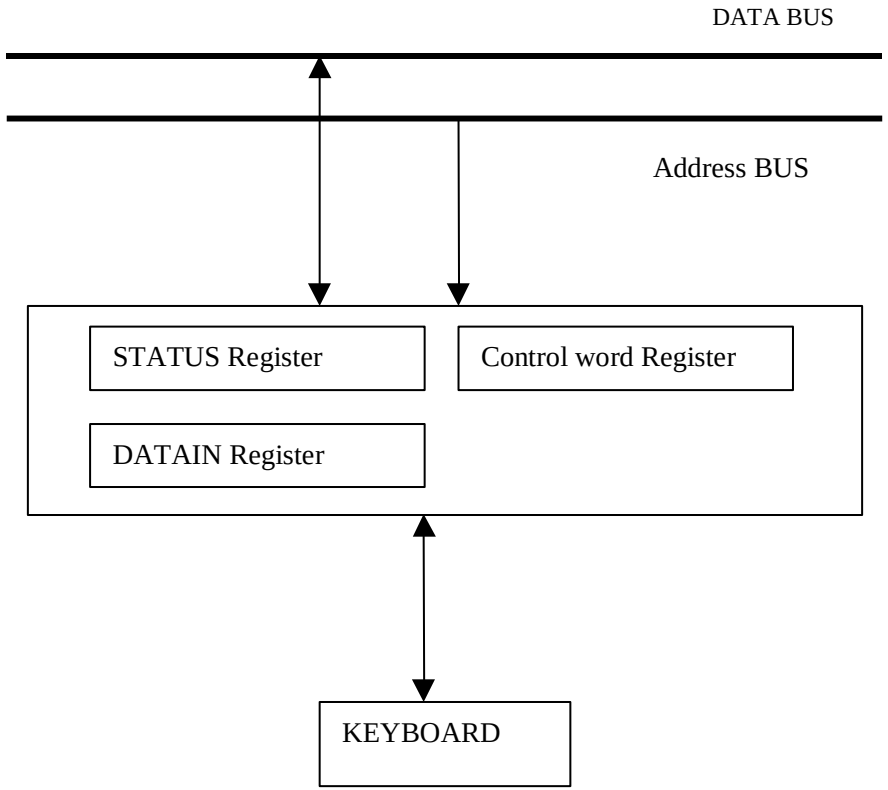
4.2 programmed I/O

- a single microcontroller can serve several devices
- 2 methods by which devices receive service from microcontroller
  programmed I/O (*polling*)
  interrupt driven I/O
- in programmed I/O, microcontroller continuously monitors the status of the devices
- when the status condition of a device is met, it performs the service
- not efficient use of microcontroller

Consider a microcontroller system

If a key is pressed
- DATAIN Register contains the new key
- STATUS Register is set



The speed of the microcontroller and the speed of the human operator are different.

We must ensure that an instruction to read character from the keyboard is executed only when a character is available in the input buffer (*DATAIN Register*) of the keyboard interface.

We must also ensure that an input character is read only once.

Consider the following pseudocode:

```
WAIT_key    read STATUS Register
            test STATUS Register
            if no new character, jump WAIT_key
            read DATAIN Register
```

The microcontroller repeatedly checks a status flag to achieve the required synchronization between the microcontroller and the keyboard.

The program enters a wait loop in which it repeatedly tests the STATUS Register.

During this period, the microcontroller is not performing any useful tasks!

