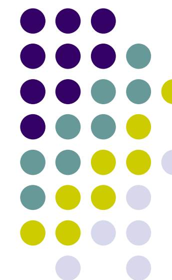
Embedded Systems
Programming and Architectures

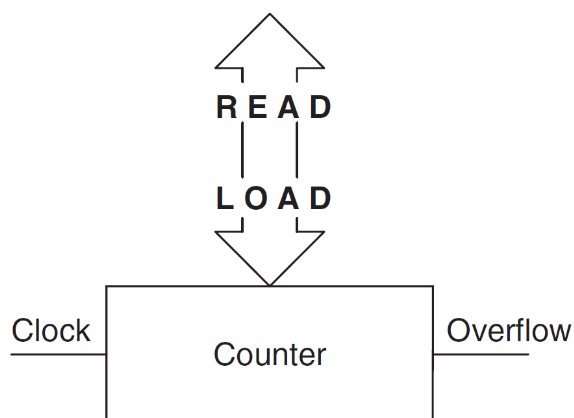
Lecture No 8 : Programming timers and interrupts in assembly

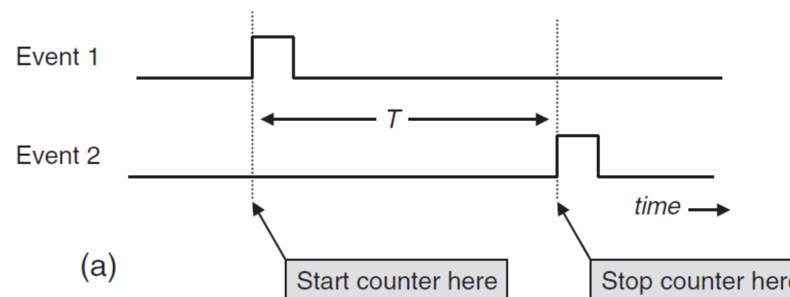
Dr John Kalomiros
Assis. Professor
Department of Post Graduate studies in Communications and Informatics



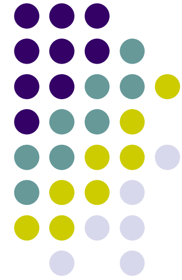
What is a timer?

1. It is a peripheral to almost every microprocessor system
2. Its functionality is based on a digital counter
3. It can be used (a) to count events or (b) to measure time intervals with accuracy

