COVENTRY UNIVERSITY

Faculty Of Engineering, Environment & Computing
School of CEM

PIC18 Microcontroller Introduction to Interrupts

Introduction
In this laboratory session we will explore the concept of Interrupts and we will carry out some experimentation in their use.
An interrupt is a hardware signal event whereby the normal program sequence that the micro is currently executing is "interrupted" by an external/internal hardware signal. The micro stops whatever was doing, saves the appropriate current information to memory and services this particular request by vectoring (going to) a specific location to execute this interrupt service routine. On completion of this interrupt service routine the program returns back to the point of interruption by retrieving previously saved information.

Background
A microcontroller can serve a number of devices. There are two different types by which the micro can respond to a service request from these devices. One of these methods is **polling** whereby the micro continuously monitors the status of the various devices in a round-robin fashion. When the status condition is met the service is performed. In this method there is no priority of service involved. In many instances a lot of the micro time is wasted by taking time in polling devices that they do not need service.
The second method in responding to a device request for service is **interrupts** as already mentioned. The advantage of interrupts is that the micro can quickly respond to a service request, can serve many devices quicker and also provide attention based on a priority that has been assigned to the device.
Every interrupt must have an **Interrupt Service Routine** (ISR), or **Interrupt Handler**.
For each interrupt there is a fixed memory location that holds the address of this ISR. The locations are called **Interrupt Vectors**. The PIC18 has two Interrupt vector locations at hex 0008 and 0018 as shown in Table 1.

Interrupt	**Vector Location**
Reset	0x00
High Priority Interrupt	0x08
Low Priority Interrupt	0x18

Table 1 Interrupt Vector Table

PIC18 sources of Interrupts
The following are sources of interrupt:

1. External edge triggered interrupt on INT0, INT1 and INT2 pins (RB0 , RB1 and RB2 which are pins of Port B 1,2 and 3)
2. Port B pins change interrupts (change of state on RB4 – RB7 pins)
3. Timer 0 overflow interrupt
4. A/D interrupt on completion of a conversion process
5. Asynchronous Serial Port (USART) receive and transmit interrupts
6. Synchronous transmit interrupt
7. Timer 1, Timer 2 and Timer 3 overflow interrupts

8. Compare Capture 1 and 2 ,CCP1 and CCP2 (pulse width modulation) interrupts
9. Parallel Slave Port PSP read/write interrupt
10. Comparator interrupt

(Depending on the PIC18 family member the micro could have many more interrupts).

A simplified logic diagram of an interrupt system

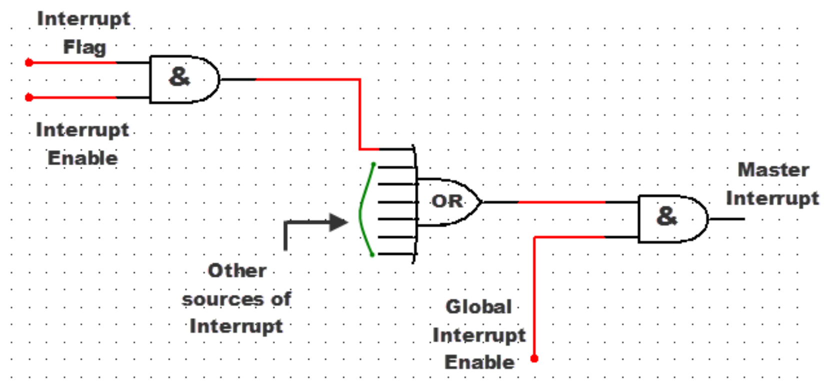


Fig.1 Simplified interrupt system diagram

When a peripheral requires attention from the micro it sets an **Interrupt Flag**. For each flag there is a corresponding interrupt enable bit which determines if the interrupt is to be recognized or not. This enable bit is ANDed with the interrupt flag allowing only the selected bit to cause an interrupt. All the Interrupt flags and interrupt enable bits from other peripheral devices are ORed together and then ANDed with a **Global Interrupt Enable** input that further allows or disallows a Master Interrupt to the mcicro.

PIC18 interrupts can be divided into two groups, **High Priority** and **Low Priority.**
Applications that need more attention will be found in the high priority group. These can stop any low priority interrupt in progress. Low priority interrupts cannot be stopped by a low priority interrupt. If our application does not need priority settings we can choose to disable the priority settings of the interrupt scheme. As we can see from Table 1 high priority interrupts vector to address 0x08 and low priority interrupts vector to address 0x018.

Steps involved in the execution of an interrupt

When an interrupt is activated the PIC micro will go through the following steps:

- It completes the instruction that is currently executing saving the address of next program instruction on the stack.
- It will then jump to the interrupt vector table. This vector table contains a **GOTO** instruction that directs the micro to the address of the interrupt service routine (ISR).
- Goes to the ISR location and starts executing the instructions of this interrupt service routine. The last instruction of the ISR is an RETFIE , (RETurn From Interrupt Exit).
- Executing the RETFIE instruction takes the micro back to the location where it was interrupted. This is achieved by retrieving the program counter address that was previously saved in the stack and so continues with the instruction following the point of interruption.

Enabling and disabling interrupts

After a processor reset all system interrupts are disabled. The word **Masked** is used to define that the interrupts are disabled. In this masked state there will be no response to interrupts.
To enable the processor to respond to interrupts we will need to enable (**unmask**) the registers associated with interrupt system.

PIC18 registers that control interrupt operations

The following registers are used to control the various functions of the PIC18 interrupt system:

• INTCON	**INT**errupt **CON**trol register 1
• INTCON2	
• INTCON3	
• RCON	**R**eset **CON**trol register
• PIR1 , PIR2	**P**eripheral **I**nterrupt **R**equest (Flag) Register(s)
• PIE1 , PIE2	**P**eripheral **I**nterrupt **E**nable **R**egister(s)
• IPR1, IPR2	peripheral **I**nterrupt **P**riority **R**egister(s)

Each interrupt source, (excluding INT0) has 3 bits that control its operation. These are the following:

- A flag that shows if an interrupt has occurred (**I**nterrupt **F**lag IF).
- An **I**nterrupt **E**nable bit (IE) that allows the enabling or disabling the interrupt source.
- An **I**nterrupt **P**riority bit (IP) that selects a high or low priority of the interrupt.

We will only examine some of the settings of some of these registers and their function as the main focus of this exercise it will be based around external interrupts. Other peripheral interrupts will be discussed at later sessions when we examine Timers, Synchronous and Asynchronous serial communications and comparators.

The Interrupt Control Register INTCON

The INTCON register is a readable and writable register containing various enable, priority and flag bits. The interrupt flag bits are set when an interrupt condition occurs. These flags can be read by the micro and be used in a polling way. The user must make sure that the interrupt flag bits are clear before enable an interrupt.

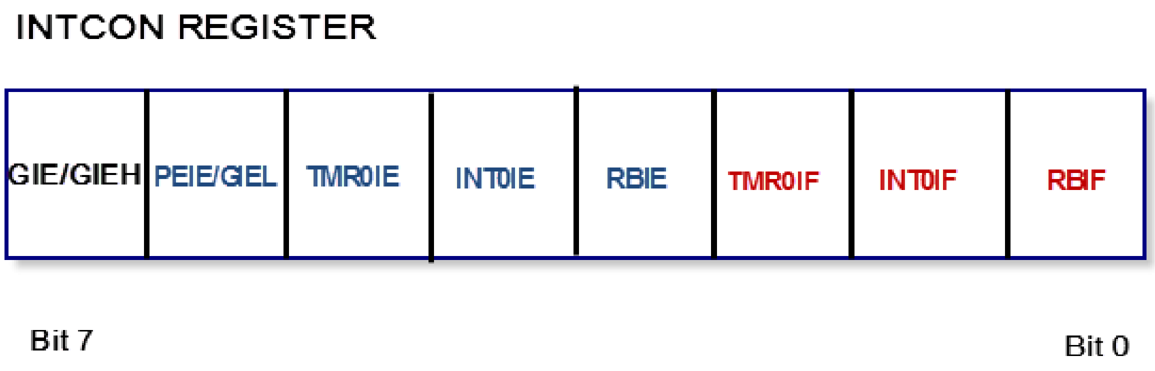


Fig.2 INTCON Register

Bit 7 of the INTCON register is the **G**lobal **I**nterrupt **E**nable bit (GIE) that allows interrupts to globally happen. In order to allow interrupts this bit has to be logic 1.
When GIE = 1 each interrupt is enabled by setting to logic 1 the corresponding IE (Interrupt Enable) flag in the various registers. Remember that we have a large number of these. In the case of the INTCON register above we can see three of these IE registers namely:

TMR0IE : Timer 0 Interrupt Enable

INT0E : External Interrupt Enable
RBIE : Port B Change Interrupt Enable

As we continue with the study of the various PIC18 peripherals we will be examining in more detail the various registers that hold the corresponding interrupt enable bits. We have to remember that in order to enable any of our PIC18 peripherals interrupts the corresponding IE flag has as well as the GIE have to be set to 1. A look at Fig.1 will make this clearer to understand.

Note: Just to slightly complicate the above some peripherals such as the serial port also need an extra setting in the PEIE (**PE**ripheral **I**nterrupt **E**nable).
A full breakdown all 8 bits of INTCON register is as follows:

Bit 7 **GIE/GIEH**: **G**lobal **I**nterrupt **E**nable Bit (Please note this works in conjunction with IPEN **I**nterrupt **P**riority **EN**able bit in the RCON register, see later).

<u>When IPEN = 0;</u>
1 = Enables all unmasked interrupts
0 = Disables all interrupts

<u>When IPEN = 1;</u>
1 = Enables all high priority interrupts
0 = Disables all interrupts

Bit 6 **PEIE/GIEL:** **PE**ripheral **I**nterrupt **E**nable bit

<u>When IPEN = 0;</u>
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

<u>When IPEN = 1;</u>
1 = Enables all low –priority peripheral interrupts
0 = Disables all low-priority peripheral interrupts

Bit 5 **TMR0IE:** Timer 0 (**TMR0)** Overflow **I**nterrupt **E**nable bit
1 = Enables TMR0 overflow interrupt
0 = Disables TMR0 overflow interrupt

Bit 4 **INT0IE:** External interrupt **INT0** **E**nable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

Bit 3 **RBIE:** **RB** (Port B) Port Change **I**nterrupt **E**nable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

Bit 2 **TMR0IF:** Timer 0 (**TMR0**) overflow **I**nterrupt **F**lag bit
1 = TMR0 has overflowed (it must be cleared in your code)
0 = TMR0 has did not overflow

Bit 1 **INT0IF:** **INT0** External **I**nterrupt **F**lag bit
1 = The INT0 external interrupt occurred (must be cleared in your code)
0 = The INT0 external interrupt did not occur

Bit 0 **RBIF:** **RB** (PortB) port change **I**nterrupt **F**lag
1 = At least one of PortB pins 4-7 changed state (needs to be cleared in code)
0 = No PortB pins 4 -7 have changed state.

