding the interrupt enable bits. It must be noted that if GIE = 0, no interrupt will be responded to, even if the corresponding interrupt enable bit is high. To understand this important point look at Example 11-1.
3. As shown in Figures 11-2 and 11-3, for some of the peripheral interrupts such as TMR1IF, TMR2IF, and TXIF, we have to enable the PEIE flag in addition to the GIE bit.




**GIE (Global Interrupt Enable)**
GIE = 0 Disables all interrupts. If GIE = 0, no interrupt is acknowledged, even if they are enabled individually.
If GIE = 1, interrupts are allowed to happen. Each interrupt source is enabled by setting the corresponding interrupt enable bit.

**TMR0IE** Timer0 interrupt enable
= 0 Disables Timer0 overflow interrupt
= 1 Enables Timer0 overflow interrupt

**INT0IE** Enables or disables external interrupt 0
= 0 Disables external interrupt 0
= 1 Enables external interrupt 0

These bits, along with the GIE, must be set high for an interrupt to be responded to. Upon activation of the interrupt, the GIE bit is cleared by the PIC18 itself to make sure another interrupt cannot interrupt the microcontroller while it is servicing the current one. At the end of the ISR, the RETFIE instruction will make GIE = 1 to allow another interrupt to come in.

**PEIE (PEripheral Interrupt Enable)**
For many of the peripherals, such as Timers 1, 2, .. and the serial port, we must enable this bit in addition to the GIE bit. (See Figure 11-2.)

**Figure 11-3. INTCON (Interrupt Control) Register**

Show the instructions to (a) enable (unmask) the Timer0 interrupt and external hardware interrupt 0 (INT0), and (b) disable (mask) the Timer0 interrupt, then (c) show how to disable (mask) all the interrupts with a single instruction.

**Solution:**

(a)
```
BSF INTCON,TMR0IE   ;enable(unmask) Timer0 interrupt
BSF INTCON,INT0IE ;enable external interrupt 1(INT0)
BSF INTCON,GIE ;allow interrupts to come in
```

We can perform the above actions with the following two instructions:

```
MOVLW B'10110000'   ;GIE = 1, TMR0IF = 1,INTIF0 = 1
MOVWF INTCON        ;load the INTCON reg
```

(b)
```
BCF INTCON,TMR0IE ;mask (disable) Timer0 interrupt
```

(c)
```
BCF INTCON,GIE      ;mask all interrupts globally
```

## External interrupts INT0, INT1, and INT2

There are three external hardware interrupts in the PIC18: INT0, INT1, and INT2. They are located on pins RB0, RB1, and RB2, respectively. See Figures 11-7 and 11-8. On default, all three hardware interrupts are directed to vector table location 0008H, unless we specify otherwise. They must be enabled before they can take effect. This is done using the INTxIE bit. The registers associated with INTxIE bits are shown in Table 11-3. For example, the instruction "`BSF INTCON, INT0IE`" enables INT0. The INT0 is a *positive-edge-triggered interrupt*, which means, when a low-to-high signal is applied to pin RB0 (PORTB.0), the INT0IF is raised, causing the controller to be interrupted. The raising of INT0IF forces the PIC18 to jump to location 0008H in the vector table to service the ISR. In Table 11-3, notice the INTxIF bits and the registers they belong to. Upon power-on reset, the PIC18 makes INT0, INT1, and INT2 rising (positive) edge-triggered interrupts. To make them falling (negative) edge-triggered interrupts, we must program the INTEDGx bits, as we will see shortly.

Examine Program 11-4 and its C version, Program 11-4C, to gain insight into external hardware interrupts.

**Table 11-3: Hardware Interrupt Flag Bits and Associated Registers**

| Interrupt (Pin) | Flag bit | Register | Enable bit | Register |
|---|---|---|---|---|
| INT0 (RB0) | INT0IF | INTCON | INT0IE | INTCON |
| INT1 (RB1) | INT1IF | INTCON3 | INT1IE | INTCON3 |
| INT2 (RB2) | INT2IF | INTCON3 | INT2IE | INTCON3 |

Program 11-4 connects a switch to INT0 and an LED to pin RB7. In this program, every time INT0 is activated, it toggles the LED, while at the same time data is being transferred from PORTC to PORTD.