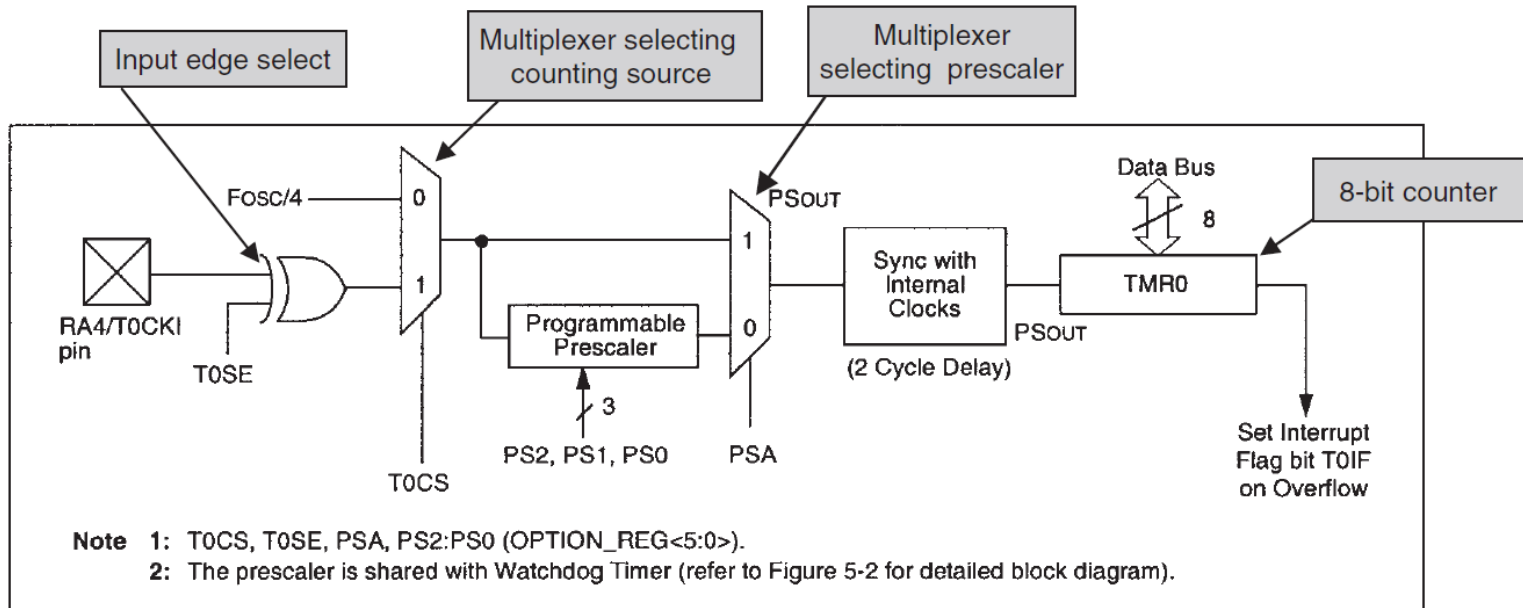




The resolution of the timer is determined by the resolution of the clock signal

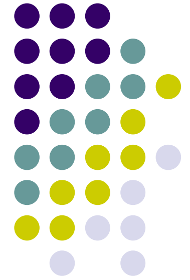


Timer0 module in the 16F series



The counter is associated with register TMR0, which can be written in order to load initial value. Reading TMR0 can be translated to elapsed time.

Internal Timing issues



If external frequency is f_{osc}, then $T_{osc}=1/ f_{osc}$

For example with f_{osc}=4MHz, T_{osc}=0,25µs

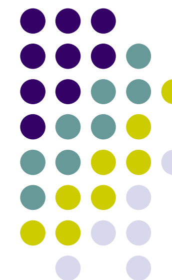
Execution of one instruction takes 4 cycles of the external clock:

fetch, decode, execute and store = 1 instruction cycle

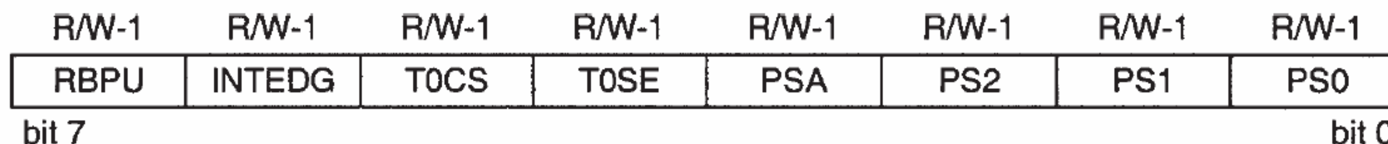
In other words: Duration of 1 instruction cycle = $4*T_{osc}$

For example with f_{osc}=4MHz, 1 instruction takes 1µs

This is the internal instruction clock: $f_{int} = f_{osc} /4$



OPTION register (bank 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
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

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

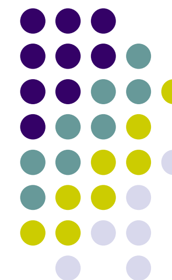
bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer 0 module

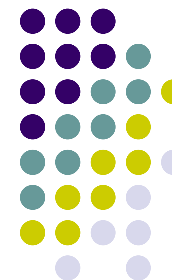
bit 2-0 **PS2:PS0**: Prescaler Rate Select bits



Timer0 Prescaler

Τιμή προμετρητή			Ρυθμός TMR0
PS2	PS1	PS0	
0	0	0	1:2
0	0	1	1:4
0	1	0	1:8
0	1	1	1:16
1	0	0	1:32
1	0	1	1:64
1	1	0	1:128
1	1	1	1:256

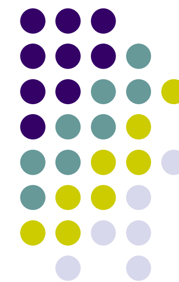
The clock frequency that increments the timer can be divided by using the prescaler. In this case **PSA=0** and PS2, PS1, PS0 are set to a dividing configuration, as above.



