INTEGRATED TECHNICAL EDUCATION CLUSTER
AT ALAMEERIA

E-626-A
Real-Time Embedded Systems (RTES)

Lecture #5
Working with Time
(Interrupts, Counters and Timers)

Instructor:
Dr. Ahmad El-Banna

SPRING 2015

Agenda

- Interrupt Structure
- Working with Interrupts
- Counters
- Timers
- Watchdog Timer & Sleep Mode

INTERRUPTS

The main idea – interrupts

- Computer **CPU** is a deeply **orderly entity**, following the instructions of the program **one by one** and doing what it is told in a precise and predictable fashion.
- **An interrupt disturbs this order**.
- Its **function** is to **alert the CPU** in no uncertain terms that some significant **external event has happened**, to stop it from what it is doing and **force it** (at the greatest speed possible) to **respond** to what has happened.
- **Originally** interrupts were applied to allow **emergency external events**, such as power failure, the system overheating or major failure of a subsystem to get the attention of the CPU.
- But the **concept** of interrupts was recognized as being very **powerful**.

Interrupt Classifications

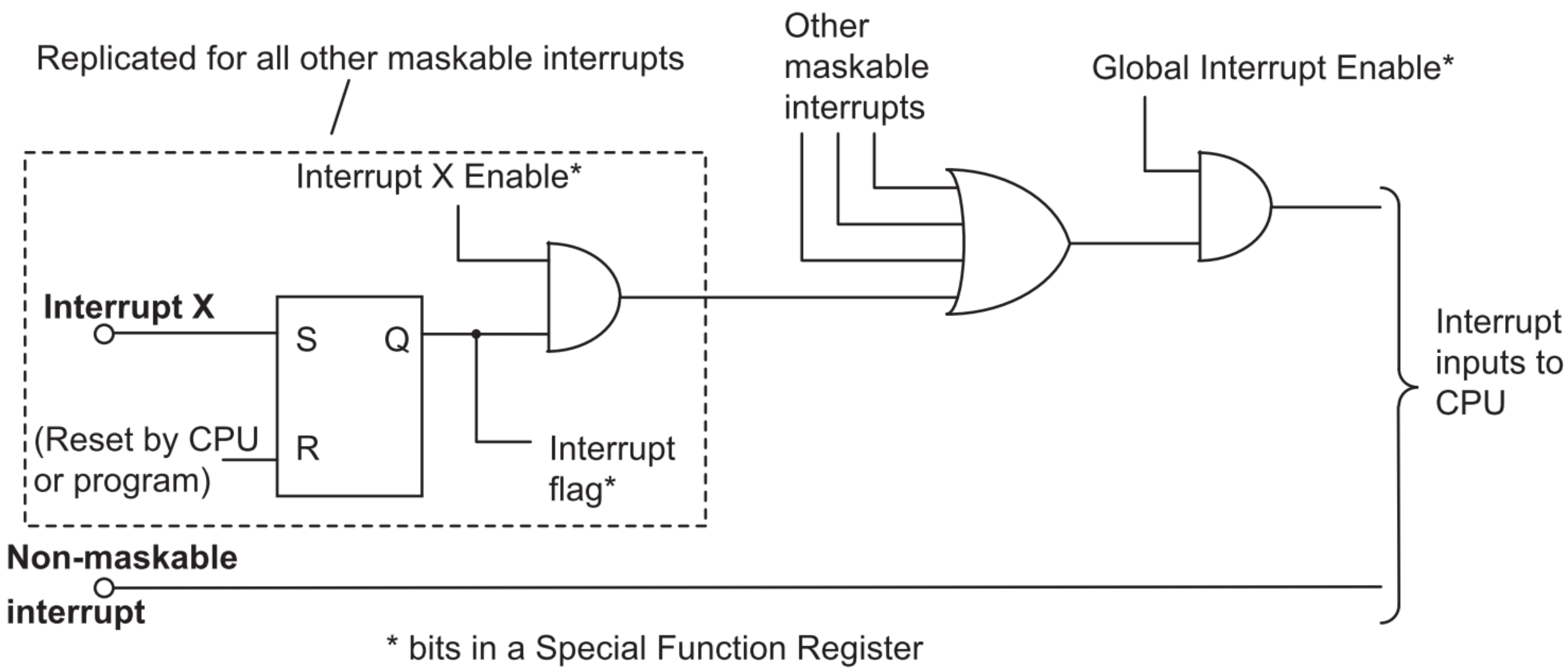
Some categories are:

- Internal/ External
- Hardware/ Software
- Maskable/ Non-maskable

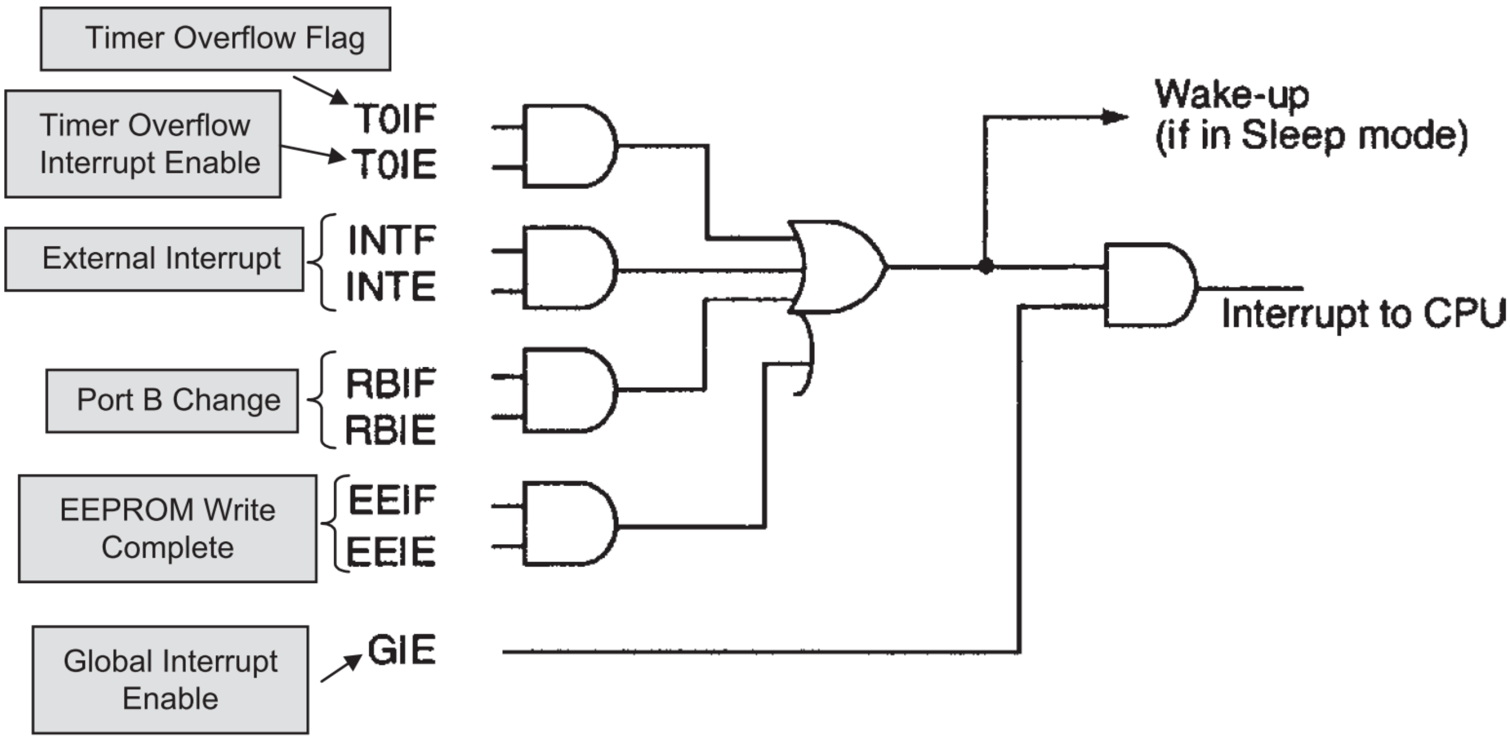
Interrupt structures

- Different microcontrollers have rather **different interrupt structures**.
- They have **more than one interrupt source**, usually with some **internally** generated and others **external**.