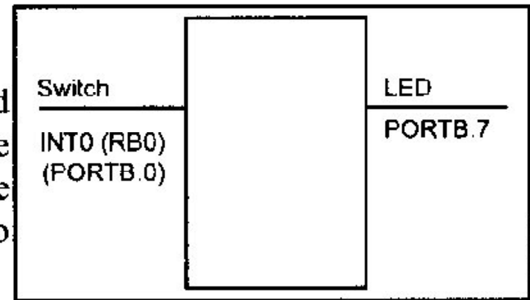



**Figure 11-9. For Program 11-4**

```
;Program 11-4
      ORG   0000H
      GOTO  MAIN               ;bypass interrupt vector table
;--on default all interrupts go to to address 00008
      ORG   0008H              ;interrupt vector table
      BTFSS INTCON,INT0IF      ;Did we get here due to INT0?
      RETFIE                   ;No. Then return to main
      GOTO  INT0_ISR           ;Yes. Then go INT0 ISR
;;--the main program for initialization
      ORG   00100H
MAIN BCF   TRISB,7             ;PB7 as an output
      BSF TRISB,INT0           ;make INT0 an input pin
      CLRF TRISD               ;make PORTD output
      SETF TRISC               ;make PORTC input
      BSF INTCON,INT0IE        ;enable INT0 interrupt
      BSF INTCON,GIE           ;enable interrupts globally
OVER  MOVFF PORTC,PORTD        ;send data from PORTC to PORTD
      BRA OVER                 ;stay in this loop forever
;-------------------------ISR for INT0
INT0_ISR
      ORG 200H
      BTG PORTB,7              ;toggle PB7
      BCF INTCON,INT0IF        ;clear INT0 interrupt flag bit
      RETFIE                   ;return from ISR
      END
```

## PORTB-CHANGE INTERRUPT

The four pins of the PORTB (RB4–RB7) can cause an interrupt when any changes are detected on any one of them. They are referred to as "PORTB-Change interrupt" to distinguish them from the INT0–INT2 interrupts, which are also located on PORTB (RB0–RB2). See Figure 11-15. The PORTB-Change interrupt has a single interrupt flag called RBIF and is located in the INTCON register. This is shown in Figure 11-14. In Figure 11-14, also notice the RBIE bit for enabling the PORTB-Change interrupt. In Section 11.3 we discussed the external hardware interrupts of INT0, INT1, and INT2. Notice the following differences between the PORTB-Change interrupt and INT0–INT2 interrupts:

(a) Each of the INT0–INT2 interrupts has its own pin and is independent of the others. These interrupts use pins PORTB.0 (RB0), PORTB.1 (RB1), and PORTB.2 (RB2), respectively. The PORTB-change interrupt uses all four of the PORTB pins RB4–PB7 and is considered to be a single interrupt even though it can use up to four pins.

(b) While each of the INT0–INT2 interrupts has its own flag, and is independent of the others, there is only a single flag for the PORTB-Change interrupt.

(c) While each of the INT0–INT2 interrupts can be programmed to trigger on the negative or positive edge, the PORTB-Change interrupt causes an interrupt if any of its pins changes status from HIGH to LOW, or LOW to HIGH. See Figure 11-16.

PORTB-Change is widely used in keypad interfacing as we will see in Chapter 12. Another way to use the PORTB-Change interrupt is shown in Program 11-8. In that program, we assume a door sensor is connected to pin RB4 and upon opening or closing the door, the buzzer will sound. See Figure 11-17.