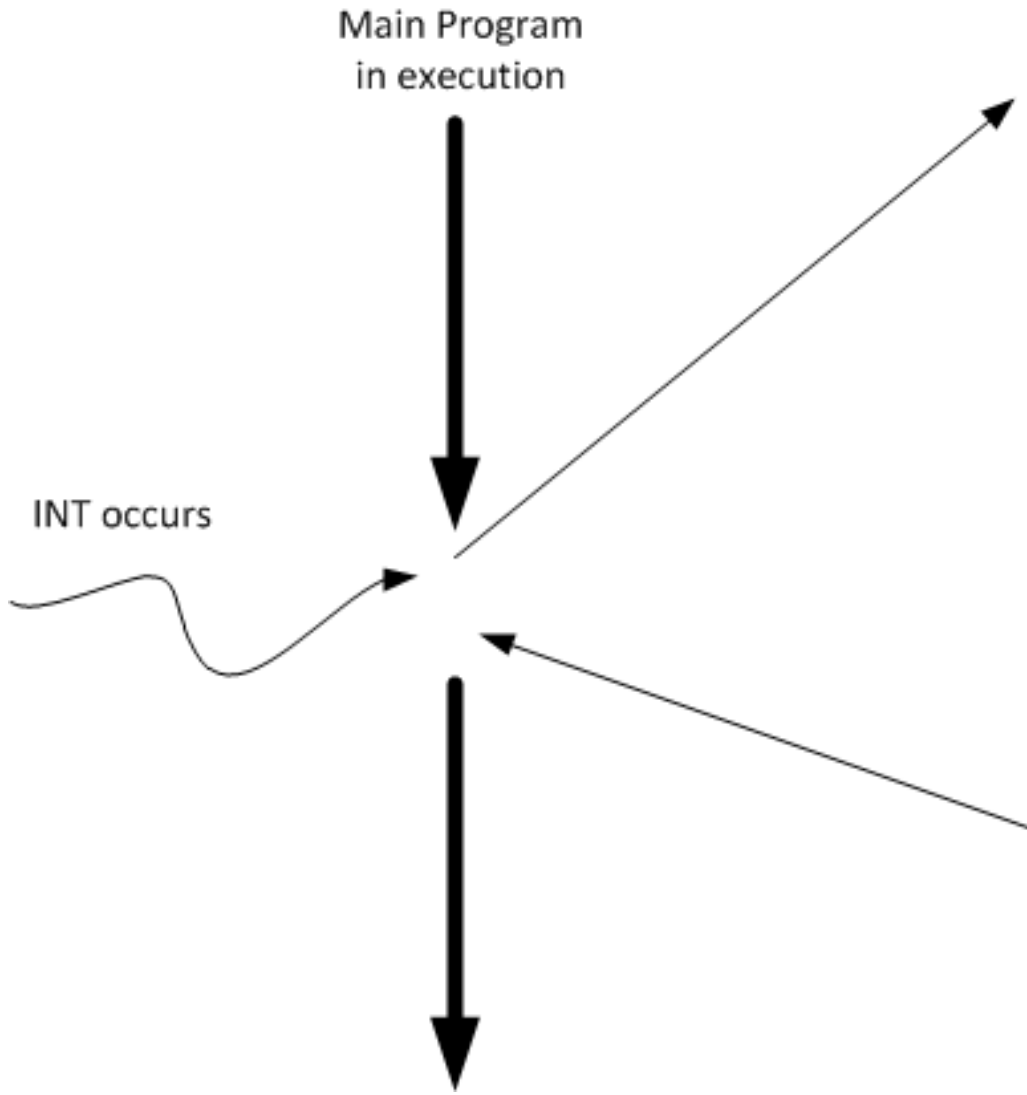
4.3 interrupt

- a mechanism to synchronize I/O operations, handle errors and emergency events, coordinate the use of shared resources, etc.
- interrupt can be generated internally by circuits inside the microcontroller (e.g. timer), or software errors (e.g. divided-by-zero)
- interrupt can be generated by external devices, e.g. keyboard

- whenever any device needs microcontroller’s service, sends an interrupt signal
- microcontroller stops whatever it is doing, serves the device
- each device can get the attention of microcontroller based on its priority
- efficient use of microcontroller
- mechanism involves some overhead, e.g. saving and restoring Program Counter

4.3.1 interrupt driven I/O

- in many situations, other tasks can be performed while the microcontroller waits for an I/O device to become ready
- the I/O device alerts the microcontroller when it is ready by sending a hardware signal called interrupt
- microcontroller allows normal program execution to be interrupted by some external signals from I/O devices
- when interrupted, it stops executing its current program and enters an *interrupt sequence*
- status of the current program is saved before entering the **Interrupt Service Routine** (ISR) that services the interrupt
- after servicing the interrupt, the status before the interrupt is restored, execution is then returned to the interrupted program



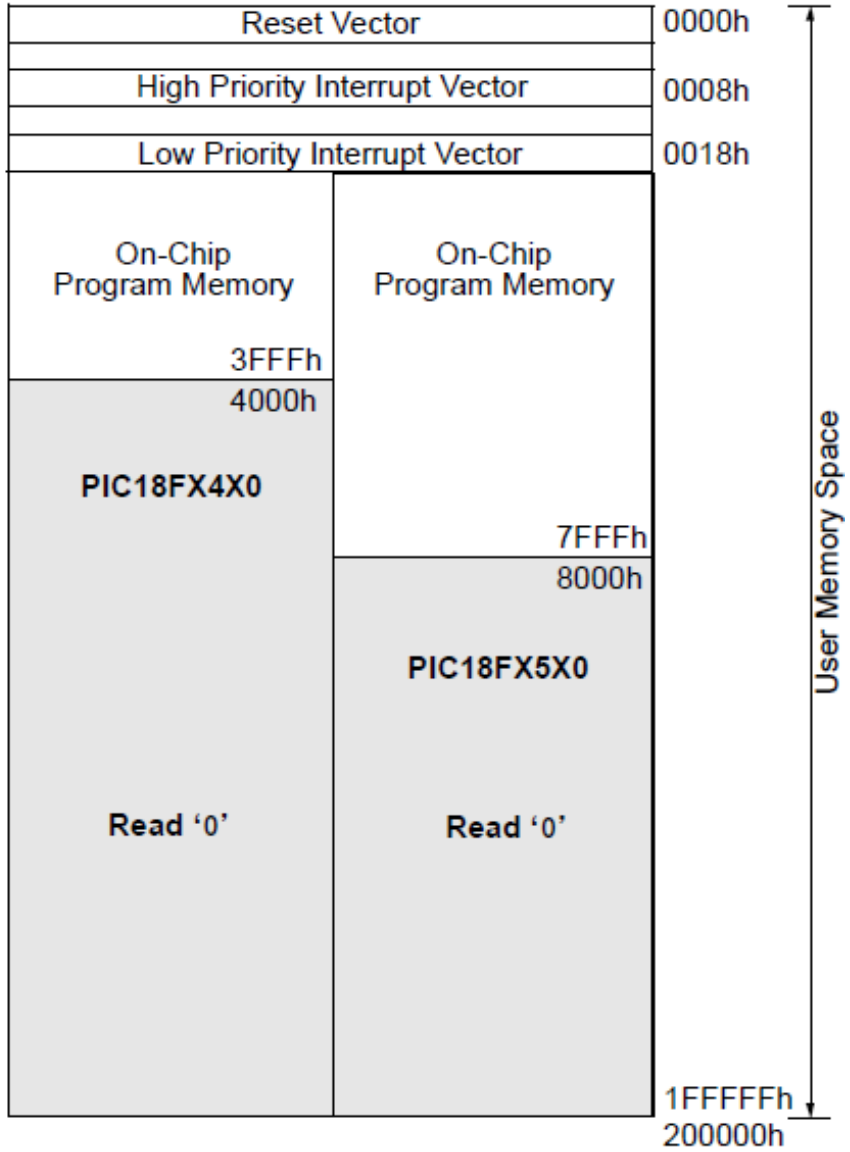
CPU completes the current instruction and saves the address of the next instruction (PC) on the stack