Except from OPTION Reg, INTCON Reg is also connected to the function of Timer0

b7	b6	b5	b4	b3	b2	b1	b0
GIE	-	T0IE	INTE	RBIE	T0IF	INTF	RBIF

INTCON is in Bank 0, and controls the function of interrupt signals.
A Flag in INTCON is associated with timer0.
This is T0IF (timer0 interrupt flag) which is set when the timer overflows.

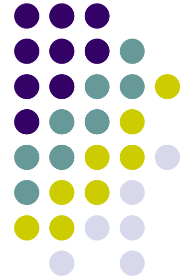


In order to enable the timer and have it running you need to initialize it:

1. Go to bank 1
2. Write OPTION Reg using the necessary timer and prescaler settings
3. Clear Timer0 interrupt Flag (T0IF) in INTCON Reg
4. Load the initial counting value in TMR0

When you load Timer0, the following series of events takes place

1. TMR0 value is incremented at each cycle of the instruction clock (nominally after each instruction is executed, unless it is a branch instruction)
2. When the counter reaches 255 it overflows at the very next clock cycle
3. Upon overflow T0IF is set in INTCON Reg
4. If you want the timer to run again then you have to clear T0IF and reload



How the timer delay is calculated

the time interval from loading the counter until overflow is given by the following relation

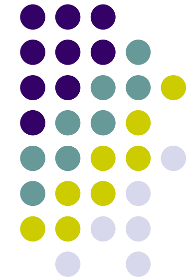
$$\text{Delay} = \frac{(256\text{-initial value in TMR0})^* \text{ prescaler ratio}^*4}{\text{Crystal frequency}}$$

For example: f_{osc}= 8MHz, prescaler is 1:256

(OPTION Reg is written with 11010111)

initial TMR0 value =178

$$\text{Delay} = \frac{(256-178)^* \; 256^*4}{8^*10^6 \; s^{-1}} = 0,01 \; s$$



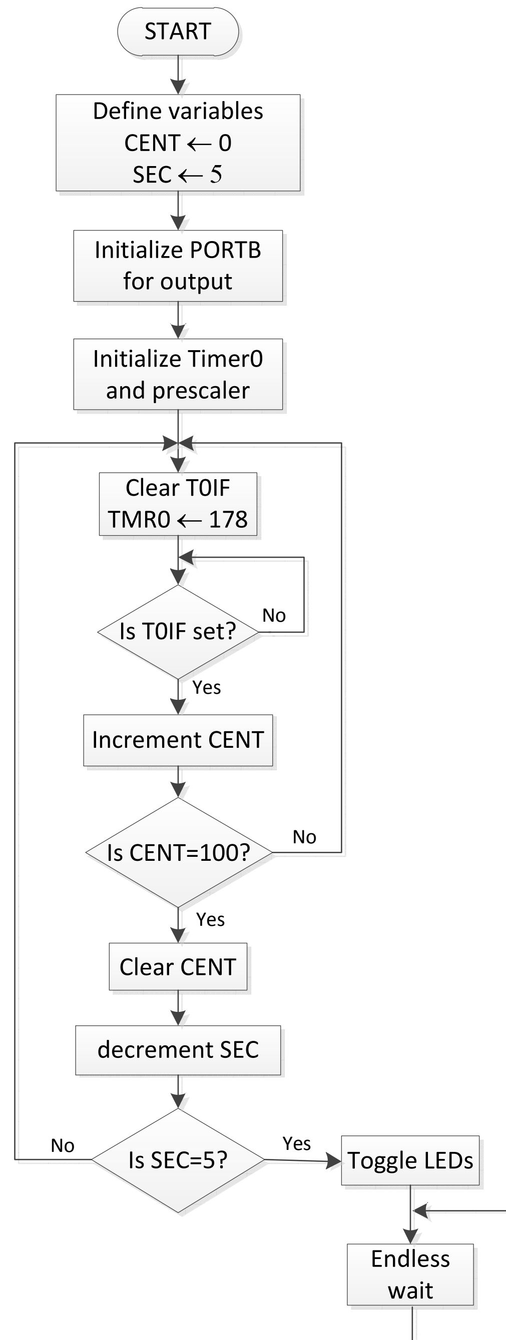
Example timer0 programming, for simulation