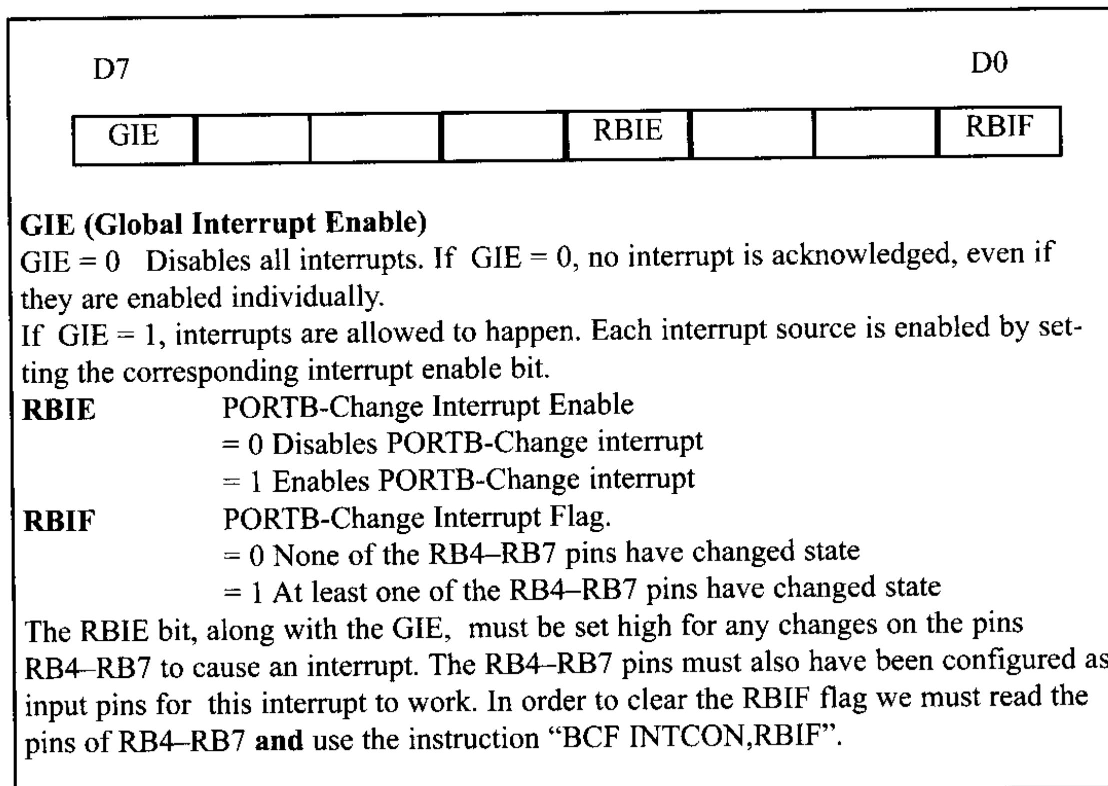
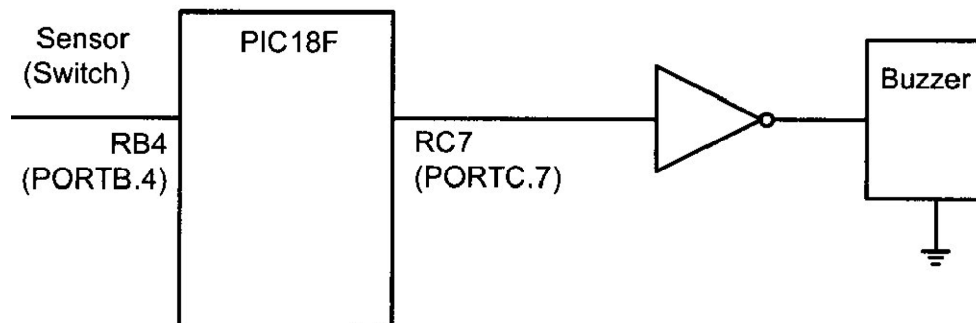



**Figure 11-14. INTCON (Interrupt Control) Register**

For Program 11-8 we have connected a door sensor to pin RB4 and a buzzer to pin RC7. In this program, every time the door is opened, it sounds the buzzer by sending it a square wave frequency.




```
;Program 11-8
MYREG EQU   0x20        ;set aside a couple of registers
DELRG EQU 0x80          ;for buzzer time delay
      ORG   0000H
      GOTO  MAIN        ;bypass interrupt vector table
;--on default all interrupts go to to address 00008
      ORG   0008H               ;interrupt vector table
      BTFSS INTCON,RBIF         ;Did we get here due to RBIF?
      RETFIE                    ;No. Then return to main
      GOTO  PB_ISR              ;Yes. Then go ISR
;--the main program for initialization
      ORG   00100H
MAIN  BCF   TRISC,7     ;PORTC.7 as an output for buzzer
      BSF   TRISB,4     ;PORTB.4 as an input for interrupt
      BSF INTCON,RBIE   ;enable PORTB-Change interrupt
      BSF INTCON,GIE    ;enable interrupts globally
OVER  BRA OVER          ;stay in this loop forever
;-------------------------ISR for PORTB-Change INT
PB_ISR
      ORG 200H
      MOVF  PORTB,W             ;we must read PORTB
      MOVLW D'250'              ;for delay
      MOVWF MYREG
BUZZ  BTG PORTC,7               ;toggle PC7 for the buzzer
      MOVLW D'255'              ;for delay
      MOVWF DELRG
DELAY DECF DELRG,F
      BNZ DELAY                 ;keep sounding the buzzer
      DECF MYREG,F
      BNZ BUZZ
      BCF INTCON,RBIF ;and clear RBIF interrupt flag bit
      RETFIE
      END
```