CPU jumps to a fixed location in program memory called the interrupt vector table.
The interrupt vector table directs the CPU to the address of the interrupt service routine (ISR).

CPU gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the ISR until it reaches the RETFIE

Upon the executing the RETFIE instruction, the CPU returns to the place where it was interrupt (it gets the PC address from the stack.)

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



Steps in executing an interrupt

Upon activation of interrupt, the microcontroller

- finishes executing the current instruction
- pushes the PC of next instruction into the stack
- jumps to the interrupt vector table to get the address of ISR and jumps to it
- begins executing the ISR instructions until it reaches the last instruction of ISR (RETFIE – RETurn From Interrupt Exit)
- executes RETFIE
  - pops the PC from the stack
  - starts to execute from that address

Program organization in MPLAB

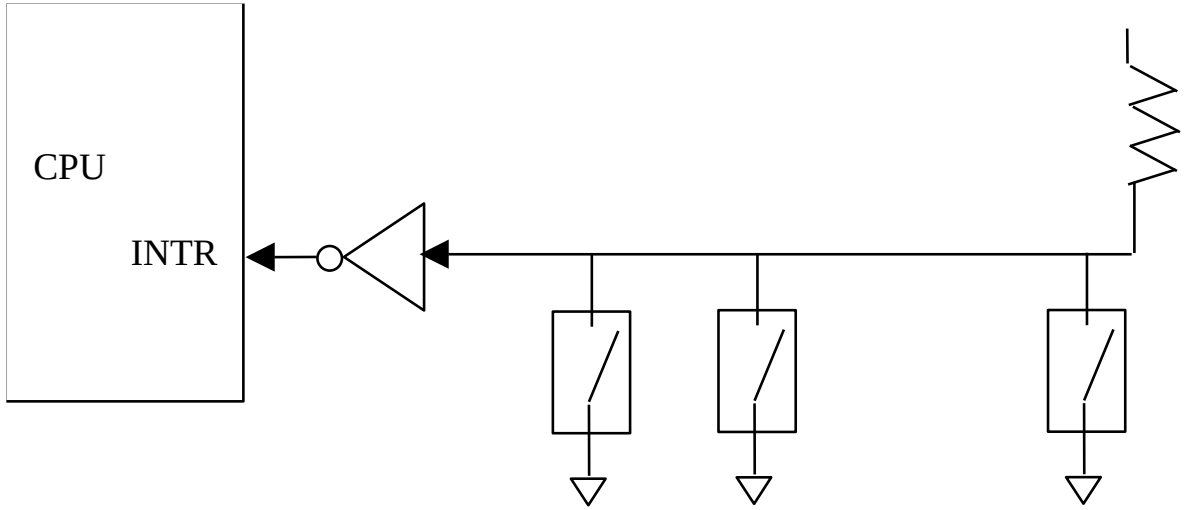
```
        ORG 0x0000
        goto    Main          ;go to start of main code

;*****************************************************************
;High priority interrupt vector
        ORG 0x0008
        bra HighInt       ;go to high priority interrupt routine

;*****************************************************************
;Low priority interrupt vector and routine

        ORG 0x0018
;   *** low priority interrupt code goes here ***
        retfie

;*****************************************************************
;High priority interrupt routine
HighInt:
;   *** high priority interrupt code goes here ***
        retfie  FAST
;*****************************************************************
;Start of main program
; The main program code is placed here.
Main:
;   *** main code goes here ***

        END
```

Sources of interrupts in PIC18

- external hardware interrupts
  - pins PORTB.0 (INT0), PORTB.1 (INT1), PORTB.2 (INT2)
- PORTB-Change interrupt
- timers
  - Timer0, Timer1, Timer2
- ADC
- serial communication
- CCP (compare capture pulse-width-modulation)
- ... etc.

