MECE336 Microprocessors I
Interrupts

Dr. Kurtuluş Erinç Akdoğan

kurtuluserinc@cankaya.edu.tr

Course Webpage: http://MECE336.cankaya.edu.tr

ÇANKAYA ÜNİVERSİTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

Working With Time: Interrupts

- Our daily lives are ruled by time
  - alarm clocks to wake us
- Embedded systems, needs to respond in a timely manner to external events.
- These requirements are met primarily by feature of a microcontroller: the **interrupt**
- **Interrupts** form part of the exciting techniques that underpin **real-time programming**

The Main Idea – Interrupts

Up to Now

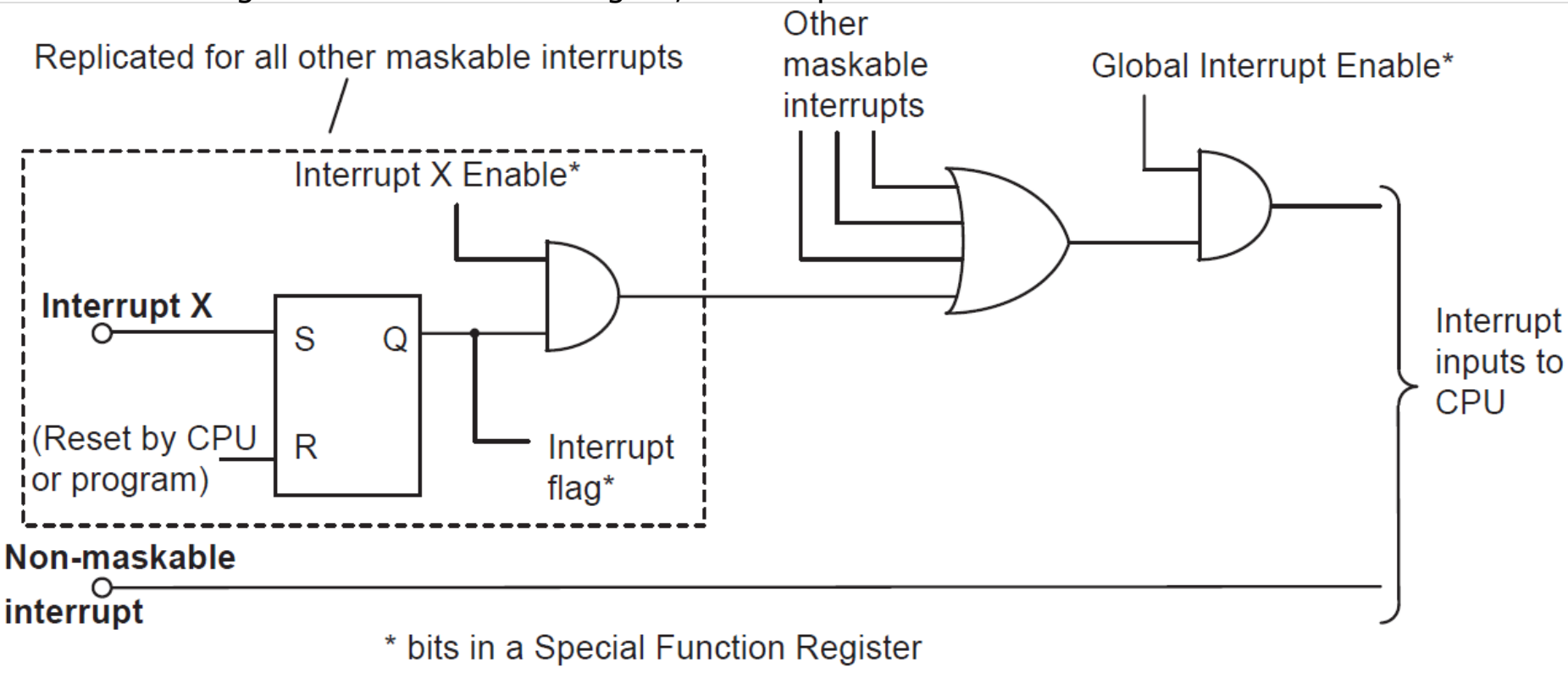
- Programs are written in a precise and predictable fashion
- Instructions are followed in a clear order and the update of the program counter is well-defined at each step of the program