- **A simple generic interrupt structure**



Example:
The 16F84A interrupt structure



INTCON register (16F84A)

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

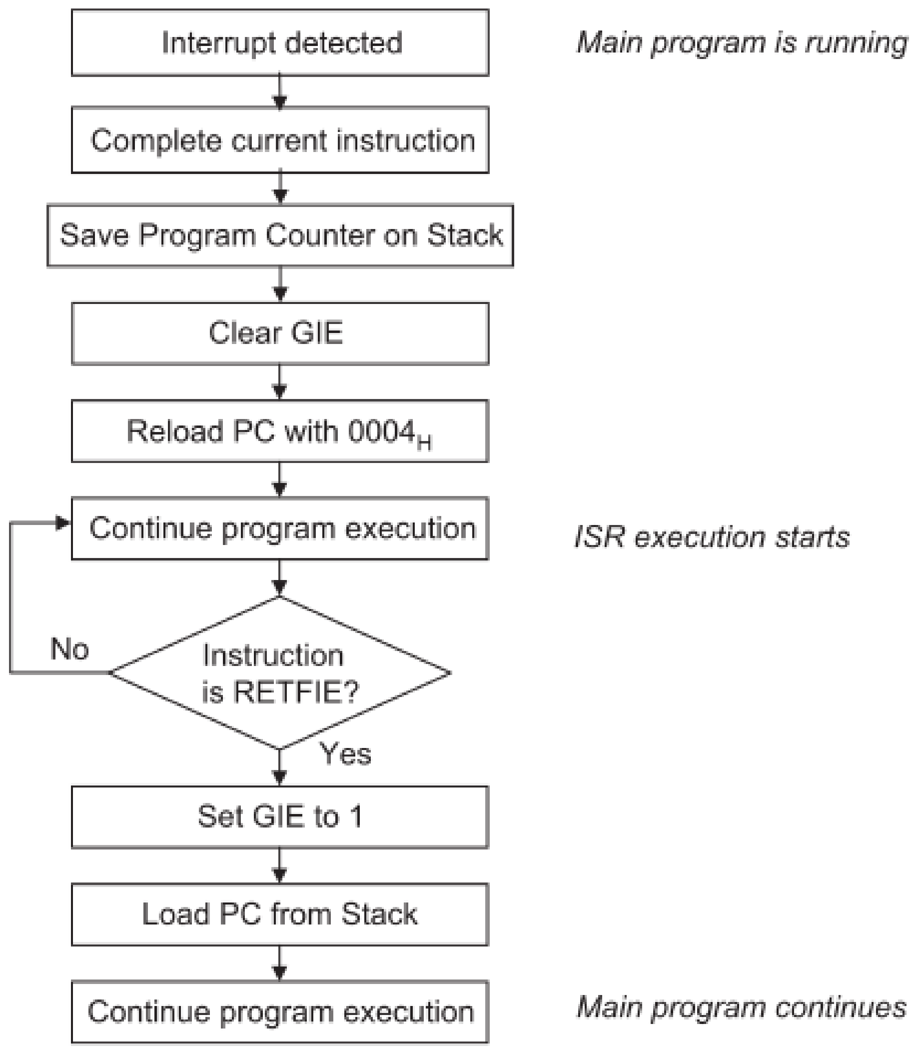
bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

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

Interrupt Response Sequence of Events



Working with Interrupts

- It is easy to write **simple programs** with just one interrupt.
- Using Assembly language!
- For success, the essential points to watch are:
  - **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**!