4.3.2 handling multiple devices

- if there are multiple interrupt lines, each line is corresponding to an interrupt routine
- e.g., if INTR i is activated, the CPU will jump to the i-INTR routine
- this scheme is called multiple interrupt line
- in many cases, several devices capable of initiating interrupts are connected to the CPU, or several interrupts use the same interrupt service routine



Questions:
How can the CPU recognize the device requesting an interrupt?

How can the processor obtain the starting address of the appropriate routine?

Device identification by program polling

When a request is received over a common INTR line, additional information is needed to identify the particular device that activated the line. The information is provided in the status register of the device.

The interrupt service routine begins by polling the devices (check interrupt request bit of the status registers of the devices) in some order. The first device encountered with its IRQ bit set is the device that is serviced, and an appropriate subroutine is called to provide the requested service.

Polling interrupt service routine

if IRQ of device A set, jump to serve device A routine
if IRQ of device B set, jump to serve device B routine
::::::::::::::::::::
::::::::::::::::::::

Advantage : simple and easy to implement
Disadvantage: time spent interrogating the IRQ bits of all the devices

Vectored Interrupts

A device requesting an interrupt can identify itself by sending a special code to the CPU over the data bus.

The CPU gets the address of the corresponding interrupt service routine from a vector table based on the code.

The CPU will jump to the corresponding interrupt service routine.

NOT all CPUs have this mechanism.

8051 does not have
PIC18 does not have
68000 family do have

Questions:
Should a device be allowed to interrupt the CPU while another interrupt is being serviced?

How should two or more simultaneous interrupt requests be handled?

Priority of Interrupts

I/O devices, or interrupts are organized in a priority structure.

An interrupt request with a high priority should be accepted while the CPU is servicing another request from a low-priority device.

A real time clock interrupt should be of higher priority than a read key interrupt.

Priority of PIC18 interrupts

INTERRUPT	PRIORITY
High Priority Interrupt	High
Low Priority Interrupt	Low

In PIC18, we can configure some bits in control word registers to set the priority of an interrupt.

Each INTR line is assigned a different priority level.

Interrupt requests received over these lines are sent to a priority arbitration circuit in the CPU (or an external circuit).

A request is accepted only if

1. It has a higher priority than other interrupts that are being served or there is no interrupt being served.
2. The corresponding interrupt enable pin is enabled.

4.3.3 interrupt in PIC18

When interrupt occurs (*i.e. it is enabled*):

- Finishes executing the current instruction
- Pushes the PC of next instruction into the stack
- Jumps to the interrupt vector table to get the address of ISR
- Jumps to ISR
- Disables Global Interrupt Enable (GIE)
- Begins executing the ISR instructions until RETFIE
- Executes RETFIE
- Pops the PC from the stack
- Sets GIE
- Starts to execute from the address of that PC

Enabling and disabling an interrupt

- when PIC18 is powered on (or resets)
  - all interrupts are masked (disabled)
  - the default ISR address is 0008h

    no priorities for interrupts
- interrupts must be enabled by software in order for PIC18 to respond to them
- GIE bit is responsible for enabling and disabling the interrupts globally