Reset Control Register RCON

IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$

Fig. 3 RCON Register

The **R**eset **CO**Ntrol register RCON contains flag bits that are used to identify the source of the last reset or wake-up from idle or sleep mode. One of the other important bits that have already been mentioned is the Interrupt Priority Enable bit IPEN (bit 7). Following a CPU reset the state of the IPEN bit is at logic 0. In this case all sources of interrupt service requests cannot be prioritized. When the CPU is interrupted, execution of the program will be transferred to the hex location 0x008. This is the **Default interrupt Vector** location. By setting IPEN to logic 1 all sources of interrupt request (apart from INT0) can be configured as either High or Low priority. For an enabled source of an interrupt request the response will depend on the priority assigned as follows:

If the service request is of High priority the execution will be transferred to the **High Priority Interrupt Vector** at hex location **0x008**.
If the service request is of Low priority execution will be transferred to the instruction located at the **Low Priority Interrupt Vector** at hex location **0x018**.
The interrupt priority is enabled when (IPEN = 1). There are two register bits that enable interrupts globally. By setting the GIEH bit 7 in the INTCON register (GIEH =1) will enable all interrupts that have the priority bit set as high priority. Setting the GIEL bit (INTCON bit 6) enables all interrupts that have the priority bit cleared as low priority. When IPEN is cleared the interrupt priority feature of the micro is disabled. All the priority settings in the interrupt priority registers (see later) have no effect. All interrupts branch to hex location 0x008.

In discussing further the priority interrupts we will discuss the following points:

For a high priority interrupt to be processed by the CPU the following has to be satisfied:

1. The interrupt enable bit of the source of the interrupt must be enabled in the corresponding bit of the appropriate register.

2. The interrupt flag of the interrupt source must be cleared (logic 0).

3. The priority bit of the source must be set (logic 1) in the appropriate register.

4. The Global interrupt bit in the INTCON register must be set (logic 1).

For a low priority interrupt to be processed by the CPU the following has to be satisfied:

1 to 2 as for the high priority interrupts. Step 3 will need the priority bit must be cleared, the low priority interrupt must be enabled by setting bit GIEL (Global Interrupt Enable Low) bit 6 in the INTCON to logic 1. The global interrupt enable bit GIEH of INTCON must be set (logic 1).

The remaining bits of the RCON register are beyond the scope of our current session. For a further explanation of their function please refer to the Appendix.

The following activity diagram shows the how an external internal of high or low priority will proceed through the appropriate **I**nterrupt **S**ervice **R**outine and it's dependency on the various interrupt enable and interrupt status flags.

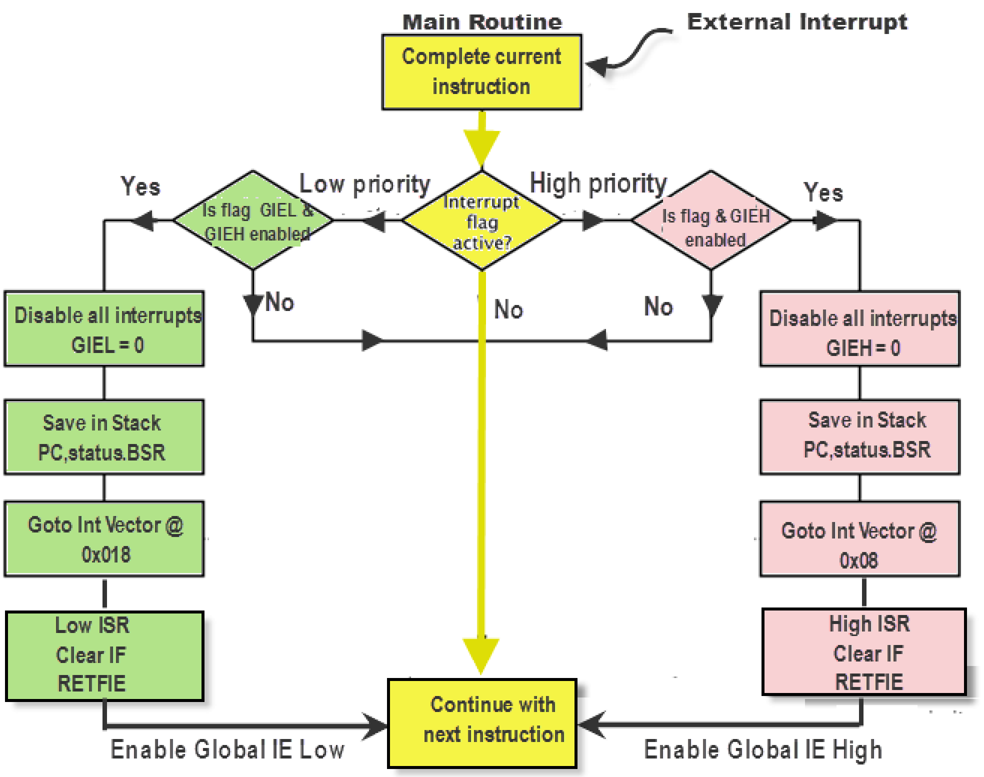


Fig. 4 Activity diagram shows interrupt priority paths

Programming external hardware interrupts

In this laboratory session we focus upon the use of external hardware interrupts. The PIC18 has three hardware interrupts namely INT0, INT1 and INT2. These are through PORTB pins RB0 , RB1 and RB2 see Fig.5.

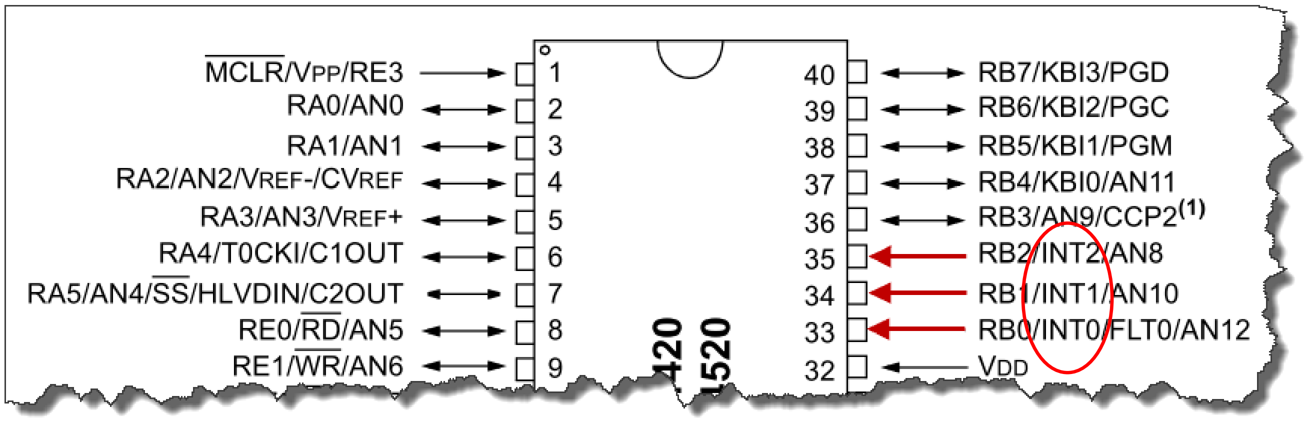


Fig. 5 INT0, INT1 and INT2 on RB port

By default all three interrupts are directed to vector location h'0008. In order for the interrupts to be recognized they will have to be enabled. To do this will we have to set the appropriate bits in the corresponding registers. These are the INTCON register bit 4 INT0IE that enables/disables external interrupt 0 , and the INTCON3 bits 3 and 4, (INT1IE and INT2IE). Table 2 shows the register and their bits associated with these external interrupts.

Interrupt (Port Pin)	Associated Flag	Enable bit	Associated Register
INT0 (RB0)	INT0IF	INT0IE	INTCON
INT1 (RB1)	INT1F	INT1IE	INTCON3
INT2 (RB2)	INT2IF	INT2IE	INTCON3

Table 2 External hardware interrupts and registers

Upon Reset External interrupt INT0, INT1 and INT2 are +ve edge triggered interrupts. This means that when a Low-to-High signal ↑ is applied to Port B pin B0 will set interrupt flag INT0IF. If the interrupt enable bit INT0IE in the INTCON register is enabled (set to logic 1) the positive logic change on RB0 will force the micro to jump to vector interrupt location 0x008 and execute the Interrupt Service Routine (ISR).
In order to change INTx to a –ve edge response the corresponding **INT**errupt **EDG**e register , **INTEDG** bits must be set. This will be further explained when we further examine the various other interrupt associated registers and during the experimentation tasks that will follow.

There is an extra set of external interrupts available on Port B, the **Keyboard Interrupts** or **PORTB –Change** interrupts KB0, KB1, KB2 and KB3. These interrupts act on a change of the input signal connected to these pins. The PORTB-Change interrupt has a single flag called RBIF and is located in bit 0 of the INTCON register (see Fig. 2).

The differences between INTx and KBx interrupts are as follows:

External interrupts INT0, INT1 and INT2 on Port B (RB0-RB2) can be set to trigger on either rising or falling edge but not both. These are separate interrupts and we can easy indentify the source as they have their own interrupt flags (INTxIF).

External Keyboard interrupt uses Port B (RB4-RB7) and is considered to be a single interrupt even though it is can use up to four pins. There is only a single flag for the KBx interrupt (RBIF).
A KBx interrupt can respond to any pin change either +ve or –ve.

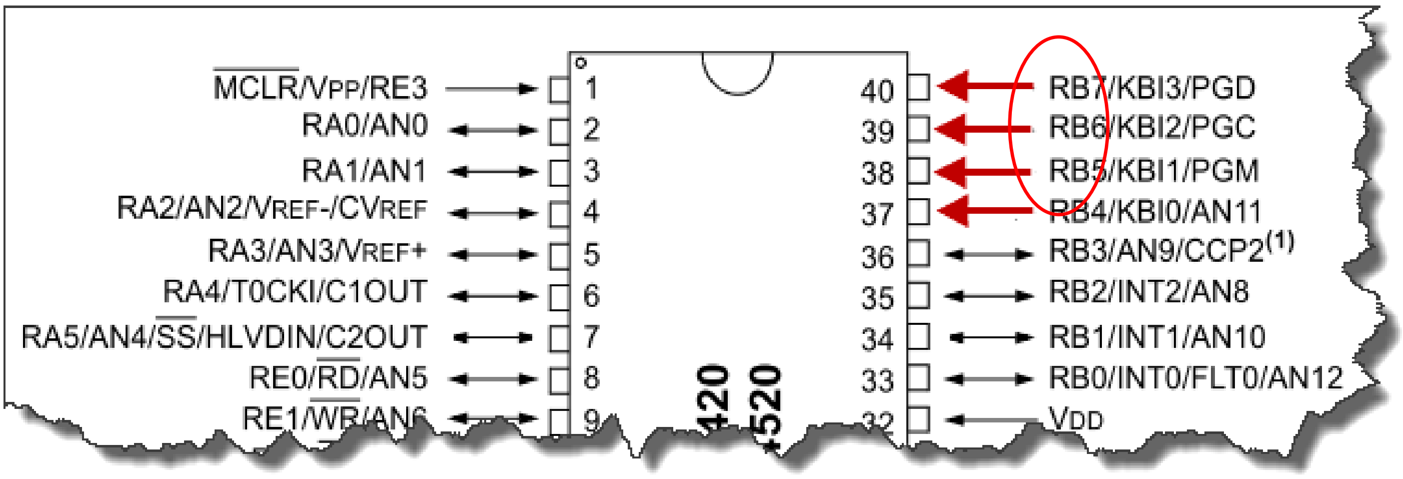


FIG.6 Keyboard or PortB - Change Interrupt

Interrupt Control Registers 2 and 3 (INTCON2 and INTCON3)

The settings of the INTerrupt CONtrol register 2 and three are shown in Figures 6 and 7.
A further discussion of the various settings will be examined in the experimentation that will follow.