Example

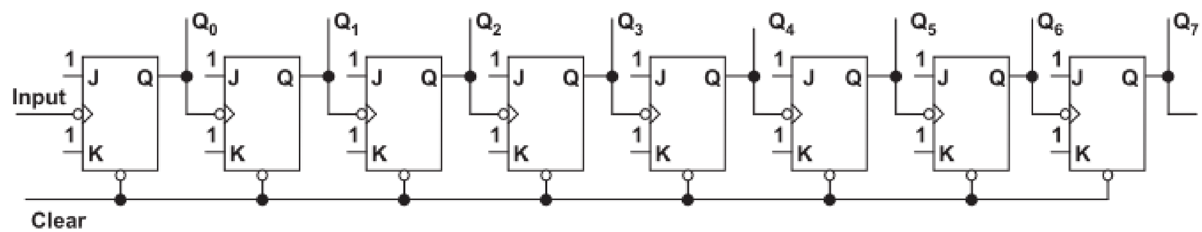
```
;*********************************************************
;Int_Demo1
;This program demonstrates simple interrupts.
;Intended for simulation.
;tjw rev.14.2.09           Tested in simulation 14.9.09
;*********************************************************
;
       include p16f84A.inc
;Port A all 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
```

COUNTERS & TIMERS

The main idea – counters and timers

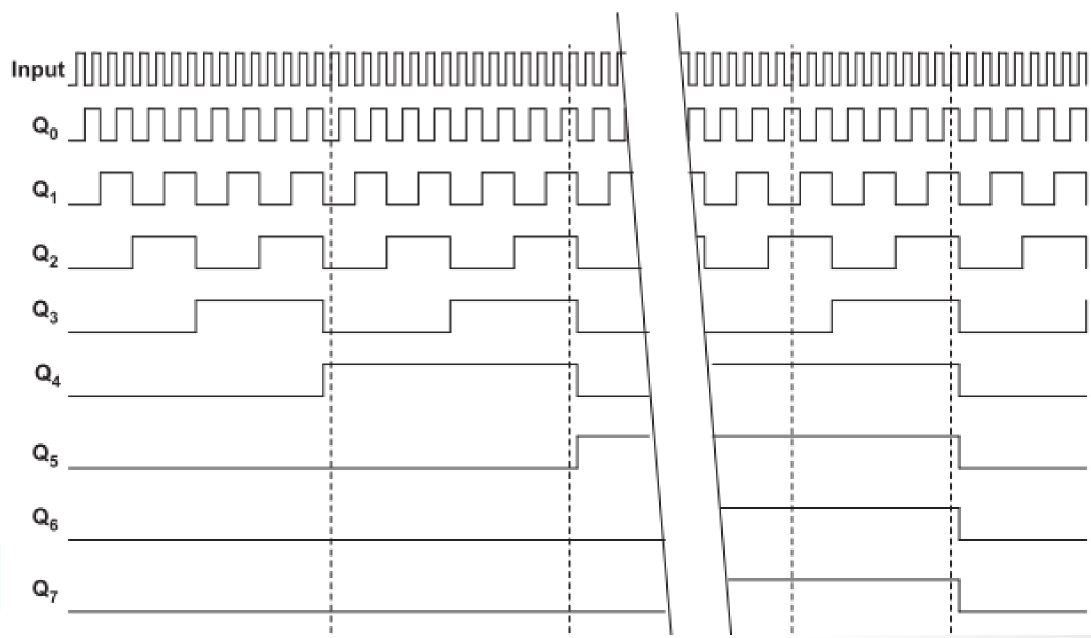
- Counters can be made which count up, count down, can be cleared back to zero, pre-loaded to a certain value, and which by the provision of an overflow output can be cascaded with other counters.

- A digital counter made of eight flip-flops



- Eight negative edge-triggered J–K bistables are interconnected, so that the Q-output of one drives the clock input of the next.
- With J and K both tied to Logic 1, the flip-flop toggles on every input negative edge.
- Q7 → MSB

output timing diagram →



The counter as a timer

- It is extremely useful for a microcontroller to be able to **count** – widgets passing on a conveyor belt, for example, coins in a slot machine, or people going through a door.
- It is, however, especially useful if it can **measure time**, and the counter allows us to do this.