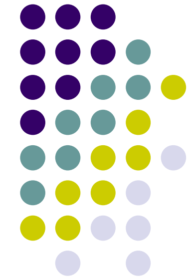
```
#include "P16F877.inc"
          ;initialization
          Org 0                 ;Reset vector
          bsf STATUS, RP0       ;switch to bank 1
          movlw b'11010001'     ;prescaler set to division by 4
          movwf OPTION_REG      ;write to OPTION REG
          bcf STATUS, RP0       ;return to bank 0
          movlw 0F0h            ;Set TMR0 to initial value 240 (decimal)
          movwf TMR0
          bcf INTCON, T0IF      ;clear Timer 0 interrupt flag
          ;main loop
loop      goto loop             ;wait for timer overflow
end
```

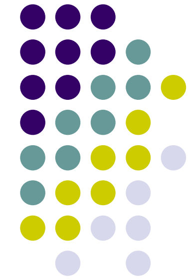
Measure 5 sec with Timer0

suppose external crystal
with frequency *fosc*=8MHz

CENT holds hundredths of sec
SEC holds number of seconds

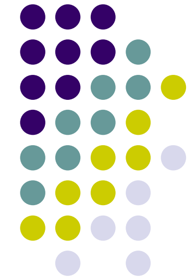






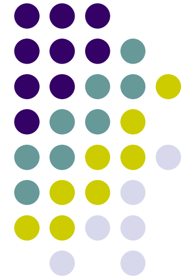
Measure 5 sec with Timer0

```
#include "P16F877.INC"
        Org 0
CENT equ 20h            ; define memory location CENT (hundredths of second)
SEC equu 21h            ; define memory location SEC (it holds seconds)
        movlw d'5'
        movwf SEC               ;Variable SEC is 5 (decimal)
        clrf CENT               ;Clear CENT
        bsf STATUS, RP0         ; Go to bank 1
        movlw b‘00000000'
        movwf TRISB             ; PORTB is output
        movlw b'11010111'
        movwf OPTION_REG        ; define prescaler 1/256 PS2:PS0=111
        bcf STATUS, RP0         ;return to bank 0
        movlw b'01010101'       ;Output a light motive to PORTB
        movwf PORTB
```



Measure 5 sec with Timer0 (continued)

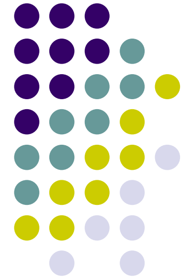
```
loop1  movlw d'178'
       movwf TMR0              ;Delay=(256-178)*256*4/8 in µsec (=0,01 sec)
       bcf INTCON, T0IF        ;Clear flag T0IF
loop2  btfss INTCON, T0IF      ;Wait for 0,01 sec
       goto loop2
       incf CENT,1
       movlw d'100'
       subwf CENT,w
       btfss STATUS,Z
       goto loop1              ;Repeat timer delay 100 times for  total delay 1 sec
       clrf CENT
       decfsz SEC,f
       goto loop1              ; Repeat SEC times
       comf  PORTB             ;Toggle the LEDs
loop3  goto loop3
       END
```



Project No 4

- Design a real-time clock that toggles an LED at b7 of PORTB every one second, while it uses the LEDs at the lower 6 bits of PORTB in order to count up to 60 secs. The system is reset every minute.

Interrupt signals in the 16F series



- An embedded system often needs to respond to external events in a timely manner.