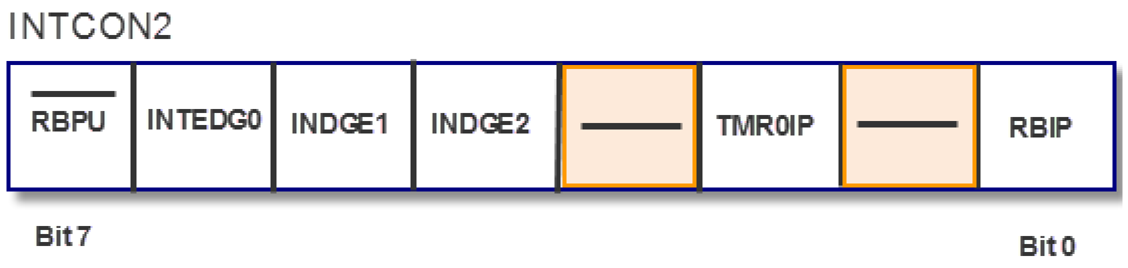


Fig. 7 Interrupt Control Register 2

Bit 7	**RBPU**:	**RB** (PORTB) **P**ull-**U**p Enable bit 1 = All PORTB pull-ups are disabled 0 = PORTB pull-ups enabled by individual port latch values
Bit 6	**INTEDG0**:	External **INT**errupt 0 (INT0) **EDG**e Select bit 1 = Interrupt on rising edge 0 = Interrupt on falling edge
Bit 5	**INTEDG1**:	External **INT**errupt 1 (INT1) **EDG**e Select bit 1 = Interrupt on rising edge 0 = Interrupt on falling edge
Bit 4	**INTEDG2**:	External **INT**errupt 2 (INT2) **EDG**e Select bit 1 = Interrupt on rising edge 0 = Interrupt on falling edge

Bit 3		*Not used (read as 0)*
Bit 2	**TMR0IP**:	**TMR0** (Timer 0) Overflow **I**nterrupt **P**riority bit 1 = High priority 0 = Low priority
Bit 1		*Not used (read as 0)*
Bit 0	**RBIP:**	**RB** Port Change **I**nterrupt **P**riority bit 1 = High priority 0 = Low priority

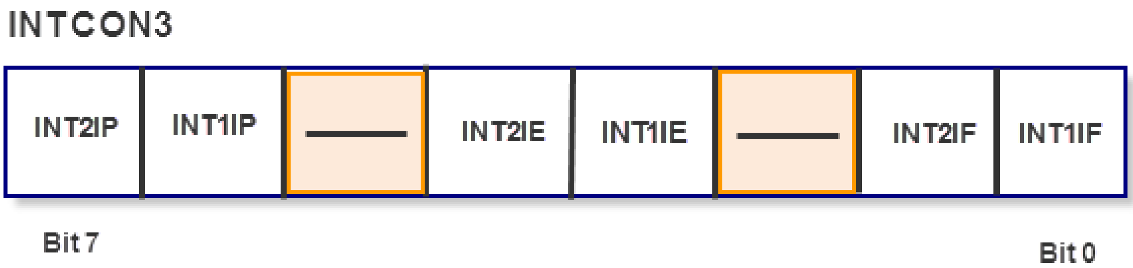


Fig. 8 Interrupt Control Register 3

Bit 7	**INT2IP**:	External Interrupt 2 **INT2** **I**nterrupt **P**riority control 1 = High priority 0 = Low priority
Bit 6	**INT1P**:	External Interrupt 1 **INT1** **I**nterrupt **P**riority control 1 = High priority 0 = Low priority
Bit 5		*Not used (read as 0)*
Bit 4	**INT2IE**:	External Interrupt 2 **INT2** **I**nterrupt **E**nable bit 1 = Enables the INT2 external interrupt 0 = Disables the INT2 external interrupt
Bit 3	**INT1IE**:	External Interrupt 1 **INT1** **I**nterrupt **E**nable bit 1= Enables the INT1 external interrupt 0 = Disables the INT1 external interrupt
Bit 2		Not used (read as 0)
Bit 1	**INT2IF**:	External Interrupt **INT2** **F**lag bit 1 = The INT2 external interrupt occurred (needs to be cleared in your code) 0 = The INT2 external interrupt did not occur

Bit 0	**INT1IF**:	External Interrupt **INT1 F**lag bit
		1 = The INT1 external interrupt occurred (needs to be cleared in your code)
		0 = The INT1 external interrupt did not occur

EXPERIMENTATION

Apparatus

PC with MPLAB V5.6 or higher and USB port
ICD 3 and USB lead
5 volt fixed PSU and connecting leads
PIC18F4520 Target board
PIC18 Prototyping board and 10way ICD connector, 2-way connecting lead for Port B
8-LED Bargraph display
3 x 4k7 resistor (already onboard the switch PCB)
1 x 3 push button switch outboard
Connecting wire links
8-way shorting header

The following diagram in Fig. 9 shows the system components and connections for the experimentation that follows. We will be examining the various external hardware PORT B interrupts. This session will not cover any PIC18 peripheral interrupts as these will be dealt with corresponding PIC18 peripherals study.

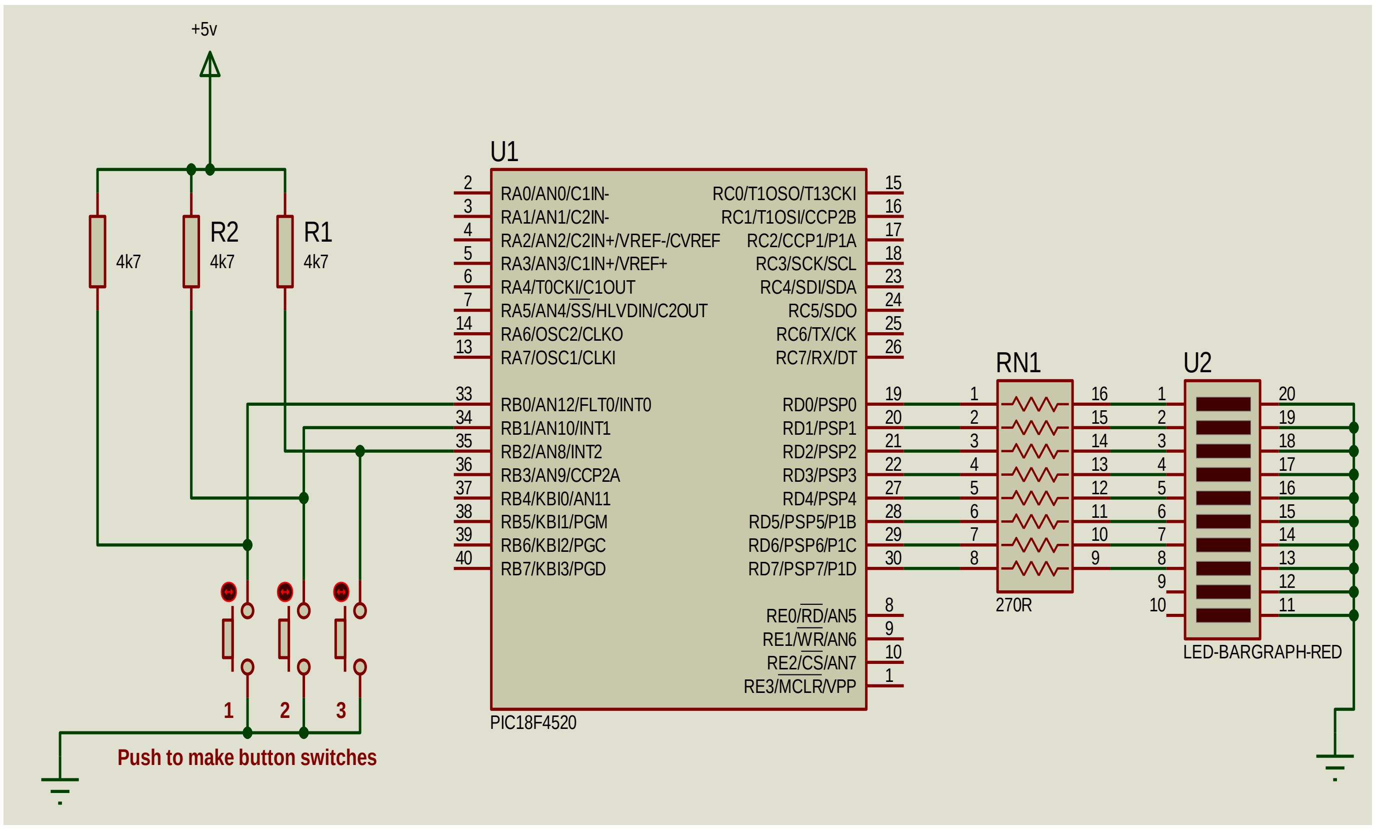


Fig. 9 System connections for interrupt experimentation

Example 1
The example that follows demonstrates the use of and external PortB INT0 interrupt. Pin RB0 of the port is connected to a **N**ormally **O**pen (NO) push to make switch SW1. The other pin of the switch is connected to ground. Pin RB0 is pulled-up to logic 1 via a 4k7 resistor so that RB0 (INT0) is normally held at logic 1. The various configuration bits in the various interrupt control registers are appropriately set/cleared so that the –ve (falling) transition on the INT0 pin. Our settings are such that the CPU responds to a falling edge high priority interrupt. The normal sequence of the program is to use PORT D as an output port displaying incrementing values with the appropriate time delay. As soon as the external interrupt switch SW1 is momentarily pressed, the interrupt routine should reset the counting sequence to zero. On return from the interrupt service routine the program starts counting on Port D from zero again.

The settings of the various interrupt registers are as follows:

INT0 is a high priority interrupt (interrupt vector @ location 0x008).
For using RB0 as INT0 you will have to set the TRISB0 register to input. (BSF TRISB,INT0)

1. ADCON 1 is used to control the voltage reference of the ADC. It is also used to specify the use of I/O functions of the PIC18. Register bits PCFG0- 3 specify the function of Ports A, B and E pins as analogue or digital I/O (see page 7 of the PIC18 ADC laboratory notes).
   By setting PCFG0 – 3 to 0111 Port B bits 0 to 4 (AN8- AN12) are configured as digital I/O. See figures 9 and 10.