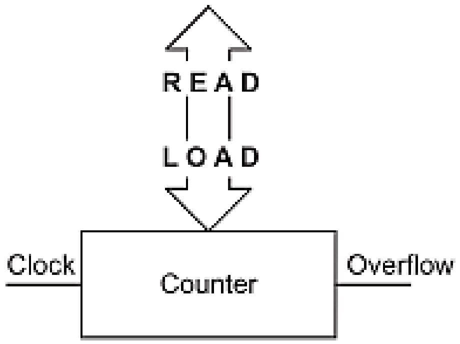
- Suppose the input signal of a counter is a stable **1 kHz clock frequency**.
  - Then the counter would **increment** exactly every **1 ms**.
  - After 16 clock cycles, exactly 16 ms would have elapsed, after 31 cycles 31 ms and so on.
- By starting the clock input at a moment of choice, it is therefore possible to measure elapsed time.
- The **resolution** of the measurement

is determined by the **period of the clock**.

- In this example the resolution is 1 ms

and we can't measure anything less than that,

or a fraction of it.



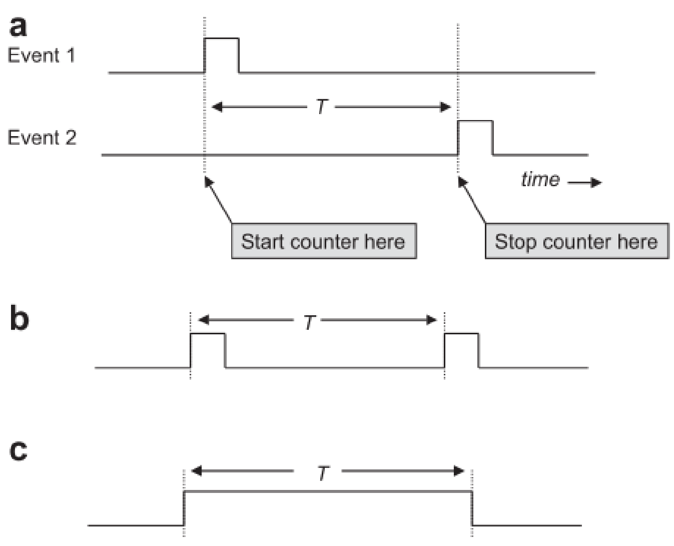
The challenge of time measurement

- The actual measurement **seems easy** – start the counter/timer running when the first event occurs and stop it at the moment of the second.
- In practice, this poses a number of **challenges**.
- For an accurate measurement, the **start** and **stop** of the counter/timer must be perfectly **synchronised** with the events.
- The best way of doing this is by using an **interrupt**.
- If we don't have an interrupt,