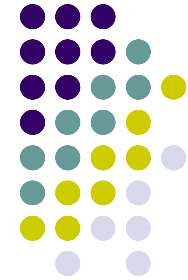
  Such events can be

  1. a subsystem dealing with an emergency and needing the intervention of the CPU (for example, power failure, overheating, a dangerous occurrence in the external world etc).

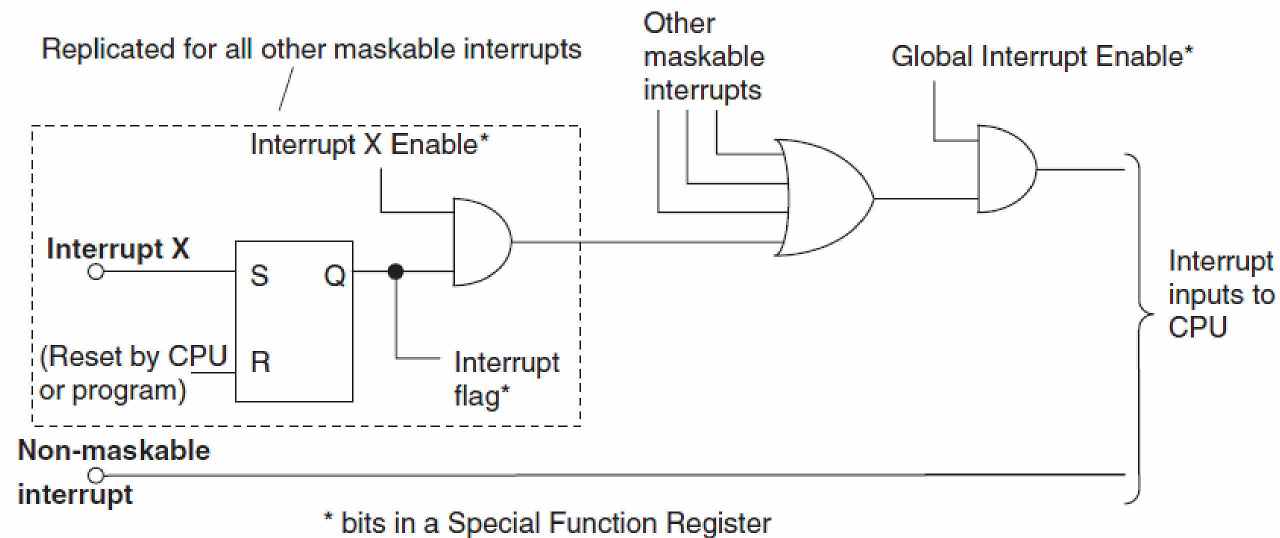
  2. an on-chip peripheral needing to exchange data with the CPU

  Therefore, an interrupt signal can originate from various sources

  Upon receiving an interrupt, the CPU disables the orderly execution of the program as soon as possible and starts executing another routine in order to service the interrupt. This is the ISR (*interrupt service routine*).



Generic interrupt structure



An interrupt that can be disabled is a maskable interrupt

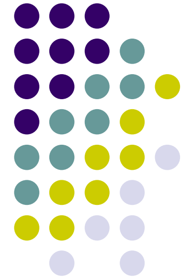
Interrupts are "stored" at an S-R bistable latch.