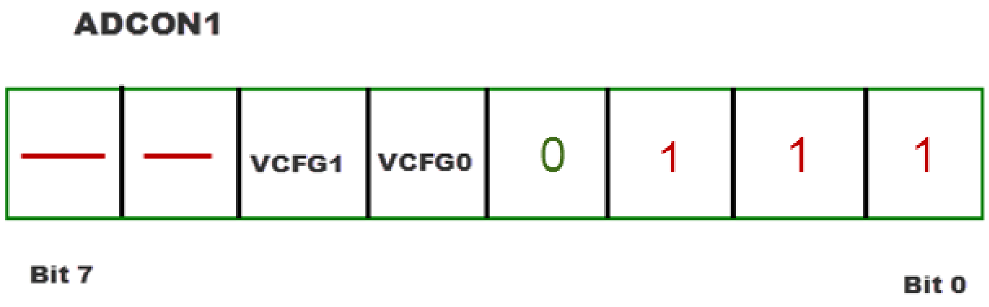
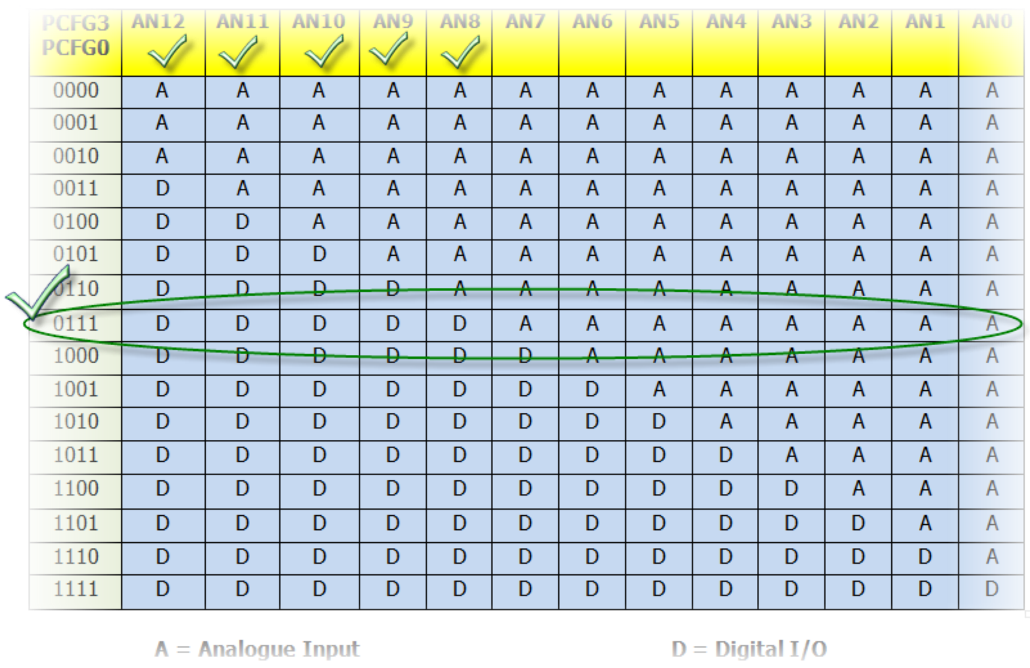


Fig.10 ADCON1 setting for configuring RB0 – RB4 to digital I/O



PCFG3	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
PCFG0	✓	✓	✓	✓	✓								
0000	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
✓ 0111	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analogue Input D = Digital I/O

Fig. 11 PCFG0-PCFG3 bit setting for analogue or digital port function

The following steps are required for configuring associated interrupt registers for extrenal INT0 interrupts:

2. INTCON register bits 5 and 7 should be set as follows:
Bit 5 is the INTerrupt 0 Enable bit. This should be set to 1 to enable interrupt from RB0.
Bit 7 is the Global Interrupt Enable bit. In order to globaly enable interrupts we will need to set this to logic1. See Fig.12 .
Associated assembly instructions for doing this:

```
BSF INTCON,INT0IF                    (BitSet Flag INT0IE in INTCON register)
BSF INTCON,GIE                       (BitSet Flag GIE in INTCON register)
```

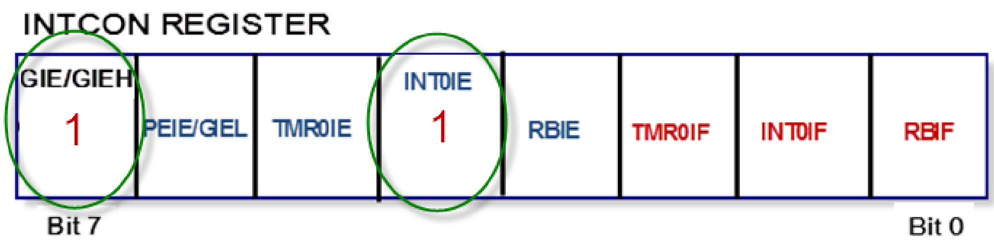


Fig. 12   Settings of INT0IE and GIE bits for enabling INT0 interrupts

3. When an INT0 occurs we will need to check the external interrupt flag bit for INT0.
If it is logic one the interrupt source was due to INT0. (bit 2 set to logic 1 see Fig.12).

Assembly code for this:

```
BTFSS          INTCON,INT0IF   ( Bit Test Skip if Set followed by the decision making code)
```

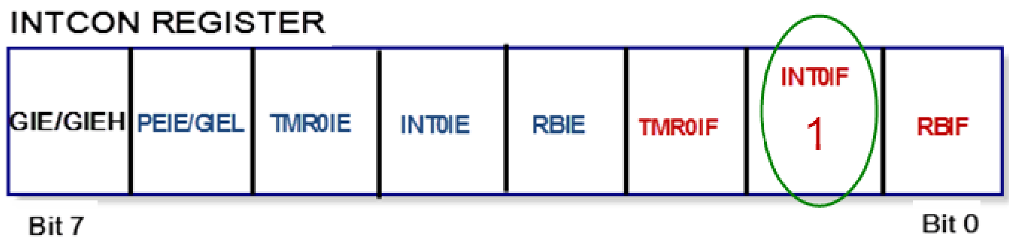


Fig.13   When INT0IF is set it indicates a transition on RB0 pin

4. During the interrupt service routine the INT0IF has to be cleared so that further INT0 interrupts are NOT recognized. (BCF INTCON,INT0IF)

<u>System connections for experiment</u>
System connections for the experimentation that follows are as shown in Fig.14.
Refer to the schematic diagram as shown in Fig.9 for connections detail.

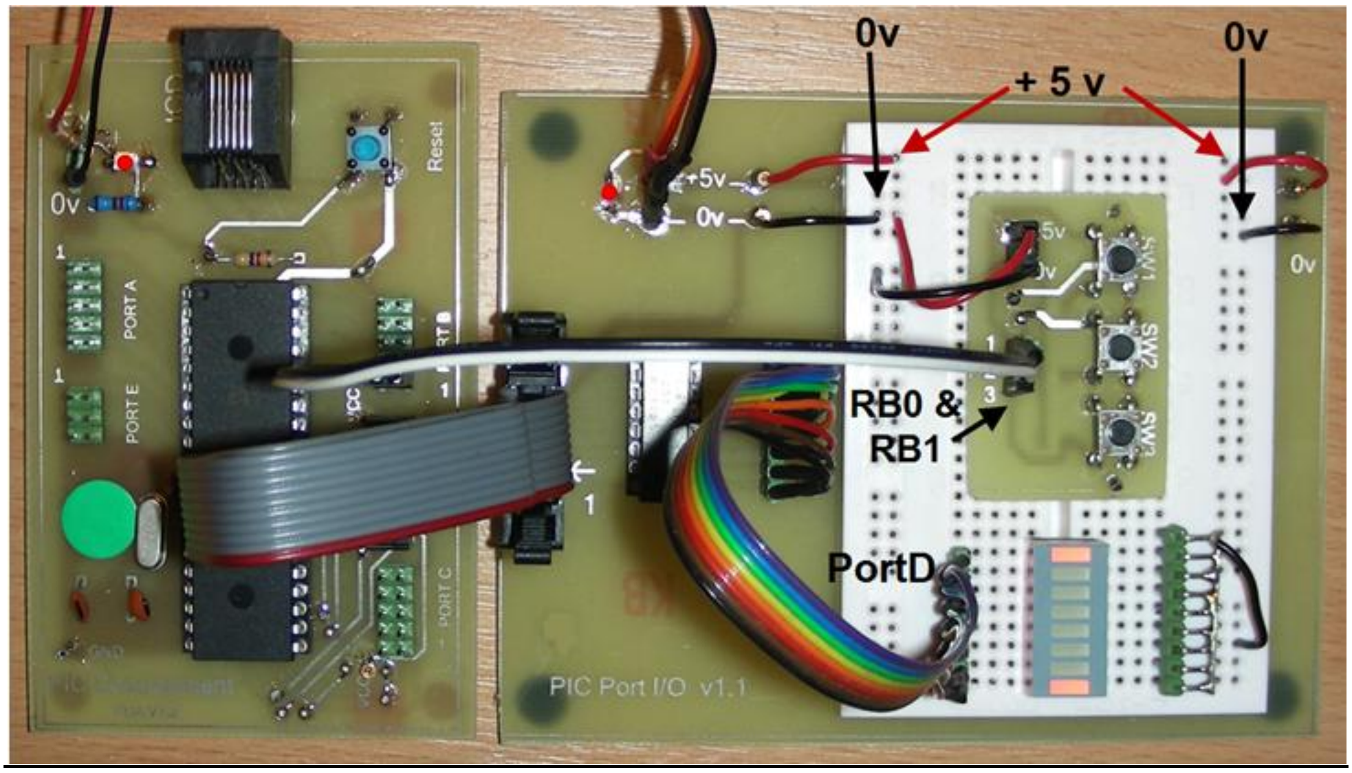


Fig.14 System connections screenshot

<u>Interrupt Example 1 Assembler version</u>: For your reference

```
;*****************************************************************
;
;    Filename:         Interrupt Example.asm
;    Date:              22/10/10
;    File Version:      1.0
;
;    Author:           PDA
;    Company:          Coventry University
;
; Program Function:  Interrupt Example PIC18F4520
; The following example shows the use of an external interrupt INT0. The main routine uses Port D as output
; displaying incrementing values on the Port with the appropriate delay. A normally open
; push to make switch provides a temporary high low to transition on RB0. The appropriate interrupt associated registers
; are set so that when the switch is pressed the signal transition is recognised. On interrupt the micro will vector to the ISR
; (interrupt service routine). The function of the ISR is to reset the contents of Port D.
;
;*******************************************************************
;                                                         *
;    Files Required: P18F4520.INC                         *
;                                                         *
;*******************************************************************

        LIST P=18F4520              ; Directive to define processor
        #include <P18F4520.INC>     ; Processor specific variable definitions

;*********************************************************************
```

```
; CONFIGURATION BITS
; (Microchip has changed the format for defining the configuration bits, please
; see the .inc file for further details on notation).  Below are a few examples.
;
;
;
;   Oscillator Selection and other fuse settings:

        CONFIG    OSC      =  HS              ; High Speed clock
        CONFIG    MCLRE    =  ON              ; MCLR enabled
        CONFIG    DEBUG    =  OFF             ; Background debugger disabled, RB6 and RB7
        CONFIG    LVP      =  OFF             ; Low Voltage Programming OFF
        CONFIG    WDT      =  OFF             ; WDT disabled
;************************************************************************


        ORG             0x0000                  ; ORG Directive

DELVAL1         EQU             0x0300                  ; Reserve location for Delay1 value
DELVAL2         EQU             0X0320                  ; and Delay value2

                GOTO            Main                    ; Go to start of main code

                ORG             0x008                   ; High Priority interrupt vector
                BTFSS           INTCON,INT0IF           ; Test INT0 Interrupt Flag if set, ie was interrupt from INT0?
                RETFIE                                  ; No, RETurn From Interrupt
                GOTO            INT0_ISR                 ; Yes, Go to Interrupt 0 service routine for INT0

                ORG             0x0100                  ; Main program starts here
Main            CLRF            TRISD                   ; Port D O/P  (LED display)
                MOVLW           0X0F                    ; Set RB0 -4 as digital I/O pins
                MOVWF           ADCON1                  ; Note that reset default sets these as analogue
                BSF             TRISB,INT0              ; make INT0 an input pin
                BSF             INTCON,INT0IE           ; Enable INT0 interrupt
                BSF             INTCON,GIE              ; Enable Global interrupts
LOOP            INCF            LATD                    ; Increment content of PortD
                CALL            DELAY
                GOTO            LOOP


; *********  Interrupt service routine   ********

INT0_ISR        ORG             0X200                   ; Interrupt service routine in here
                BCF             INTCON,INT0IF           ; Clear INT0 Interrupt Flag
                CLRF            LATD                    ; Clear PortD
                RETFIE                                  ; Return from interrupt

;  ********   Delay subroutine *************

DELAY           MOVLW           0XFF                    ; Move literal hex FF into W Reg
                MOVWF           DELVAL1                 ; Move this W register value now to DELVAL location
REPEAT          MOVLW           0xFF                    ; Also in DELVAL2
                        MOVWF           DELVAL2                 ;
HERE            DECF            DELVAL2,F               ; Decrement contents of DELVAL2 by one (result in File Reg)
                BNZ             HERE                    ; If value not zero go back to HERE
                DECF            DELVAL1,F               ; Decrement DELVAL1 result in file reg.
                BNZ             REPEAT                  ; If result not zero go to REPEAT
                RETURN                                  ; Return for subroutine
                END                                     ; End directive specifies end of assembly code
```