Interrupt Idea

- Function of an interrupt is to alert the CPU that some **significant event** such as power failure, the system overheating, has happened
- Interrupt can occur at any time
- Effect of an interrupt is generally to stop the CPU from doing its current task and to force it to respond to the interrupt cause
- Interrupt disturbs predictable execution of a program
- There can be different interrupt sources on a microcontroller

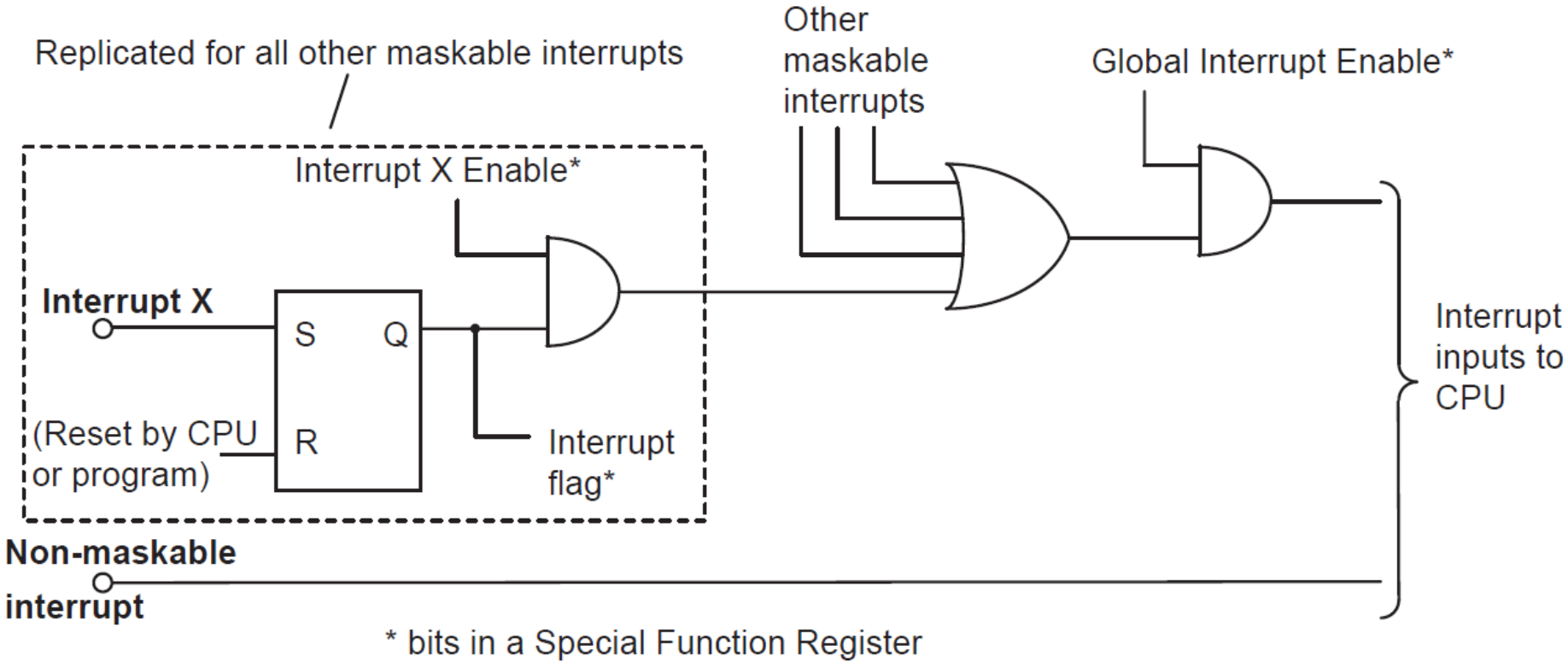
Interrupt structures

- Microcontrollers have more than one interrupt source generated
  - **internally and**
  - **externally.**
- Assume 'Interrupt X' occurs, sets an S–R bistable and its occurrence is recorded.
- The output of the is called the 'interrupt flag'.
- This is then gated with an enable signal, 'Interrupt X Enable'.



Interrupt structures

- If enable is high, then the interrupt signal progresses to an OR gate with other maskable enabled interrupt inputs.
- Any interrupt signal can reach the CPU as long as 'Global Interrupt Enable' is enabled.
- When the CPU has responded to an interrupt, it is necessary to clear the interrupt flag.
- The action of disabling an interrupt is sometimes called 'masking'.
- There are unmaskable which are always external and of the greatest importance



The 16F84A Interrupt Structure

External Interrupt

- This is the only **external hardware interrupt** input on PIC16F84A. It shares a pin with Port B, bit 0 (pin RB0). It is edge triggered.