## INTERRUPT PRIORITY IN THE PIC18

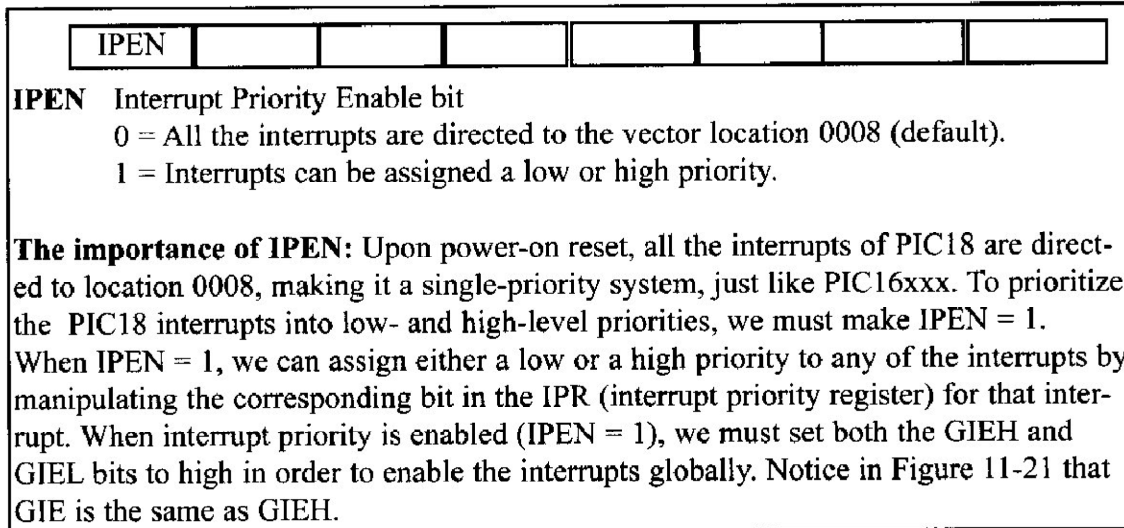
The next topic that we must deal with is what happens if two interrupts are activated at the same time? Which of these two interrupts is responded to first? Interrupt priority is the main topic of discussion in this section.

### Setting interrupt priority

In the PIC18 microcontroller, there are only two levels of interrupt priority: (a) low level, and (b) high level. While address 0008 is assigned to high-priority interrupts, the low-priority interrupts are directed to address 00018 in the interrupt vector table. See Table 11-5. Upon power-on reset, all interrupts are automatically designated as high priority and will go to address 00008H. This is done to make the PIC18 compatible with the earlier generation of PIC microcontrollers such as PIC16xxx. We can make the PIC18 a two-level priority system by way of programming the IPEN (interrupt priority enable) bit in the RCON register. Figure 11-19 shows the IPEN bit of the RCON register. Upon power-on reset, the IPEN bit contains 0, making the PIC18 a single priority level chip, just like the PIC16xxx. To make the PIC18 a two-level priority system, we must first set the IPEN bit to HIGH. It is only after making IPEN = 1 that we can assign a low priority to any of the interrupts by programming the bits called IP (interrupt priority). Figure 11-20 shows IPR1 (interrupt priority register) with the IP bits for TXIP, RCIP, TMR1IP, and TMR2IP. If IPEN = 1, then the IP bit will take effect and will assign a given interrupt a low priority. As a result of assigning a low priority to a given interrupt, it will land at the address 0018 instead of 0008 in the interrupt vector table. The IP (interrupt priority) bit along with the IF (interrupt flag) and IE (interrupt enable) bits will complete all the flags needed to program the interrupts for the PIC18. Table 11-6 shows the three flags and the registers they belong to for some of the interrupts used in this chapter. In Table 11-6, notice the absence of the INT0 priority flag. The INT0 has only one priority and that is high priority. That means all the PIC18 interrupts can be assigned a low or high priority level, except the external hardware interrupt of INT0. Study Figures 11-22 through 11-25 very carefully. When examining these figures, the following point must be noted. By making IPEN = 1, we enable the interrupt priority feature. Now we must also enable two bits to enable the interrupts: (a) We must set GIEH = 1. The GIEH bit is part of the INTCON register (Figure 11-21) and is the same as GIE, which we have used in previous sections. In this regard there is no difference between the priority and no-priority systems. (b) The second bit we must set high is GIEL (part of INTCON). Making GIEL = 1 will enable all the interrupts whose IP = 0. That means all the interrupts that have been given the low priority will be forced to vector location 00018H.