Interrupt programming in C

When writing interrupt code in C the user needs to declare the function that handles the interrupt as an interrupt service routine (ISR). This is done by using the **pragma** directive.
As we have two levels of interrupt priority i.e. **High** and **Low** we have to make the appropriate declarations. These are as follows:

```
#pragma interruptlow  function_name          // low priority ISR
#pragma interrupt function_name              // high priority ISR.
```

The MCC18 compiler does not place the ISR at the interrupt vector location automatically. The normal way to do this is to place a **GOTO** instruction in assembly code at the appropriate interrupt vector location in order to transfer control to ISR.

The following template shows how an interrupt service routine could be written in MCC18 C.

```
    void low_ISR(void);
    void high_ISR(void);

  /* For the PIC18 the high interrupt vector is found at 0x008. The code that follows will send
  the micro to the high interrupt service routine function that handles the high vector interrupts
  */


    #pragma code high_vector = 0x08    // High-priority interrupt vector location set by the
                                       // pragma code directive to absolute location 0x08
    void high_interrupt (void)

    {
            _asm
                    GOTO high_ISR;
            _endasm
    }

    #pragma code                        // end of this interrupt code returns compiler
                                       // to default code section

    #pragma interrupt high_ISR
    Void high_ISR (void)
    /* *****      High interrupt service routine      ******

    {
            // your high ISR goes in here

    }

  /* For the PIC18 the low interrupt vector is found at 0x18. The code that follows will send the
  micro to the low interrupt service routine function that handles the Low vector interrupts */

    #pragma code low_vector = 0x18     // the pragma code directive sets to code to absolute
                                       // location hex 0x018
    void  low_interrupt (void)

    {
            _asm
                    GOTO low_ISR;
```

```
                _endasm
        }

        #pragma code                      // end of low priority code, returns compiler to
                                          // default code section

        #pragma interrupt low_ISR
        void  low_ISR (void)

        /* ******    Low interrupt service routine    *********

        {
                // your low ISR goes in here

        }
```

Interrupt Example 1 MCC18 Version

The following listing shows the MCC18 version of external INT0 interrupts.

```
/*          Sample source program external interrupts
/*          File name:  Interrupt PIC18 INT0
/*          Version:    1.0
/*          Author:
/*          Company:  Coventry University
/*          Date:
/*          Program function: The following program demonstrates the use of external
             interrupts on Port B. Port B0 is our INT0 interrupt source. This pin is connected
            to a push to make switch that gives a low going pulse.
            The main program routine configures Port D as output displaying incrementing values
            with a time delay. On INT0 interruption the interrupt routine clears Port D, delays
            for a while and returns to the main program.  */

/* ----     The following configure operational parameters of the PIC ----  */

#pragma   config OSC = HS                 // HS oscillator
#pragma   config WDT = OFF                // Watcdog timer off
#pragma   config MCLRE = ON               // Master clear reset On
#pragma   config DEBUG = OFF              // Debug off
#pragma   config LVP = OFF                // Low voltage programming off

/*   -------  Include the following header files  ------ */

#include<p18f4520.h>                      // header files of PIC18F4520
#include<delays.h>                        // include delays
#include <portb.h>                        // Header files required for port B interrupts

void test_isr (void);                     //
void INT0_ISR(void) ;                     //Prototype for the goto that follows
#pragma code my_interrupt = 0x08          //High Interrupt vector @ 0x08

void my_interrupt (void)                  // At loc 0x08 we place the assembler instruction GOTO

             {

            _asm                          // This is the assembly code at vector location
                        GOTO test_isr
            _endasm

             }
```

```
#pragma code                          // used to allow linker to locate remaining code

/*   -----  Bellow we test for the source of the interrupt  ----   */

#pragma interrupt test_isr
void test_isr (void)

            {

            if (INTCONbits.INT0IF == 1)       // Was interrupt caused by INT0 ?
            INT0_ISR ();                      // Yes , execute INT0 program

                }

void main (void)

            {

            ADCON1 =0x0F;                     // Set Ports as Digital I/O rather than analogue
            TRISD = 0x00;                     // Make Port D all output
            LATD = 0x00;                      // Clear Port D

/*  ------   One other way in setting Interrupts could be the following  ----------

            OpenRB0INT (PORTB_CHANGE_INT_ON & PORTB_PULLUPS_ON & FALLING_EDGE_INT);
            however from a detaild point view we will break this down to the individual
            register settings as shown bellow. See appendix for further details */

            INTCONbits.INT0IE = 1;            //Enable INT0 interrupts
            INTCONbits.INT0IF = 0;            // Clear INT0 Interrupt flag
            INTCONbits.GIE = 1;               // Enable global interrupts

            while (1)                         // Repeat for ever until interrupted

            {
                        LATD++;               // Increment PORT D
                        Delay10KTCYx(200);    // Call delay function

                }
            }

/* -----   Interrupt routine  INT0_ISR follows bellow: ----  */

            void INT0_ISR (void)              // This is the interrupt service routine

            {

            INTCONbits.INT0IE = 0;            // Disable any interrupt within interrupts
            LATD = 0X00;                      // Clear Port D
            Delay10KTCYx(400);                // Delay for a while so results can be seen
            INTCONbits.INT0IE = 1;            // Re-enable INT0 interrupts
            INTCONbits.INT0IF = 0;            // Clear INT0 flag before returning to main program
             }
```

Exercise 1

Modify the C version of example 1 so that the main routine remains as in the example.
Your interrupt routine on INT0 should display the values of hex data 55 and AA with an appropriate time delay between the two values displayed before returning to the main routine.

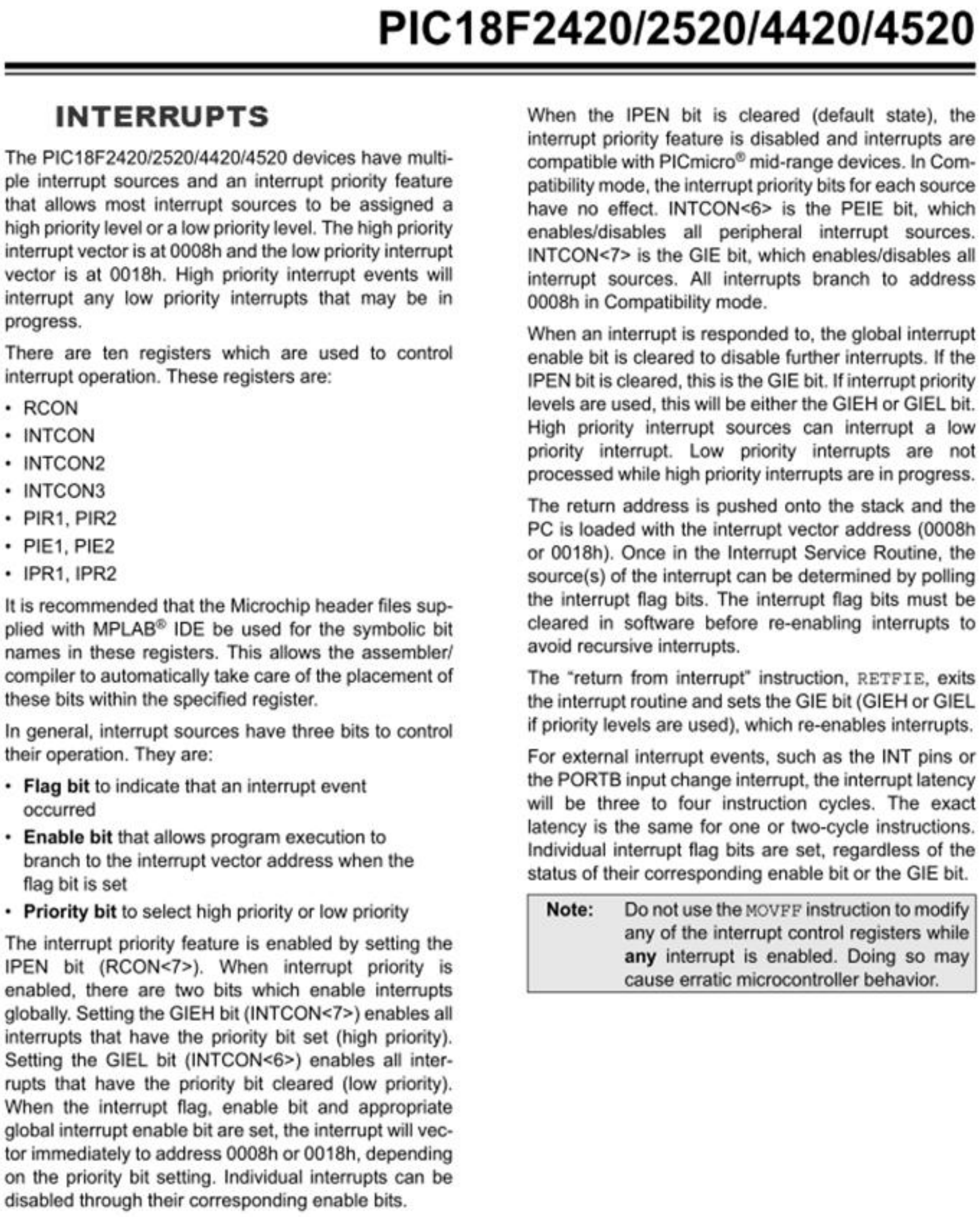
Exercise 2.

Modify MCC18 C version of Example 1 so that your main code shows a left shifted display on Port D (starting with D0). On an interrupt on RB0 your interrupt service routine should change the shifting to the right (starting with Port Bit D7) until all 8 bits have been fully shifted.

Exercise 3.