Timer Overflow

- This is an interrupt caused by the Timer0 module. It occurs when the timers 8-bit counter overflows. This the **software interrupt** or **internal interrupt**.

Port B interrupt on change

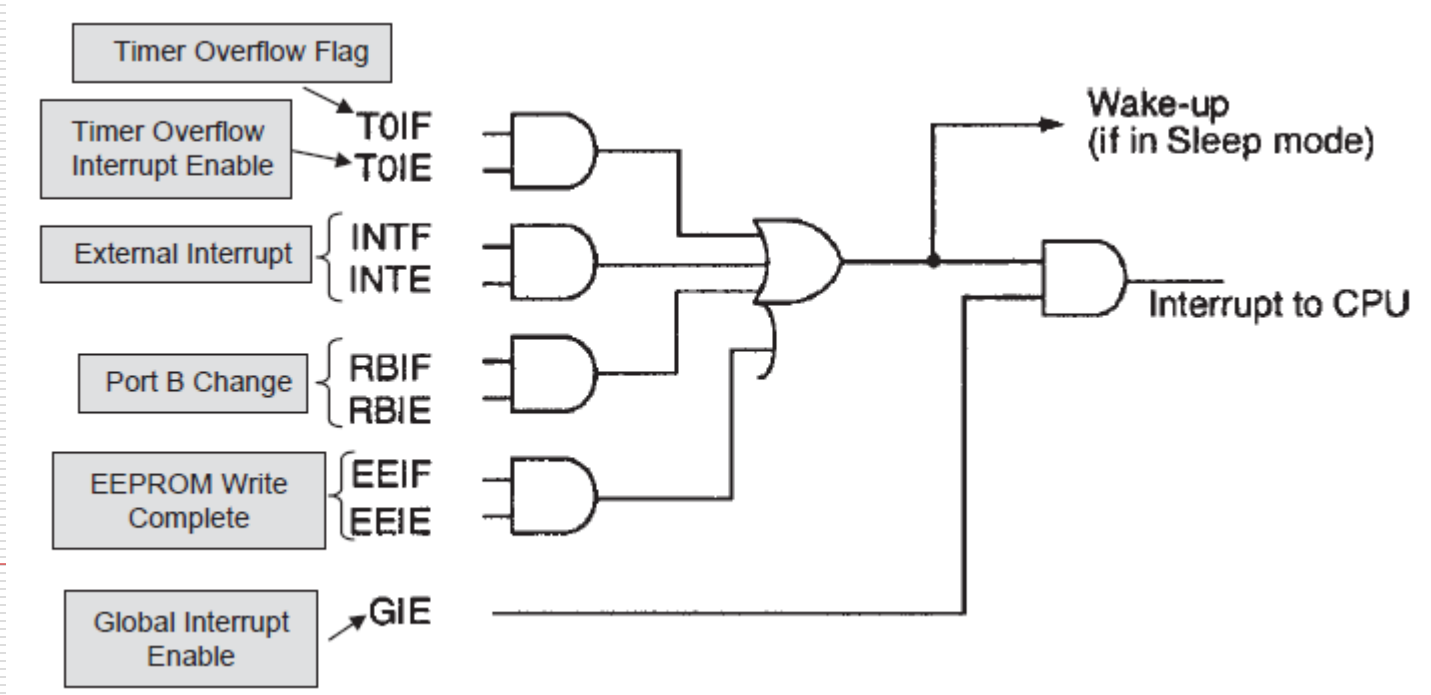
- This interrupt occurs when a change is detected on any of the higher four bits of Port B (RB4,RB5,RB6, and RB7).

EEPROM write complete

- This interrupt occurs when a write instruction to the EEPROM memory is completed (see later lectures).

Interrupt Logic

- The four interrupt sources appear labelled on the left of diagram.
- Each source has an enable line (**E**) and a flag line (**F**).
- The INTCON register contains the enable bits for all interrupt sources.