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

Flag bit

- to indicate that an interrupt event occurred

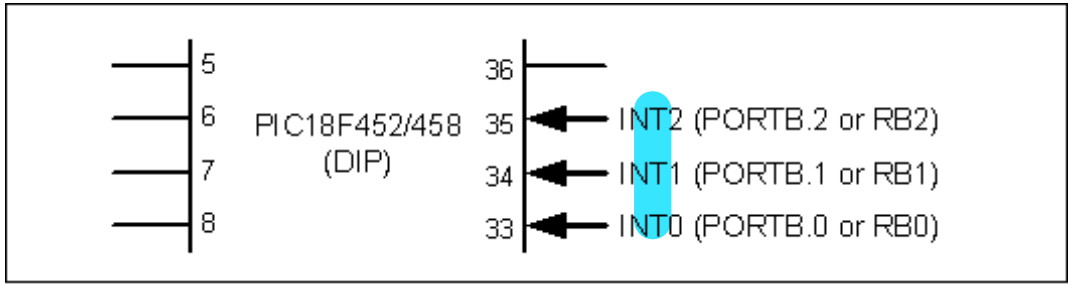
Enable bit

- 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

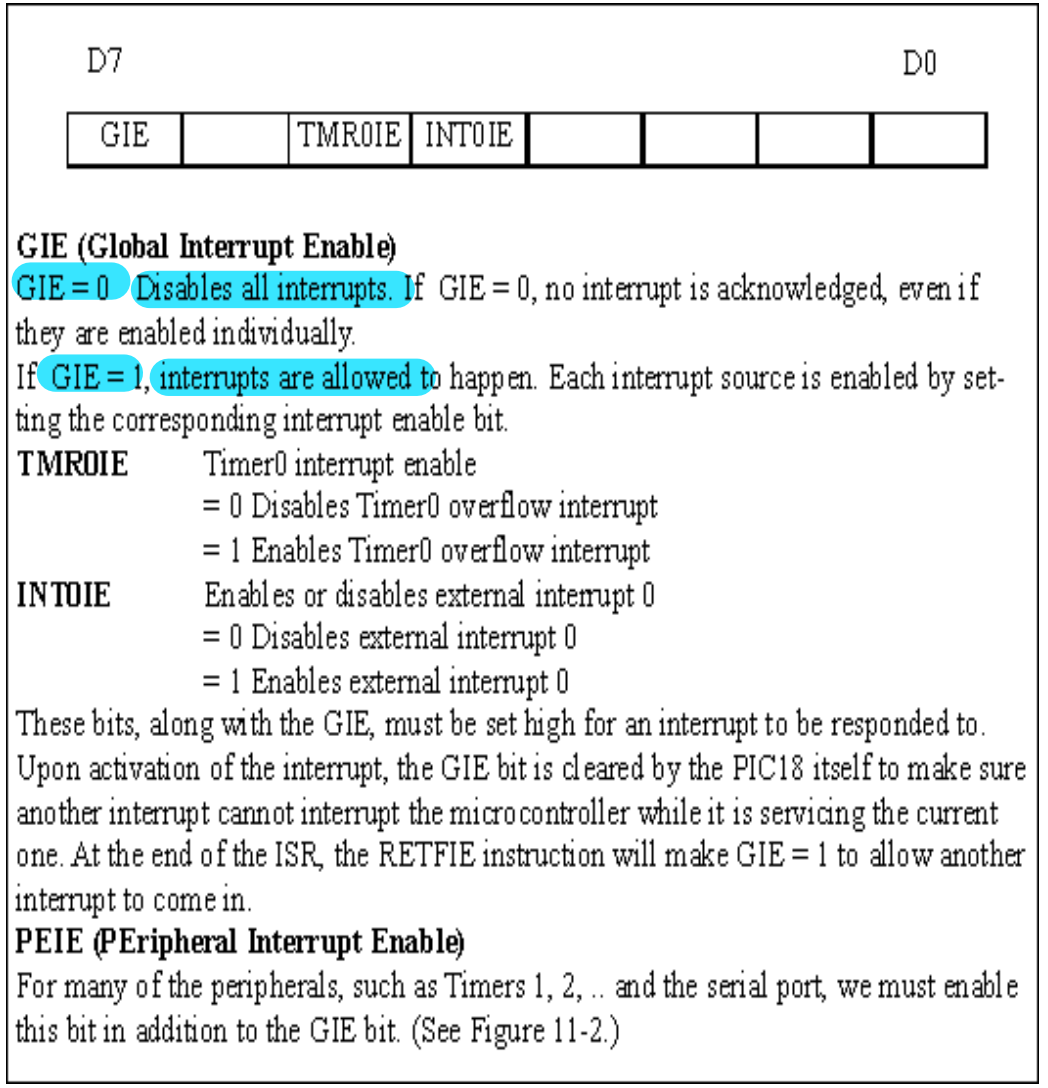
External interrupts INT0, INT1, INT2



INT (pin)	**Flag bit** **Register**	**Enable bit** **Register**	**Pos or Neg edge** **Register**
INT0 (RB0)	INT0IF INTCON	INT0IE INTCON	INTEDG0 INTCON2
INT1 (RB1)	INT1IF INTCON3	INT1IE INTCON3	INTEDG1 INTCON2
INT2 (RB2)	INT2IF INTCON3	INT2IE INTCON3	INTEDG2 INTCON2
	Set to 1 by the interrupt event	0 disable 1 enable	0 falling edge high to low 1 rising edge (default power-on)

Steps in enabling an interrupt

- Set the GIE bit from INTCON register
- Set the IE bit for that interrupt
- If the interrupt is one of the peripheral (timers 1, 2, serial, etc.) set PEIE bit from INTCON register (D6)



D7 D0

GIE		TMR0IE	INT0IE				

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.)

Example

a) enable

BSF INTCON, TMR0IE

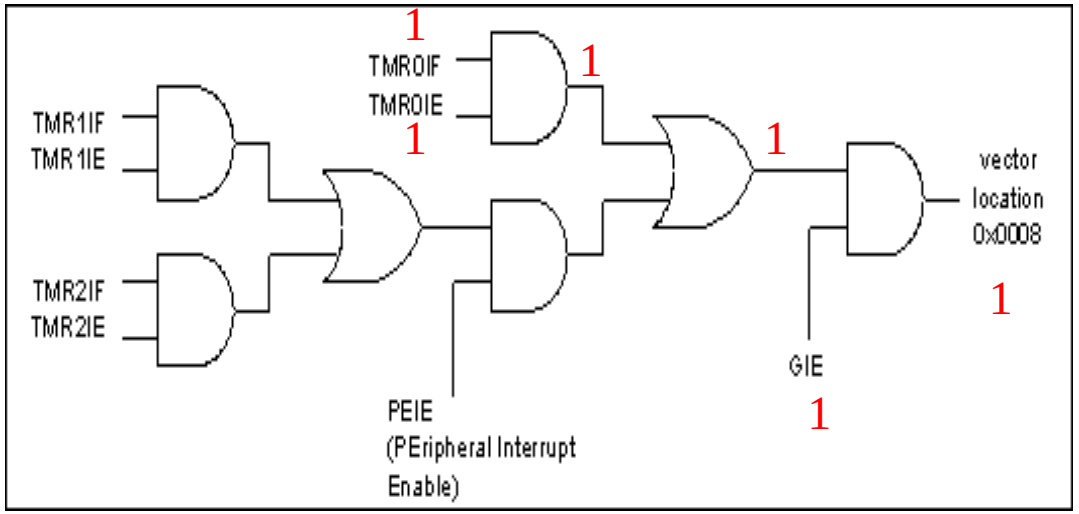
BSF INTCON, INT0IE

BSF INTCON, GIE

or

MOVLW B’10110000’