Exercise 3 will use two interrupting inputs INT0 and INT1. Note that INT0 is a high priority interrupt.
INT1 would be configured as a low priority interrupt.
The main routine should count on Port D as in our previous example.
Using two of the switches on the switch outboard write a program in MCC18 or assembly that will interrupt the main routine.
The ISR for INT0 should clear Port D with the appropriate time delay.
ISR for INT1 should flash all Port D LEDs twice with the appropriate time delay. On return from interrupts the system resumes the counting routine.
For further help see interrupt code template (pages 15-16).
Refer to the circuit diagram of Fig. 9 for system connections.

Appendix



PIC18F2420/2520/4420/4520

INTERRUPTS

The PIC18F2420/2520/4420/4520 devices have multiple interrupt sources and an interrupt priority feature that allows most interrupt sources to be assigned a high priority level or a low priority level. The high priority interrupt vector is at 0008h and the low priority interrupt vector is at 0018h. High priority interrupt events will interrupt any low priority interrupts that may be in progress.

There are ten registers which are used to control interrupt operation. These registers are:

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

It is recommended that the Microchip header files supplied with MPLAB® IDE be used for the symbolic bit names in these registers. This allows the assembler/compiler to automatically take care of the placement of these bits within the specified register.

In general, interrupt sources have three bits to control their operation. They are:

- **Flag bit** to indicate that an interrupt event occurred
- **Enable bit** that allows program execution to branch to the interrupt vector address when the flag bit is set
- **Priority bit** to select high priority or low priority

The interrupt priority feature is enabled by setting the IPEN bit (RCON<7>). When interrupt priority is enabled, there are two bits which enable interrupts globally. Setting the GIEH bit (INTCON<7>) enables all interrupts that have the priority bit set (high priority). Setting the GIEL bit (INTCON<6>) enables all interrupts that have the priority bit cleared (low priority). When the interrupt flag, enable bit and appropriate global interrupt enable bit are set, the interrupt will vector immediately to address 0008h or 0018h, depending on the priority bit setting. Individual interrupts can be disabled through their corresponding enable bits.

When the IPEN bit is cleared (default state), the interrupt priority feature is disabled and interrupts are compatible with PICmicro® mid-range devices. In Compatibility mode, the interrupt priority bits for each source have no effect. INTCON<6> is the PEIE bit, which enables/disables all peripheral interrupt sources. INTCON<7> is the GIE bit, which enables/disables all interrupt sources. All interrupts branch to address 0008h in Compatibility mode.

When an interrupt is responded to, the global interrupt enable bit is cleared to disable further interrupts. If the IPEN bit is cleared, this is the GIE bit. If interrupt priority levels are used, this will be either the GIEH or GIEL bit. High priority interrupt sources can interrupt a low priority interrupt. Low priority interrupts are not processed while high priority interrupts are in progress.

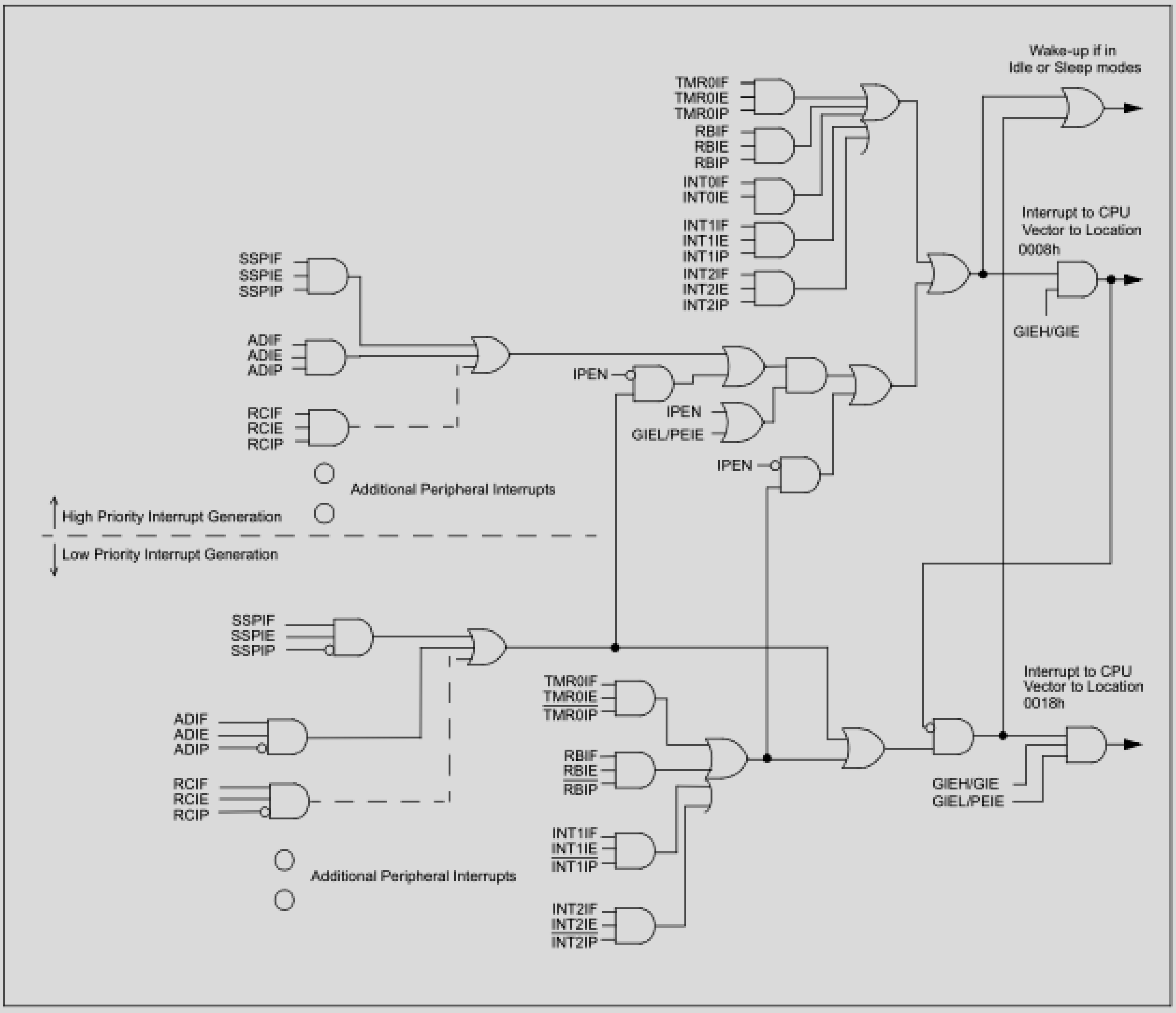
The return address is pushed onto the stack and the PC is loaded with the interrupt vector address (0008h or 0018h). Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bits must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

The "return from interrupt" instruction, `RETFIE`, exits the interrupt routine and sets the GIE bit (GIEH or GIEL if priority levels are used), which re-enables interrupts.

For external interrupt events, such as the INT pins or the PORTB input change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency is the same for one or two-cycle instructions. Individual interrupt flag bits are set, regardless of the status of their corresponding enable bit or the GIE bit.

Note:	Do not use the `MOVFF` instruction to modify any of the interrupt control registers while **any** interrupt is enabled. Doing so may cause erratic microcontroller behavior.

FIGURE 9-1: PIC18 INTERRUPT LOGIC



INTCON Registers

The INTCON registers are readable and writable registers, which contain various enable, priority and flag bits.

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

INTCON REGISTER

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
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit

1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit

1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit

1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

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

1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit

1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

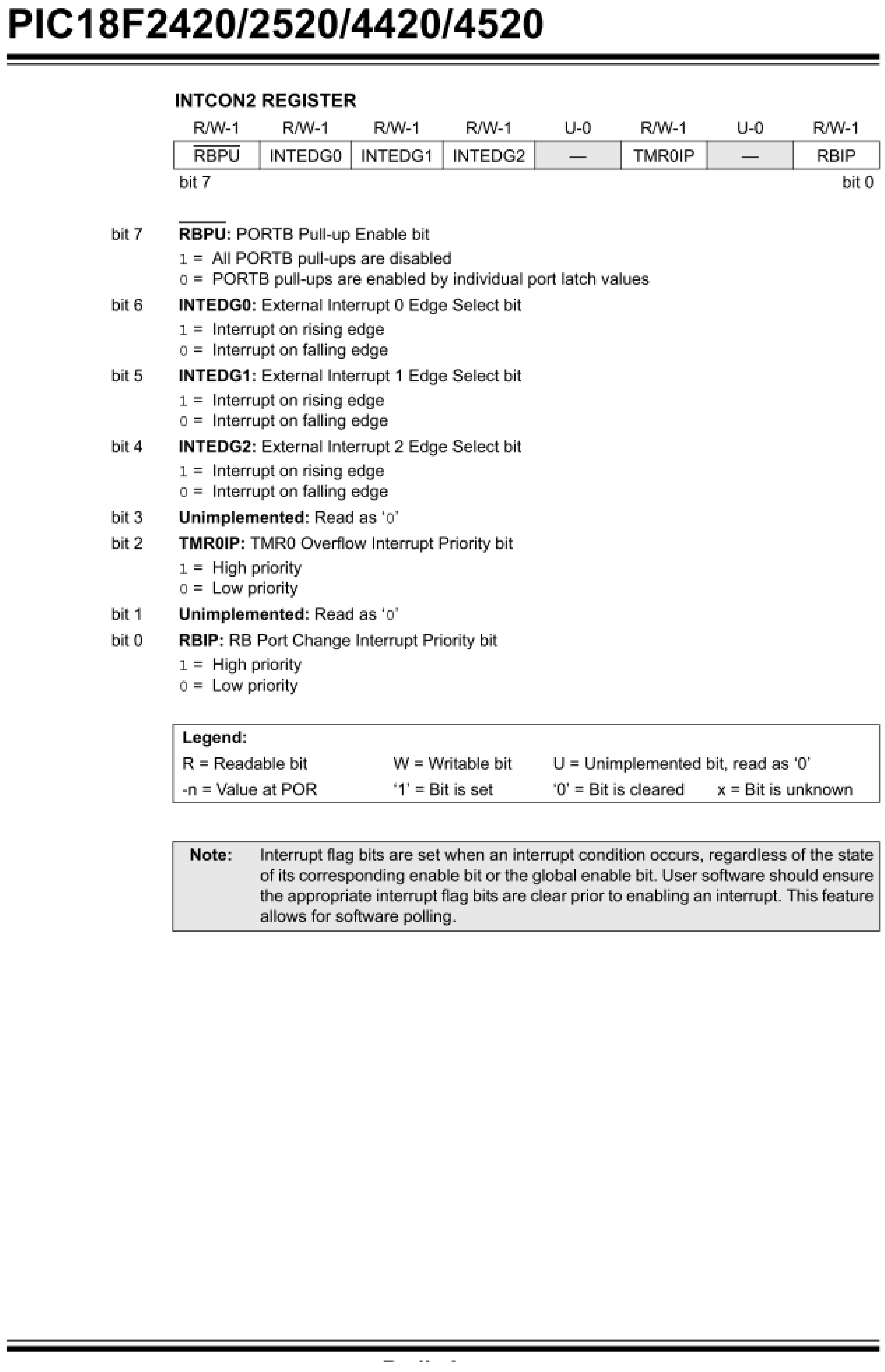
bit 0 **RBIF:** RB Port Change Interrupt Flag bit

1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown



INTCON2 REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

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

bit 6 **INTEDG0:** External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1:** External Interrupt 1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2:** External Interrupt 2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 3 **Unimplemented:** Read as '0'

bit 2 **TMR0IP:** TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **RBIP:** RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

INTCON3 REGISTER

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
`1` = High priority
`0` = Low priority

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
`1` = High priority
`0` = Low priority

bit 5 **Unimplemented:** Read as '`0`'

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
`1` = Enables the INT2 external interrupt
`0` = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
`1` = Enables the INT1 external interrupt
`0` = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '`0`'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
`1` = The INT2 external interrupt occurred (must be cleared in software)
`0` = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
`1` = The INT1 external interrupt occurred (must be cleared in software)
`0` = The INT1 external interrupt did not occur

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

PIR Registers

The PIR registers contain the individual flag bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Request Flag registers (PIR1 and PIR2).