INTCON REGISTER (ADDRESS 0Bh, 8Bh)

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	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
0 = Disables the EE Write Complete interrupt

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT 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 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT 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

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

REGISTER FILE MAP - PIC16F84A

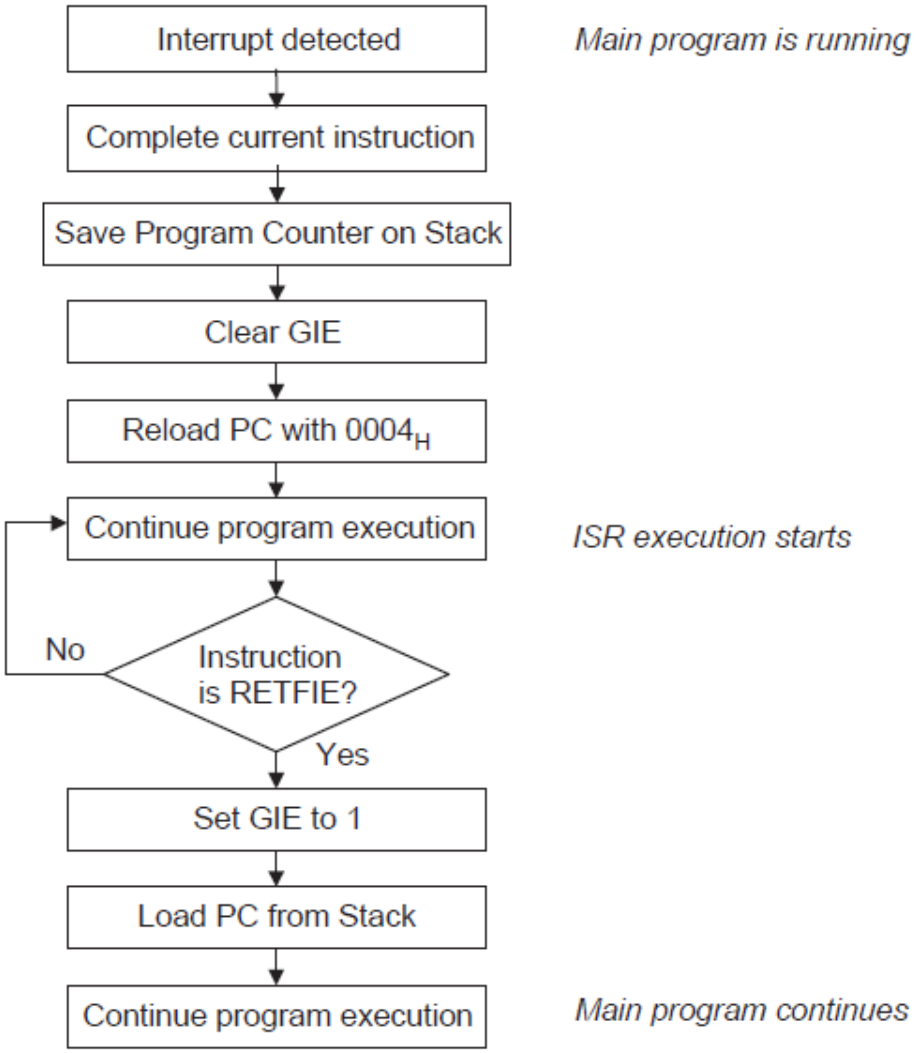
File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	—	—	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68 General Purpose Registers (SRAM)	Mapped (accesses) in Bank 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

The CPU Response To An Interrupt

- Assume that an interrupt has occurred, and both its local and the global enable are set.
- CPU executes a special section of program called the Interrupt Service Routine (ISR).
- CPU saves the value of the Program Counter on the top of the Stack to 'know' where to come back to when the ISR is complete.
- To avoid other interrupts possibly interrupting this interrupt, it also clears the Global Interrupt Enable.
- Program Counter is loaded with memory location 0004 as ISR must start at here.
- ISR must end with **retfie** instruction.
- When this is detected, the CPU sets the GIE to 1, loads the Program Counter from the top of the Stack and then resumes program execution.
- Thus, it returns to the instruction which follows the instruction during which the interrupt was detected.