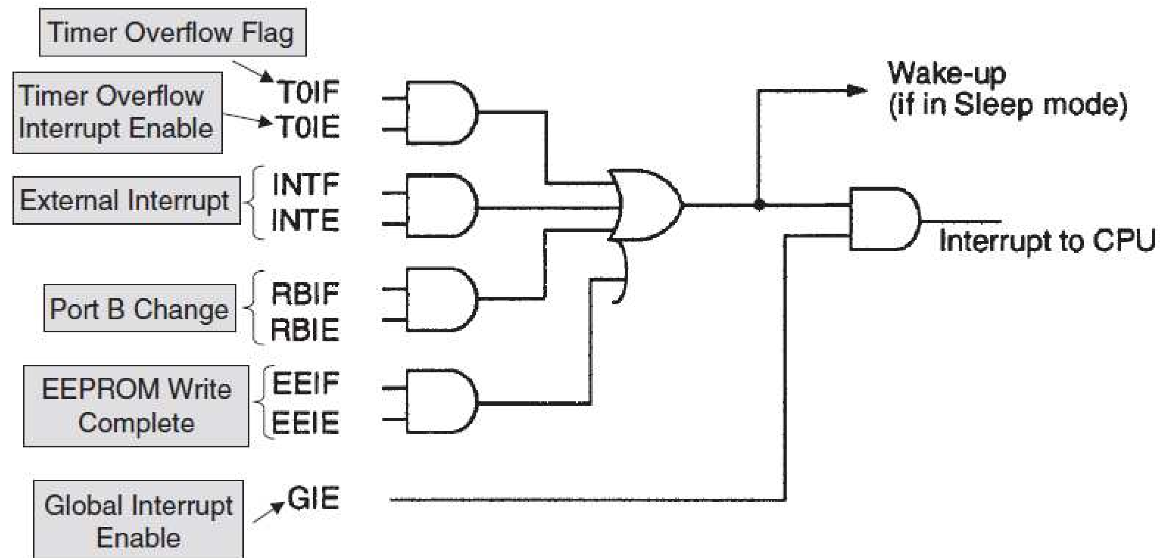
Different maskable sources are ORed before reaching a CPU and a *priority* may be given

Interrupt sources in the 16F series:

External interrupt (RB0 or INT)
Timer0 overflow
PORTB interrupt on change
EEPROM write complete

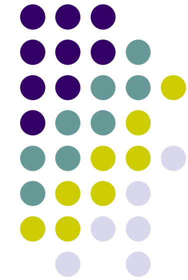
Each interrupt source is associated with an enable bit and an Interrupt flag. These are bits in the INTCON Register.





External interrupts are edge-triggered. The edge the interrupt responds to is controlled by the setting of the INTEDG bit of the OPTION Register

The 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	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

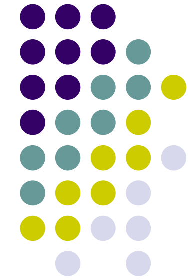
bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

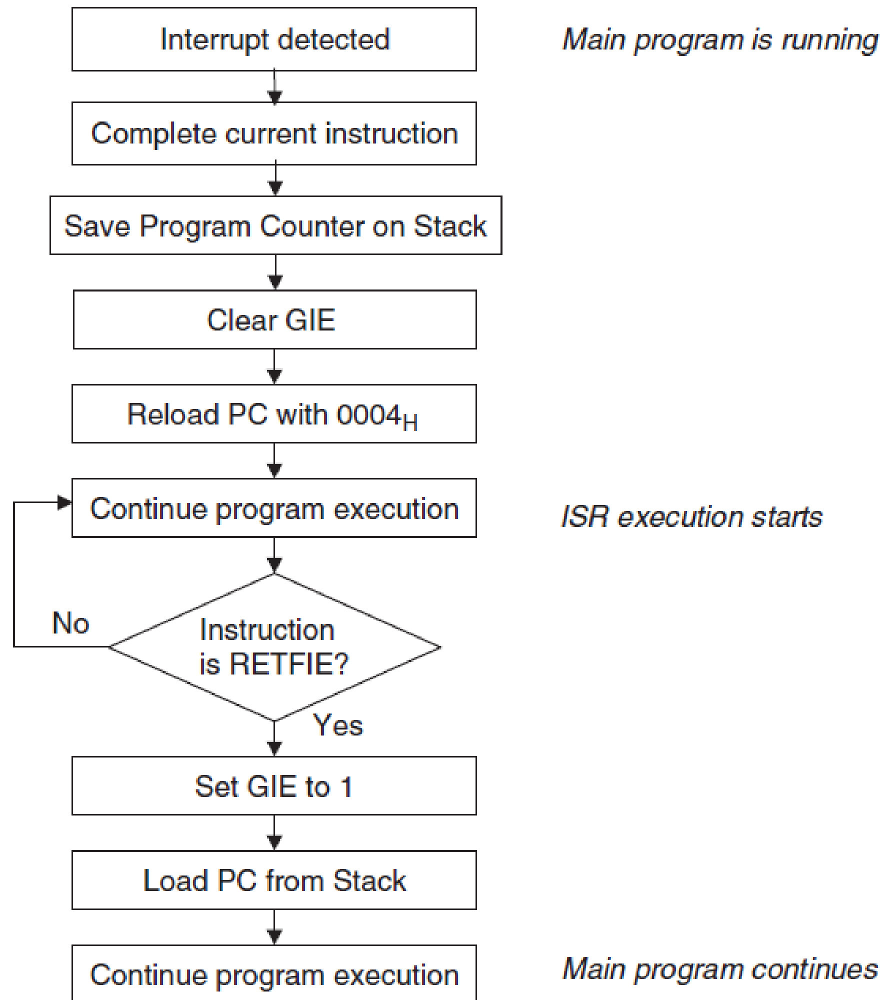
bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