Note 1: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the Global Interrupt Enable bit, GIE (INTCON<7>).

2: User software should ensure the appropriate interrupt flag bits are cleared prior to enabling an interrupt and after servicing that interrupt.

PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit(1)
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

Note 1: This bit is unimplemented on 28-pin devices and will read as '0'.

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared in software)
0 = The A/D conversion is not complete

bit 5 **RCIF:** EUSART Receive Interrupt Flag bit
1 = The EUSART receive buffer, RCREG, is full (cleared when RCREG is read)
0 = The EUSART receive buffer is empty

bit 4 **TXIF:** EUSART Transmit Interrupt Flag bit
1 = The EUSART transmit buffer, TXREG, is empty (cleared when TXREG is written)
0 = The EUSART transmit buffer is full

bit 3 **SSPIF:** Master Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete (must be cleared in software)
0 = Waiting to transmit/receive

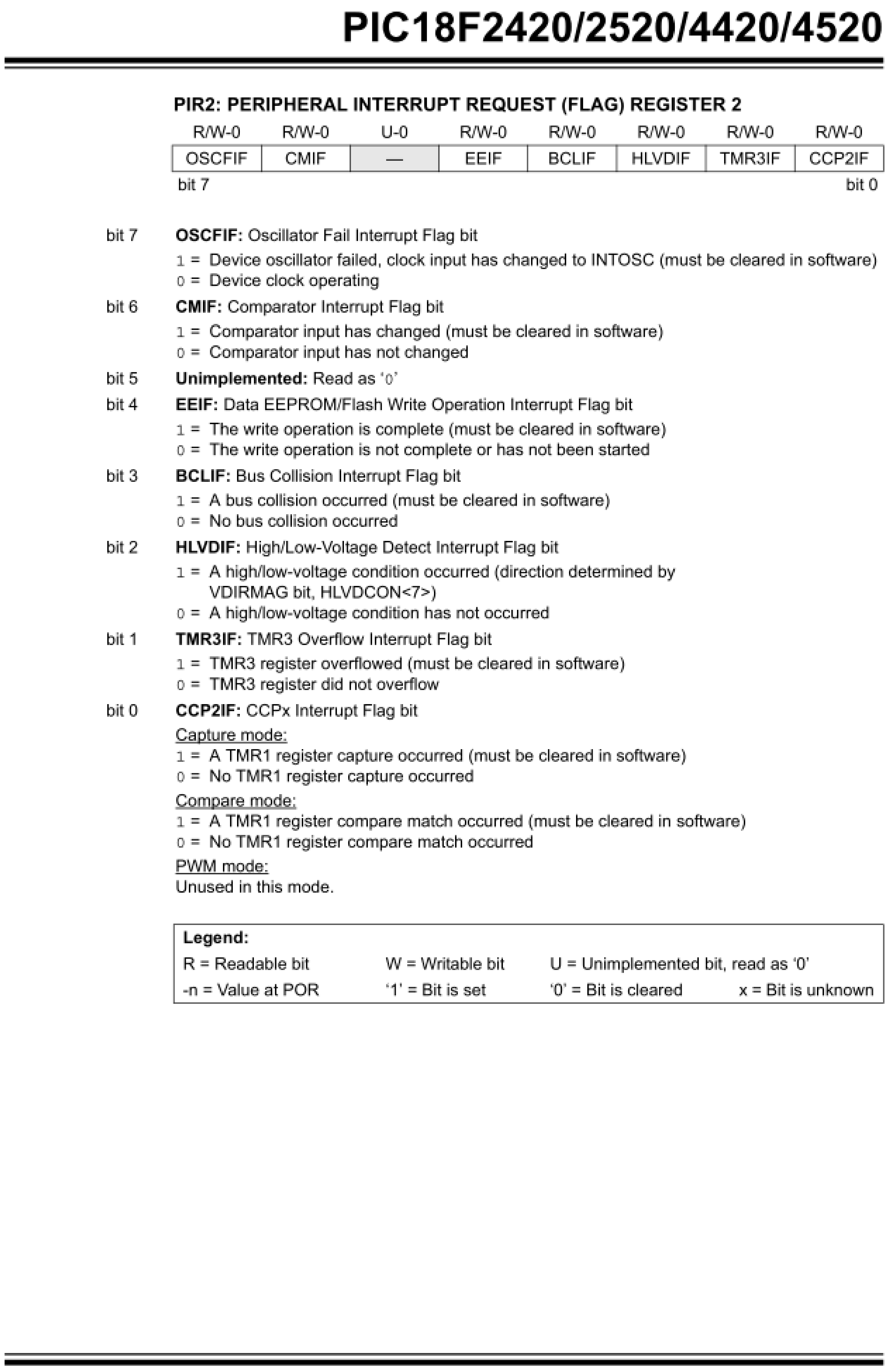
bit 2 **CCP1IF:** CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

bit 1 **TMR2IF:** TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF:** TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown



PIR2: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2

R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIF	CMIF	—	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF
bit 7							bit 0

bit 7 **OSCFIF:** Oscillator Fail Interrupt Flag bit
1 = Device oscillator failed, clock input has changed to INTOSC (must be cleared in software)
0 = Device clock operating

bit 6 **CMIF:** Comparator Interrupt Flag bit
1 = Comparator input has changed (must be cleared in software)
0 = Comparator input has not changed

bit 5 **Unimplemented:** Read as ‘0’

bit 4 **EEIF:** Data EEPROM/Flash Write Operation Interrupt Flag bit
1 = The write operation is complete (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **BCLIF:** Bus Collision Interrupt Flag bit
1 = A bus collision occurred (must be cleared in software)
0 = No bus collision occurred

bit 2 **HLVDIF:** High/Low-Voltage Detect Interrupt Flag bit
1 = A high/low-voltage condition occurred (direction determined by VDIRMAG bit, HLVDCON<7>)
0 = A high/low-voltage condition has not occurred

bit 1 **TMR3IF:** TMR3 Overflow Interrupt Flag bit
1 = TMR3 register overflowed (must be cleared in software)
0 = TMR3 register did not overflow

bit 0 **CCP2IF:** CCPx Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

PIE Registers

The PIE registers contain the individual enable bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Enable registers (PIE1 and PIE2). When IPEN = 0, the PEIE bit must be set to enable any of these peripheral interrupts.

PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE[(1)]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7	**PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit[(1)] 1 = Enables the PSP read/write interrupt 0 = Disables the PSP read/write interrupt **Note 1:** This bit is unimplemented on 28-pin devices and will read as '0'.
bit 6	**ADIE:** A/D Converter Interrupt Enable bit 1 = Enables the A/D interrupt 0 = Disables the A/D interrupt
bit 5	**RCIE:** EUSART Receive Interrupt Enable bit 1 = Enables the EUSART receive interrupt 0 = Disables the EUSART receive interrupt
bit 4	**TXIE:** EUSART Transmit Interrupt Enable bit 1 = Enables the EUSART transmit interrupt 0 = Disables the EUSART transmit interrupt
bit 3	**SSPIE:** Master Synchronous Serial Port Interrupt Enable bit 1 = Enables the MSSP interrupt 0 = Disables the MSSP interrupt
bit 2	**CCP1IE:** CCP1 Interrupt Enable bit 1 = Enables the CCP1 interrupt 0 = Disables the CCP1 interrupt
bit 1	**TMR2IE:** TMR2 to PR2 Match Interrupt Enable bit 1 = Enables the TMR2 to PR2 match interrupt 0 = Disables the TMR2 to PR2 match interrupt
bit 0	**TMR1IE:** TMR1 Overflow Interrupt Enable bit 1 = Enables the TMR1 overflow interrupt 0 = Disables the TMR1 overflow interrupt

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER 2

R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIE	CMIE	—	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE
bit 7							bit 0

bit 7	**OSCFIE:** Oscillator Fail Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 6	**CMIE:** Comparator Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 5	**Unimplemented:** Read as '0'
bit 4	**EEIE:** Data EEPROM/Flash Write Operation Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 3	**BCLIE:** Bus Collision Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 2	**HLVDIE:** High/Low-Voltage Detect Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 1	**TMR3IE:** TMR3 Overflow Interrupt Enable bit 1 = Enabled 0 = Disabled
bit 0	**CCP2IE:** CCP2 Interrupt Enable bit 1 = Enabled 0 = Disabled

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

IPR Registers

The IPR registers contain the individual priority bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Priority registers (IPR1 and IPR2). Using the priority bits requires that the Interrupt Priority Enable (IPEN) bit be set.

REGISTER 9-8: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PSPIP[(1)]	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
bit 7							bit 0

bit 7 **PSPIP:** Parallel Slave Port Read/Write Interrupt Priority bit[(1)]
1 = High priority
0 = Low priority

Note 1: This bit is unimplemented on 28-pin devices and will read as '0'.

bit 6 **ADIP:** A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RCIP:** EUSART Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXIP:** EUSART Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSPIP:** Master Synchronous Serial Port Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **CCP1IP:** CCP1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR2IP:** TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP:** TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

IPR2: PERIPHERAL INTERRUPT PRIORITY REGISTER 2

R/W-1	R/W-1	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
OSCFIP	CMIP	—	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP
bit 7							bit 0

bit 7 **OSCFIP:** Oscillator Fail Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **CMIP:** Comparator Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **Unimplemented:** Read as ‘0’

bit 4 **EEIP:** Data EEPROM/Flash Write Operation Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **BCLIP:** Bus Collision Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **HLVDIP:** High/Low-Voltage Detect Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR3IP:** TMR3 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **CCP2IP:** CCP2 Interrupt Priority bit
1 = High priority
0 = Low priority

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

RCON Register

The RCON register contains flag bits which are used to determine the cause of the last Reset or wake-up from Idle or Sleep modes. RCON also contains the IPEN bit which enables interrupt priorities.

The operation of the SBOREN bit and the Reset flag bits is discussed in more detail in **Section 4.1 "RCON Register"**.

REGISTER 9-10: RCON REGISTER

R/W-0	R/W-1[(1)]	U-0	R/W-1	R-1	R-1	R/W-0[(1)]	R/W-0
IPEN	SBOREN	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16XXX Compatibility mode)

bit 6 **SBOREN:** Software BOR Enable bit[(1)]
For details of bit operation, see Register 4-1.

Note 1: Actual Reset values are determined by device configuration and the nature of the device Reset. See Register 4-1 for additional information.

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{\text{RI}}$:** RESET Instruction Flag bit
For details of bit operation, see Register 4-1.

bit 3 **$\overline{\text{TO}}$:** Watchdog Time-out Flag bit
For details of bit operation, see Register 4-1.

bit 2 **$\overline{\text{PD}}$:** Power-down Detection Flag bit
For details of bit operation, see Register 4-1.