MOVWF INTCON



b) disable Timer0 interrupt

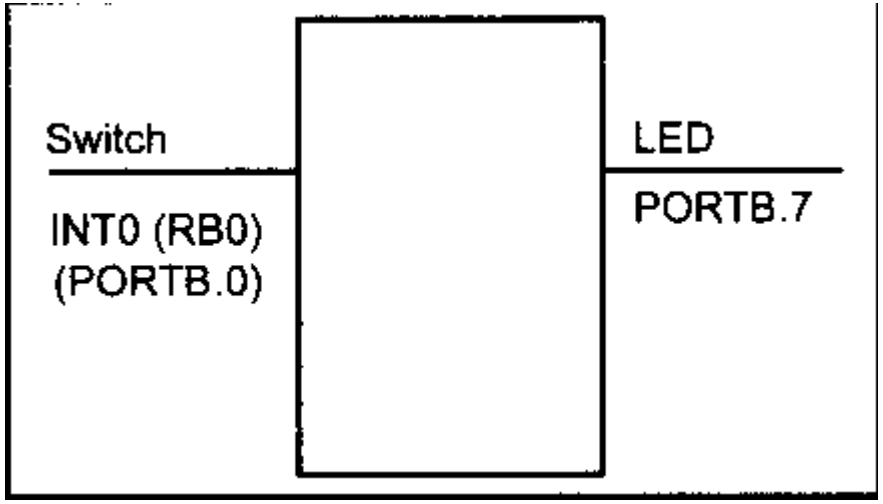
BCF INTCON, TMR0IE

c) disable all interrupts

BCF INTCON, GIE

Example

- connect a switch to INT0 and an LED to pin RB7
- every time INT0 is activated, toggle the LED
- at the same time, data is being transferred from PORTC to PORTD

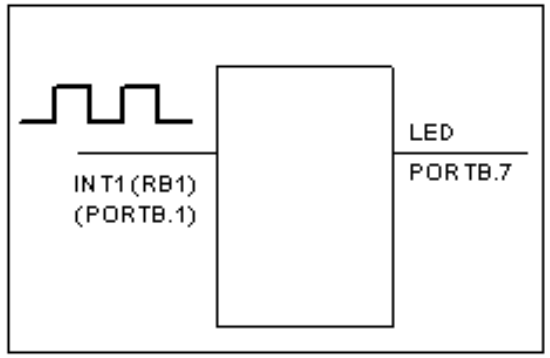


Program

```
                ORG 0000H
                GOTO MAIN

                                        High priority interrupt
                ORG 0008H
                BTFSS INTCON, INT0IF
                RETFIE
                GOTO INT0_ISR

                ORG 00100H
MAIN:
                BCF TRISB, 7
                BSF TRISB, 0
                CLRF TRISD
                SETF TRISC
                BCF INTCON, INT0IF
                BSF INTCON, INT0IE
                BSF INTCON, GIE
OVER:           MOVFF  PORTC, PORTD
                BRA OVER
INT0_ISR:
                ORG 200H
                BTG PORTB, 7
                BCF INTCON, INT0IF      if no INTOIF reset -> cont'd to toggle
                RETFIE
                END
```

Negative edge-triggered interrupt

```
                ORG 0000H
                GOTO MAIN

                ORG 0008H
                BTFSS INTCON3, INT1IF
                RETFIE
                GOTO INT1_ISR

                ORG 00100H
MAIN:
                BCF TRISB, 7
                BSF TRISB, 1
                BSF INTCON3, INT1IE
                BCF INTCON3, INT1IF
                BCF INTCON2, INTEDG1
                BSF INTCON, GIE
OVER:           MOVFF  PORTC, PORTD
                BRA OVER
INT1_ISR:
                ORG 200H
                BTG PORTB, 7
                BCF INTCON3, INT1IF
                RETFIE
                END
```



Sampling the edge-triggered interrupt

- The external source must be held high for at least two instruction cycles, and then held low for at least two instruction cycles

- For XTAL 10MHz
- Instruction cycle time is 400ns or 0.4μs
- So minimum pulse duration to detect edge-triggered interrupt = 2 instruction cycles = 0.8μs

Example

- connect a switch to INT0 and another switch to INT1
- every time INT0 is activated, increment the content in location 0
- every time INT1 is activated, decrement the content in location 0
- at the same time, data is being transferred from PORTC to PORTD