Programming With A Single Interrupt

For a succesful interrupt application

- Start the ISR at the interrupt vector, location 0004.
- Enable the interrupt that is to be used by setting the enable bit in the INTCON register.
- Set the Global Enable bit, GIE.
- Clear the interrupt flag within the ISR.
- End the ISR with a retfie instruction.
- Ensure that the interrupt source, for example Port B or Timer 0, is actually set up to generate interrupts.
- The program starts as usual at the reset vector 0000;

Example Program

- branches over the reset vector to location start, where initialisation takes place.
- Within this we see the GIE and INTE bits being set
- The main program simply outputs the bit patterns 0AH and 15H to Port A
- When an interrupt occurs the interrupt vector address is loaded into the Program Counter, from where program execution continues.
- The first action of the ISR is to jump to location Int Routine. This is placed at program memory location 0080H to give clarity to the simulation.
- The ISR simply clears Port A before clearing its interrupt flag and returning to the main program.

```
;*****************************************************
;Int_Demo1
;This program demonstrates simple interrupts.
;Intended for simulation.
;tjw rev.14.2.09            Tested in simulation 14.9.09
;*****************************************************
;
      include p16f84A.inc
;Port A all output
;Port B: bit 0 = Interrupt Input
;
       org    00
       goto   start
;
       org    04     ;here if interrupt occurs
       goto   Int_Routine
;
       org    0010
;Initialise
start  bsf    status,rp0   ;select bank 1
       movlw  01
       movwf  trisb        ;portb bits 1-7 output
                           ;      bit 0 is input
       movlw  00
       movwf  trisa        ;porta bits all output
;Comment in or out following instruction to change
;interrupt edge
;      bcf    option_reg,intedg
       bcf    status,rp0  ;select bank 0
       bsf    intcon,inte ;enable external interrupt
       bsf    intcon,gie  ;enable global int

wait   movlw  0a     ;set up initial port output values
       movwf  porta
       nop
       movlw  15
       movwf  porta
       goto   wait
;
       org    0080
Int_Routine
       movlw  00
       movwf  porta
       bcf    intcon,intf  ;clear the interrupt flag
       retfie
       end
```

Interrupt Subroutine: General Idea

- The interrupt subroutine is called from the program memory location

  0004 using a goto instruction

  org 0x04

  goto interrupt subroutine

- To disable further interrupts during the interrupt subroutine, the interrupt enable flag should be reset

  bcf INTCON,INTE; (for external interrupt)

- The interrupt flag should be reset at the end of the subroutine

  bcf INTCON,INTF; (for external interrupt)

- The content of the W register and STATUS register should be saved

  movwf TEMP W;

  swapf STATUS,0;

  movwf TEMP S;

General Structure For An Assembly Program With External Interrupt Subroutine

```
        LIST P=16F84A
        INCLUDE "P16F84A.INC"
        ORG     0X000    ; address of the main program
        GOTO    START
        ORG     0X004    ;address of the ISR
        GOTO    MY_ISR
START
        BSF      INTCON, GIE      ;global interrupt enable
        BSF     INTCON, INTE      ; external interrupt enable
.................

LOOP
        GOTO     LOOP
MY_ISR
        BCF     INTCON, INTF      ; clear interrupt flag
.....................
        RETFIE
        END
```

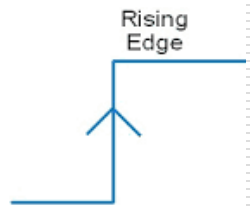
External interrupts and OPTION register

For external interrupts,

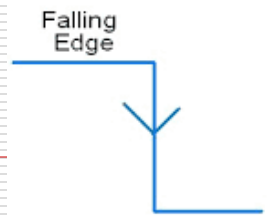
- 1. RB0 must be input.
- 2. INTE must be 1.
- 3. Bit_6 of the OPTION register (INTEDG) is the interrupt edge select bit;

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

- If INTEDG= 1, interrupt occurs rising edge of the signal.



- If INTEDG= 0, interrupt occurs falling edge of the signal.



- Depends on the hardware, INTEDG must be 0 or 1.

REGISTER FILE MAP - PIC16F84A

File Address			File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	—	—	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68 General Purpose Registers (SRAM)	Mapped (accesses) in Bank 0	8Ch
4Fh			CFh
50h			D0h
7Fh	Bank 0	Bank 1	FFh

Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

swapf

- swapf f,d: swaps the lower nibble and the higher nibble of the content in file register f. In other words, the lower 4 bits are put into the higher 4 bits, and the higher 4 bits are put into the lower 4 bits. Write the result to
  - Working register W if d is 0
  - File register f if d is 1

Example

- Find the contents of the MYREG register in the following code.