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status bit
For details of bit operation, see Register 4-1.

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit
For details of bit operation, see Register 4-1.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

INTn Pin Interrupts

External interrupts on the RB0/INT0, RB1/INT1 and RB2/INT2 pins are edge-triggered. If the corresponding INTEDGx bit in the INTCON2 register is set (= `1`), the interrupt is triggered by a rising edge; if the bit is clear, the trigger is on the falling edge. When a valid edge appears on the RBx/INTx pin, the corresponding flag bit, INTxF, is set. This interrupt can be disabled by clearing the corresponding enable bit, INTxE. Flag bit, INTxF, must be cleared in software in the Interrupt Service Routine before re-enabling the interrupt.

All external interrupts (INT0, INT1 and INT2) can wake-up the processor from Idle or Sleep modes if bit INTxE was set prior to going into those modes. If the Global Interrupt Enable bit, GIE, is set, the processor will branch to the interrupt vector following wake-up.

Interrupt priority for INT1 and INT2 is determined by the value contained in the interrupt priority bits, INT1IP (INTCON3<6>) and INT2IP (INTCON3<7>). There is no priority bit associated with INT0. It is always a high priority interrupt source.

9.7 TMR0 Interrupt

In 8-bit mode (which is the default), an overflow in the TMR0 register (FFh → 00h) will set flag bit, TMR0IF. In 16-bit mode, an overflow in the TMR0H:TMR0L register pair (FFFFh → 0000h) will set TMR0IF. The interrupt can be enabled/disabled by setting/clearing enable bit, TMR0IE (INTCON<5>). Interrupt priority for Timer0 is determined by the value contained in the interrupt priority bit, TMR0IP (INTCON2<2>). See **Section 11.0 "Timer0 Module"** for further details on the Timer0 module.

9.8 PORTB Interrupt-on-Change

An input change on PORTB<7:4> sets flag bit, RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit, RBIE (INTCON<3>). Interrupt priority for PORTB interrupt-on-change is determined by the value contained in the interrupt priority bit, RBIP (INTCON2<0>).

9.9 Context Saving During Interrupts

During interrupts, the return PC address is saved on the stack. Additionally, the WREG, Status and BSR registers are saved on the fast return stack. If a fast return from interrupt is not used (see **Section 5.3 "Data Memory Organization"**), the user may need to save the WREG, Status and BSR registers on entry to the Interrupt Service Routine. Depending on the user's application, other registers may also need to be saved. Example 9-1 saves and restores the WREG, Status and BSR registers during an Interrupt Service Routine.

EXAMPLE SAVING STATUS, WREG AND BSR REGISTERS IN RAM

```
MOVWF     W_TEMP                        ; W_TEMP is in virtual bank
MOVFF     STATUS, STATUS_TEMP           ; STATUS_TEMP located anywhere
MOVFF     BSR, BSR_TEMP                 ; BSR_TMEP located anywhere
;
; USER ISR CODE
;
MOVFF     BSR_TEMP, BSR                 ; Restore BSR
MOVF      W_TEMP, W                     ; Restore WREG
MOVFF     STATUS_TEMP, STATUS           ; Restore STATUS
```

MCC18 PORT B Functions

PORTB is supported with the following functions:

Function	Description
`ClosePORTB`	Disable the interrupts and internal pull-up resistors for PORTB.
`CloseRB`***`x`***`INT`	Disable interrupts for PORTB pin ***x*** .
`DisablePullups`	Disable the internal pull-up resistors on PORTB.
`EnablePullups`	Enable the internal pull-up resistors on PORTB.
`OpenPORTB`	Configure the interrupts and internal pull-up resistors on PORTB.
`OpenRB`***`x`***`INT`	Enable interrupts for PORTB pin ***x*** .

Function Descriptions

ClosePORTB

Function:	Disable the interrupts and internal pull-up resistors for `PORTB`.
Include:	`portb.h`
Prototype:	`void ClosePORTB( void );`
Remarks:	This function disables the `PORTB` interrupt-on-change and the internal pull-up resistors.
File Name:	`pbclose.c`

CloseRB0INT
CloseRB1INT
CloseRB2INT

Function:	Disable the interrupts for the specified `PORTB` pin.
Include:	`portb.h`

Prototype:	`void CloseRB0INT( void );`
	`void CloseRB1INT( void );`
	`void CloseRB2INT( void );`

Remarks:	This function disables the `PORTB` interrupt-on-change.

File Name:	`rb0close.c`
	`rb1close.c`
	`rb2close.c`

DisablePullups

Function:	Disable the internal pull-up resistors on `PORTB`.
Include:	`portb.h`
Prototype:	`void DisablePullups( void );`
Remarks:	This function disables the internal pull-up resistors on `PORTB`.
File Name:	`pulldis.c`

EnablePullups

Function:	Enable the internal pull-up resistors on `PORTB`.
Include:	`portb.h`
Prototype:	`void EnablePullups( void );`
Remarks:	This function enables the internal pull-up resistors on `PORTB`.
File Name:	`pullen.c`

OpenPORTB

Function: Configure the interrupts and internal pull-up resistors on `PORTB`.

Include: `portb.h`

Prototype: `void OpenPORTB( unsigned char config);`

Arguments: ***config***

A bitmask that is created by performing a bitwise AND operation ('`&`') with a value from each of the categories listed below. These values are defined in the file `portb.h`.

Interrupt-on-change:

`PORTB_CHANGE_INT_ON`	Interrupt enabled
`PORTB_CHANGE_INT_OFF`	Interrupt disabled

Enable Pullups:

`PORTB_PULLUPS_ON`	pull-up resistors enabled
`PORTB_PULLUPS_OFF`	pull-up resistors disabled

Remarks: This function configures the interrupts and internal pull-up resistors on `PORTB`.

File Name: `pbopen.c`

Code Example:

```
OpenPORTB( PORTB_CHANGE_INT_ON & PORTB_PULLUPS_ON);
```

OpenRB0INT
OpenRB1INT
OpenRB2INT

Function: Enable interrupts for the specified `PORTB` pin.

Include: `portb.h`

Prototype:

```
void OpenRB0INT( unsigned char config );
void OpenRB1INT( unsigned char config );
void OpenRB2INT( unsigned char config );
```

Arguments: ***config***

A bitmask that is created by performing a bitwise AND operation ('`&`') with a value from each of the categories listed below. These values are defined in the file `portb.h`.

Interrupt-on-change:

`PORTB_CHANGE_INT_ON`	Interrupt enabled
`PORTB_CHANGE_INT_OFF`	Interrupt disabled

Interrupt-on-edge:

`RISING_EDGE_INT`	Interrupt on rising edge
`FALLING_EDGE_INT`	Interrupt on falling edge

Enable Pullups:

`PORTB_PULLUPS_ON`	pull-up resistors enabled
`PORTB_PULLUPS_OFF`	pull-up resistors disabled

Remarks: This function configures the interrupts and internal pull-up resistors on `PORTB`.

File Name:
`rb0open.c`
`rb1open.c`
`rb2open.c`

Code Example:

```
OpenRB0INT( PORTB_CHANGE_INT_ON & RISING_EDGE_INT &
PORTB_PULLUPS_ON);
```

Map of Interrupt Sources, Flags, Enabling and Priorities

Interrrupt	IP	IE	IF	EDGE
INT0	<high>	INTCON.INT0IE	INTCON.INT0IF	INTCON2.INTEDG0
INT1	INTCON3.INT1IP	INTCON3.INT1IE	INTCON3.INT1IF	INTCON2.INTEDG1
INT2	INTCON3.INT2IP	INTCON3.INT2IE	INTCON3.INT2IF	INTCON2.INTEDG2
INT3	INTCON2.INT3IP	INTCON3.INT3IE	INTCON3.INT3IF	INTCON2.INTEDG3
INTB	INTCON2.RBIP	INTCON.RBIE	INTCON.RBIF	-
TMR0	INTCON2.TMR0IP	INTCON.TMR0IE	INTCON.TMR0IF	
TMR1	IPR1.TMR1IP	PIE1.TMR1IE	PIR1.TMR1IF	
TMR2	IPR1.TMR2IP	PIE1.TMR2IE	PIR1.TMR2IF	
TMR3	IPR2.TMR3IP	PIE2.TMR3IE	PIR2.TMR3IF	
TMR4	IPR3.TMR4IP	PIE3.TMR4IE	PIR3.TMR4IF	
ADC	IPR1.ADIP	PIE1.PSPIE	PIR1.ADIF	
RC1	IPR1.RC1IP	PIE1.RC1IE	PIR1.RC1IF	
TX1	IPR1.TX1IP	PIE1.TX1IE	PIR1.TX1IF	
RC2	IPR3.RC2IP	PIE3.RC2IE	PIR3.RC2IF	
TX2	IPR3.TX2IP	PIE3.TX2IE	PIR3.TX2IF	
EEPROM	IPR2.EEIP	PIE2.EEIE	PIR2.EEIF	
PSP	IPR1.PSPIP	PIE1.PSPIE	PIR1.PSPIF	
SSP1	IPR1.SSP1IP	PIE1.SSP1IE	PIR1.SSP1IF	
BCL1	IPR2.BCL1IP	PIE2.BCL1IE	PIR1.BCL1IF	
SSP2	IPR3.SSP2IP	PIE3.SSP2IE	PIR3.SSP2IF	
BCL2	IPR3.BCL2IP	PIE3.BCL2IE	PIR2.BCL2IF	
CCP1	IPR1.CCP1IP	PIE1.CCP1IE	PIR1.CCP1IF	
CCP2	IPR2.CCP2IP	PIE2.CCP2IE	PIR2.CCP2IF	
CCP3	IPR3.CCP3IP	PIE3.CCP3IE	PIR3.CCP3IF	
CCP4	IPR3.CCP4IP	PIE3.CCP4IE	PIR3.CCP4IF	
CCP5	IPR3.CCP5IP	PIE3.CCP5IE	PIR3.CCP5IF	
OSC	IPR2.OSCFIP	PIE2.OSCFIE	PIR2.OSCFIF	
CMP	IPR2.CMIP	PIE2.CMIE	PIR2.CMIF	
HLVD	IPR2.HLVDIP	PIE2.HLVDIE	PIR2.HLVDIF	

LATENCY

The time between when an interrupt occurs and when the first ISR instruction is executed is the latency of the interrupt. The three elements that affect latency are:

1. **Processor servicing of interrupt**: The amount of time it takes the processor to recognize the interrupt and branch to the first address of the interrupt vector. To determine this value refer to the processor data sheet for the specific processor and interrupt source being used.
2. **Interrupt vector execution**: The amount of time it takes to execute the code at the interrupt vector that branches to the ISR.
3. **ISR prologue code**: The amount of time it takes MPLAB C18 to save the compiler managed resources and the data in the `save=` list.

NESTING INTERRUPTS

Low-priority interrupts may be nested since active registers are saved onto the software stack. Only a single instance of a high-priority interrupt service routine may be active at a time since these ISR's use the single-level hardware shadow registers.

If nesting of low-priority interrupts is desired, a statement to set the `GIEL` bit can be added near the beginning of the ISR. See the processor data sheet for details.

Ref http://www.microchip.com/