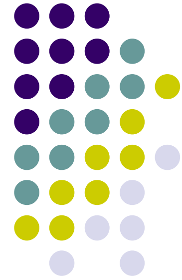
bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

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



The CPU response to an interrupt



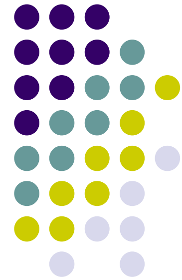


How to program interrupts

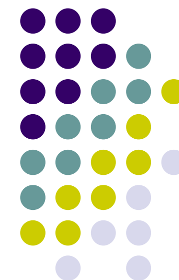
- Start the ISR at the interrupt vector (location 0004 of program memory)
- Enable the interrupt that is to be used, by setting the enable bit in the INTCON Reg
- Set the Global Interrupt Enable (GIE) bit
- Clear the Interrupt flag within the ISR
- End the ISR with a **retfie** instruction
- Don't forget to set up the interrupt source as an input!

Beware: upon interrupt the 16F series CPU loads PC with 04h.
This is the interrupt vector.
You may write the ISR starting from this address or you may have the program jump to the ISR using a **goto** *ISR* instruction in 04h.

A program that uses interrupts. Appropriate for simulation!



```
#include "P16F877.inc"
TEMP equ 20h
        Org 0                   ;Reset vector is 00h
        goto start              ; Go to the main program
        Org 4                   ;Interrupt vector is 04h
        incf TEMP, F            ;Increment TEMP, count interrupts!
        movf TEMP,W
        movwf PORTC             ;Contents of TEMP are shown on LEDs of PORTC
        bcf INTCON, INTF        ;Clear Interrupt flag INTF
        retfie                  ;Return form interrupt and enable GIE
start
        clrf TEMP               ;zero TEMP
        bsf STATUS, RP0         ;go to bank 1
        movlw b'00000000'
        movwf TRISC             ;Initialize PORTC as output
        bcf STATUS, RP0         ;go back to bank 0
        bsf INTCON, INTE        ;Enable external interrupt INT
        bcf INTCON, INTF        ;Clear interrupt flag INTF
        bsf INTCON, GIE         ;Enable Interrupts
loop goto loop                  ;main loop: wait for interrupt!
       END
```

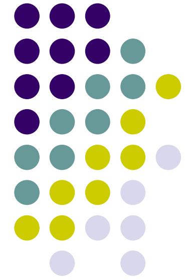


Project 5: The timer triggers an interrupt

Design an application which uses Timer0 in order to produce an interrupt signal with frequency 20 Hz. Consider f_{osc}=4MHz.

The ISR increments the content of a memory location.

Back in main program the result is displayed on PORTB.



Required reading:

You are expected to study chapter 6 (Working with time: Interrupts, counters and timers) in *Designing Embedded Systems with PIC microcontrollers* by Tim Wilmshurst.