Solution

```
MYREG EQU 0X20
MOVLW 0X72 ;WREG=72
MOVWF MYREG ;MYREG=72
SWAPF MYREG,F ;MYREG=27
```

Protect The Contents Of The Working Register And STATUS Register

- Interrupt subroutine should be written as

```
.....
        ORG     h'004'
        GOTO MY_ISR
.....
MY_ISR
        MOVWF       SAVE_W          ; SAVE_W=W_initial
        SWAPF       STATUS,W        ;W=SWAP STATUS_initial
        MOVWF       SAVE_S ; SAVE_S=SWAP STATUS_initial
        .....
        SWAPF       SAVE_S, W  ;W=STATUS
        MOVWF       STATUS      ;STATUS=STATUS_initial
        SWAPF       SAVE_W, F  ;W=SWAP W_initial
        SWAPF       SAVE_W, W  ;W=W_initial
        RETFIE
```

Moving To Multiple Interrupts – Identifying The Source

- 16F84A has four interrupt sources but only one interrupt vector.
- Therefore, if more than one interrupt is enabled, it is not obvious at the beginning of an ISR which interrupt has occurred.
- In this case the programmer must write the ISR so that at its beginning it tests the flags of all possible interrupts and determines from this which one has been called.

```
interrupt btfsc intcon,0       ;test RBIF
       goto portb_int
       btfsc intcon,1          ;test external interrupt flag
       goto ext_int
       btfsc intcon,2          ;test timer overflow flag
       goto timer_int
       btfsc eecon1,4          ;test EEPROM write complete flag
       goto eeprom_int

portb_int
...
place portb change ISR here
...
       bcf    intcon,0         ;and clear the interrupt flag
       retfie

ext_int
...
place external interrupt ISR here
...
       bcf    intcon,1         ;and clear the interrupt flag
       retfie

timer_int
...
place timer overflow ISR goes here
...
       bcf    intcon,2         ;and clear the interrupt flag
       retfie
       eeprom_int
...
place EEPROM write complete ISR here
...
       bcf    eecon1,4         ;and clear the interrupt flag
       retfie
```

Example: Multiple Interrupts

```
        LIST P=16F84A
        INCLUDE "P16F84A.INC"
        ORG     0X000     ; address of the main program
        GOTO    START
        ORG     0X004     ;address of the ISR
        BTFSC   INTCON,INTF
        GOTO    MY_ISR_RB0_INT
        BTFSC   INTCON,RBIF
        GOTO    MY_ISR_RB
        BTFSC   INTCON,TOIF
        GOTO    MY_ISR_TO
        GOTO    MY_ISR_EE
START
        BSF     INTCON, GIE       ;global interrupt enable
        BSF     INTCON, INTE      ; external interrupt enable
        BSF     INTCON, RBIE      ;PORTB_change interrupt enable
        BSF     INTCON, TOIE      ; Timer overflow interrupt enable
        BSF     INTCON, EEIE      ;eeprom interrupt enable
.....
LOOP
        GOTO    LOOP
```

```
MY_ISR_RB0_INT
        BCF     INTCON, INTF    ; clear interrupt flag
        BSF     PORTB,1
        RETFIE
MY_ISR_RB
        BCF     INTCON, RBIF    ; clear interrupt flag
        BSF     PORTB,2
        RETFIE
MY_ISR_TO
        BCF     INTCON, TOIF    ; clear interrupt flag
        BSF     PORTB,3
        RETFIE
MY_ISR_EE
        BSF     PORTB,0
        RETFIE
        END
```

Example

```
LIST P=16F84A
   INCLUDE "P16f84A.INC"
   __config _CP_OFF&_WDT_OFF&_XT_OSC
   org 0x00;
   goto START
   org 0x04;
   GOTO    ISR     ; go to interrupt service routine


MAIN_PROG CODE                      ; let linker place main program

START
   BSF STATUS, RP0
   CLRF TRISA   ;set all PORTA as OUTPUT
   MOVLW 0xF0
   MOVWF TRISB   ;buttons attached to RB4 to RB7
   BCF STATUS, RP0   ;go to bank 0
   MOVLW b'10001000'
   MOVWF INTCON   ;Global interrupt enabled, RB Change interrupt enabled
   GOTO MAIN

;Main routine--------------------------------------------------------------------
MAIN
   BSF PORTA,0 ;Set RA.0
   GOTO MAIN ;Loop

;Interrupt service routine-------------------------------------------------------
ISR
   BCF INTCON, GIE ;Disable all interrupts inside interrupt service routine
   BCF PORTA,0 ;clear RA.0
   BCF INTCON,RBIF ;Clear external interrupt flag bit
   BSF INTCON, GIE ;Enable all interrupts on exit
   GOTO MAIN


   END
```