Program

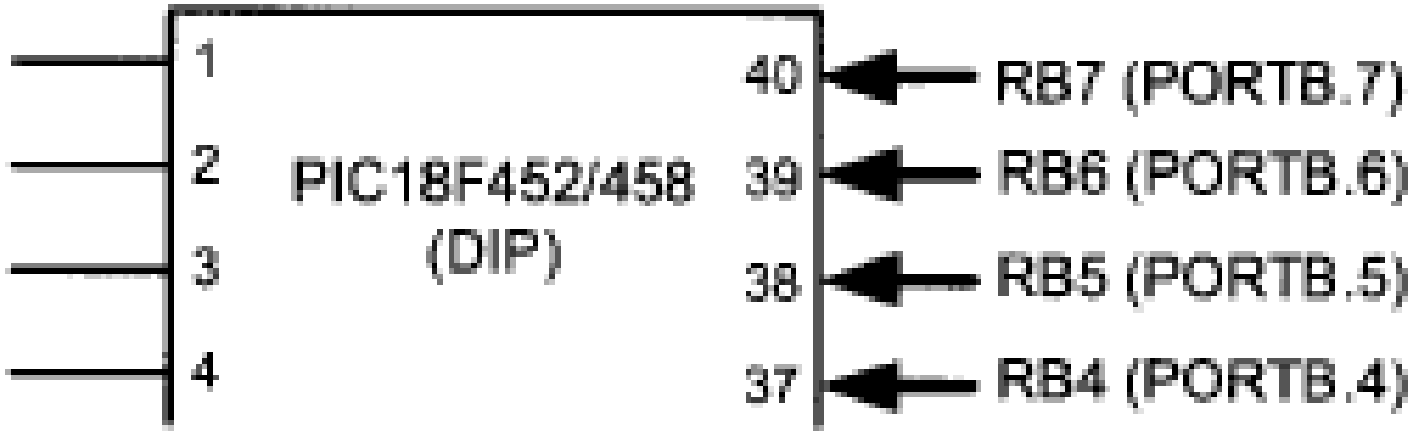
```
        ORG 0000H
        GOTO MAIN

        ORG 0008H
        BTFSC  INTCON, INT0IF
        RCALL  INT0_ISR
        BTFSC  INTCON3, INT1IF
        RCALL  INT1_ISR
        RETFIE

        ORG 00100H
MAIN:
        BSF TRISB, 0
        BSF TRISB, 1
        CLRF TRISD
        SETF TRISC

        BCF INTCON, INT0IF
        BSF INTCON, INT0IE
        BCF INTCON2, INTEDG0

        BSF INTCON3, INT1IE
        BCF INTCON3, INT1IF
        BCF INTCON2, INTEDG1

        BSF INTCON, GIE
OVER:   MOVFF  PORTC, PORTD
        BRA OVER

INT0_ISR:
        INCF 0, F
        BCF INTCON, INT0IF
        RETURN

INT1_ISR:
        DECF 0, F
        BCF INTCON3, INT1IF
        RETURN

        END
```

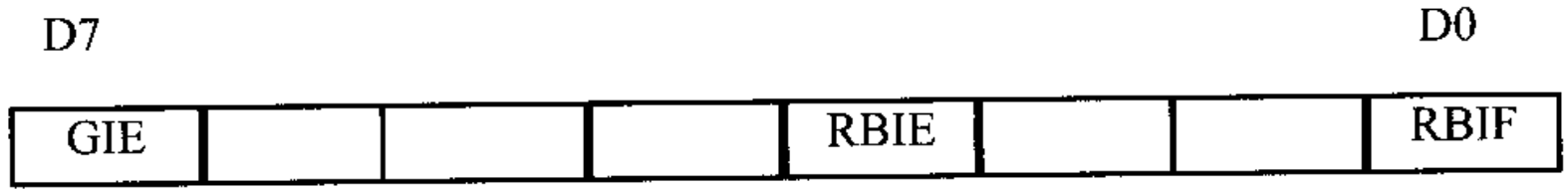
- Note that an ISR may change WREG, BSR, and Status register.
- If this happens, when returns from INT, the original information may lost.
- How to solve this problem?

PORTB-Change interrupt

- cause interrupt when any pin (RB4 to RB7) changes from high to low, or low to high

- commonly used in keypad interfacing





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.

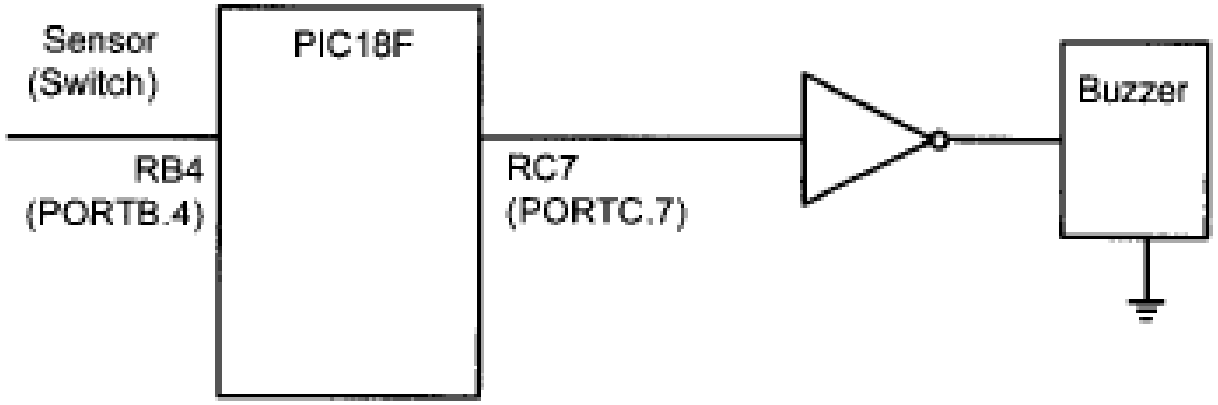
RBIE PORTB-Change Interrupt Enable
= 0 Disables PORTB-Change interrupt
= 1 Enables PORTB-Change interrupt