**Table 11-5: Interrupt Vector Table for the PIC18**

| Interrupt | ROM Location (Hex) |
|---|---|
| Power-on-Reset | 0000 |
| High-priority Interrupt | 0008 (Default upon power-on reset) |
| Low-priority Interrupt | 0018 (Selected with IP bit) |

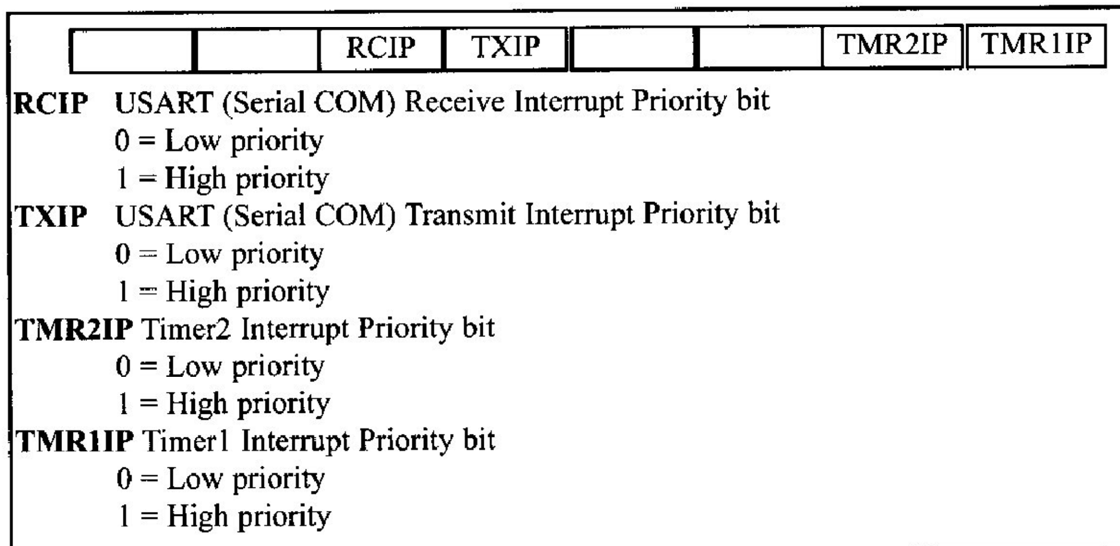


| IPEN | | | | | | | |
|---|---|---|---|---|---|---|---|

**IPEN** Interrupt Priority Enable bit
0 = All the interrupts are directed to the vector location 0008 (default).
1 = Interrupts can be assigned a low or high priority.

**The importance of IPEN:** Upon power-on reset, all the interrupts of PIC18 are directed to location 0008, making it a single-priority system, just like PIC16xxx. To prioritize the PIC18 interrupts into low- and high-level priorities, we must make IPEN = 1. When IPEN = 1, we can assign either a low or a high priority to any of the interrupts by manipulating the corresponding bit in the IPR (interrupt priority register) for that interrupt. When interrupt priority is enabled (IPEN = 1), we must set both the GIEH and GIEL bits to high in order to enable the interrupts globally. Notice in Figure 11-21 that GIE is the same as GIEH.

**Figure 11-19. RCON Register. IPEN Allows Prioritizing the Interrupt into 2 Levels**



| | | RCIP | TXIP | | | TMR2IP | TMR1IP |
|---|---|---|---|---|---|---|---|

**RCIP** USART (Serial COM) Receive Interrupt Priority bit
0 = Low priority
1 = High priority

**TXIP** USART (Serial COM) Transmit Interrupt Priority bit
0 = Low priority
1 = High priority

**TMR2IP** Timer2 Interrupt Priority bit
0 = Low priority
1 = High priority

**TMR1IP** Timer1 Interrupt Priority bit
0 = Low priority
1 = High priority

**Figure 11-20. IPR1 Peripheral Interrupt Priority Register 1**

**Table 11-6: Interrupt Flag Bits for PIC18 Timers**

| Interrupt | Flag bit (Register) | Enable bit (Register) | Priority (Register) |
|---|---|---|---|
| Timer0 | TMR0IF (INTCON) | TMR0IE (INTCON) | TMR0IP (INTCON2) |
| Timer1 | TMR1IF (PIR1) | TMR1IE (PIE1) | TMR1IP (IPR1) |
| Timer2 | TMR2IF (PIR1) | TMR2IE (PIE1) | TMR2IP (IPR1) |
| Timer3 | TMR3IF (PIR3) | TMR3IE (PIE2) | TMR3IP (IPR2) |
| INT1 | INT1IF (PIR1) | INT1IE (PIE1) | INT1IP (INTCON3) |
| INT2 | INT2IF (PIR1) | INT2IE (PIE1) | INT2IP (INTCON) |
| TXIF | TXIF (PIR1) | TXIE (PIE1) | TXIP (IPR1) |
| RCIF | RCIF (PIR1) | RCIE (PIE1) | RCIP (IPR1) |
| RB INT | RBIF (INTCON) | RBIE (INTCON) | RBIP (INTCON2) |

*Note*: INT0 has only the high-level priority.