RBIF PORTB-Change Interrupt Flag.
= 0 None of the RB4–RB7 pins have changed state
= 1 At least one of the RB4–RB7 pins have changed state

The RBIE bit, along with the GIE, must be set high for any changes on the pins RB4–RB7 to cause an interrupt. The RB4–RB7 pins must also have been configured as input pins for this interrupt to work. In order to clear the RBIF flag we must read the pins of RB4–RB7 **and** use the instruction "BCF INTCON,RBIF".

Example

- connect a door sensor to pin RB4 and a buzzer to pin RC7
- every time the door is open, sound the buzzer by sending it a square wave for a while



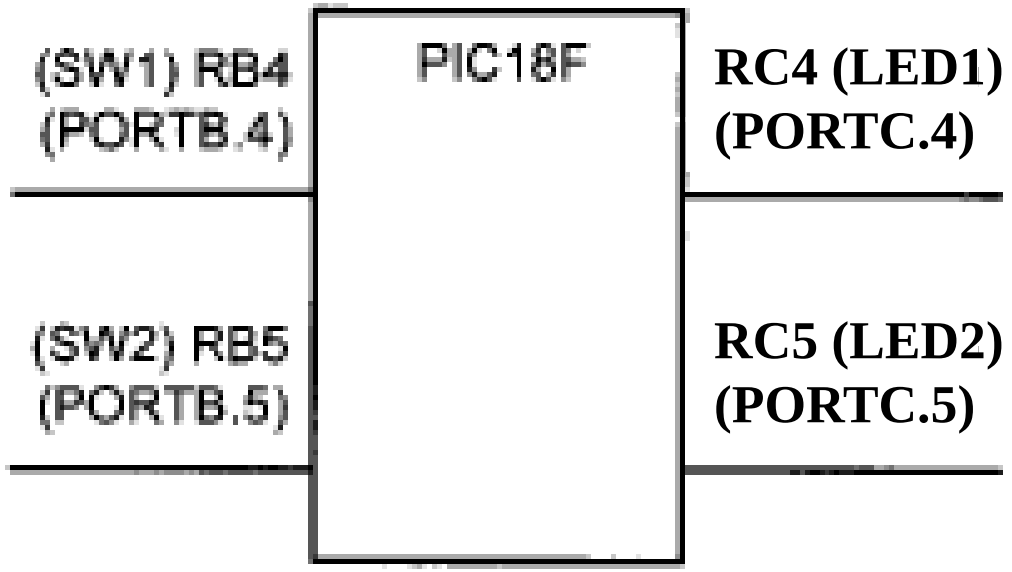
Program

```
        MYREG   EQU  0x20

DELRG   EQU 0x80
        ORG     0000H
        GOTO    MAIN

        ORG     0008H
        BTFSS INTCON, RBIF
        RETFIE
        GOTO  PB_ISR

        ORG     00100H
MAIN    BCF  TRISC, 7
        BSF  TRISB, 4
        BSF INTCON, RBIE
        BSF INTCON, GIE
OVER    BRA OVER
```

```
PB_ISR
        ORG 200H
        MOVF  PORTB, W
        MOVLW D'250'

        MOVWF MYREG
BUZZ    BTG PORTC, 7
        MOVLW D'255'

        MOVWF DELRG
DELAY   DECF DELRG, F
        BNZ DELAY

        DECF MYREG, F
        BNZ BUZZ
        BCF INTCON, RBIF
        RETFIE
        END
```

Another Example

- connect SW1 and SW2 to RB4 and RB5 respectively
- activate SW1 ➔ state change in LED1
- activate SW2 ➔ state change in LED2



Program

```
        ORG     0000H
        GOTO    MAIN

        ORG     0008H
        BTFSS INTCON,RBIF
        RETFIE
        GOTO   PB_ISR

        ORG     0100H
MAIN    BCF TRISC,4
        BCF TRISC,5
        BSF TRISB,4
        BSF TRISB,5
        BSF INTCON,RBIE
        BSF INTCON,GIE
OVER    BRA OVER
```

```
PB_ISR
        ORG 200H
        MOVFF PORTB,W
        ANDLW 0x30
        MOVFF W,PORTC
        BCF INTCON,RBIF
        RETFIE
        END
```

00110000 -> only keep bit 4 and 5

Modify the program if LED1 is connected to RC6 and LED2 is connected to RC7. RLNCF

Summary

- concept of programmed I/O and interrupt
- PIC18 interrupt