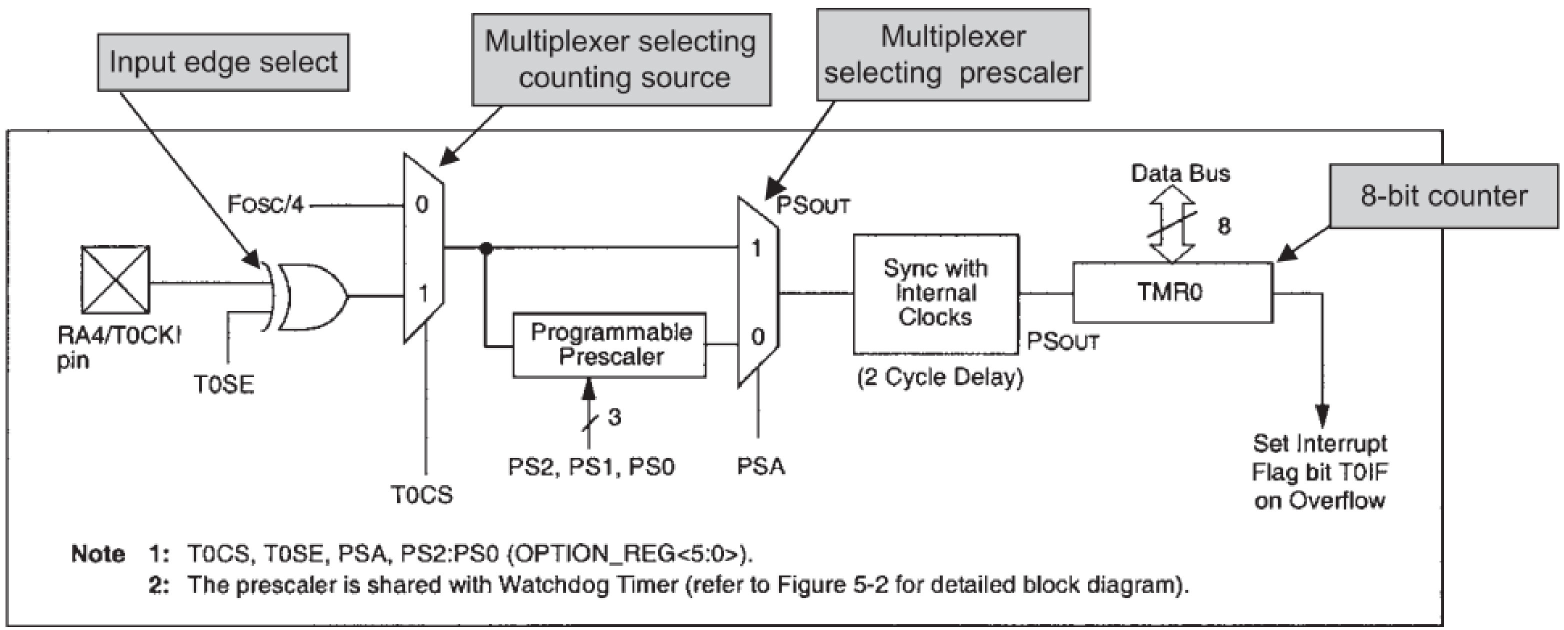
then we will have to

continuously scan the input

to detect when the event occurs .



16F84A Timer 0 module



16F84A OPTION register

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

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

The Watchdog Timer

- A big **danger** with any **computer-based** system is that the **software fails** in some way and that the system locks up or becomes unresponsive.
- In a desktop computer such a lock-up can be annoying and one would normally have to **reboot**.
- In an **embedded system** it can be disastrous, as there may be **no user to notice that there is something wrong** and maybe no user interface anyway.
- The **WDT** offers a fairly brutal '**solution**' to this problem.
- It is a **counter**, internal to the microcontroller, which is **continually counting up**.
- **If** it ever **overflows**, it **forces** the microcontroller into **Reset**.

Sleep Mode

- It is an important way of **saving power**.
- The microcontroller can be put into this mode by executing the instruction **SLEEP**.
- Once in Sleep mode, the microcontroller almost goes into **suspended** animation.
- The clock oscillator is switched off, the WDT is cleared, program execution is suspended, all ports retain their current settings, and the **PD** and **TO** bits in the Status register are **cleared** and **set** respectively.
- If enabled, the **WDT** continues **running**.
- Under these conditions, **power consumption** falls to a **negligible** amount.

Sleep Mode..

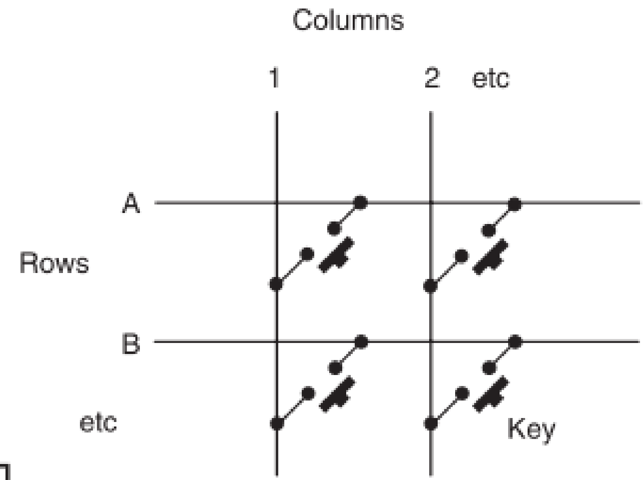
- The MCU **wakes** from Sleep in the following **situations**:

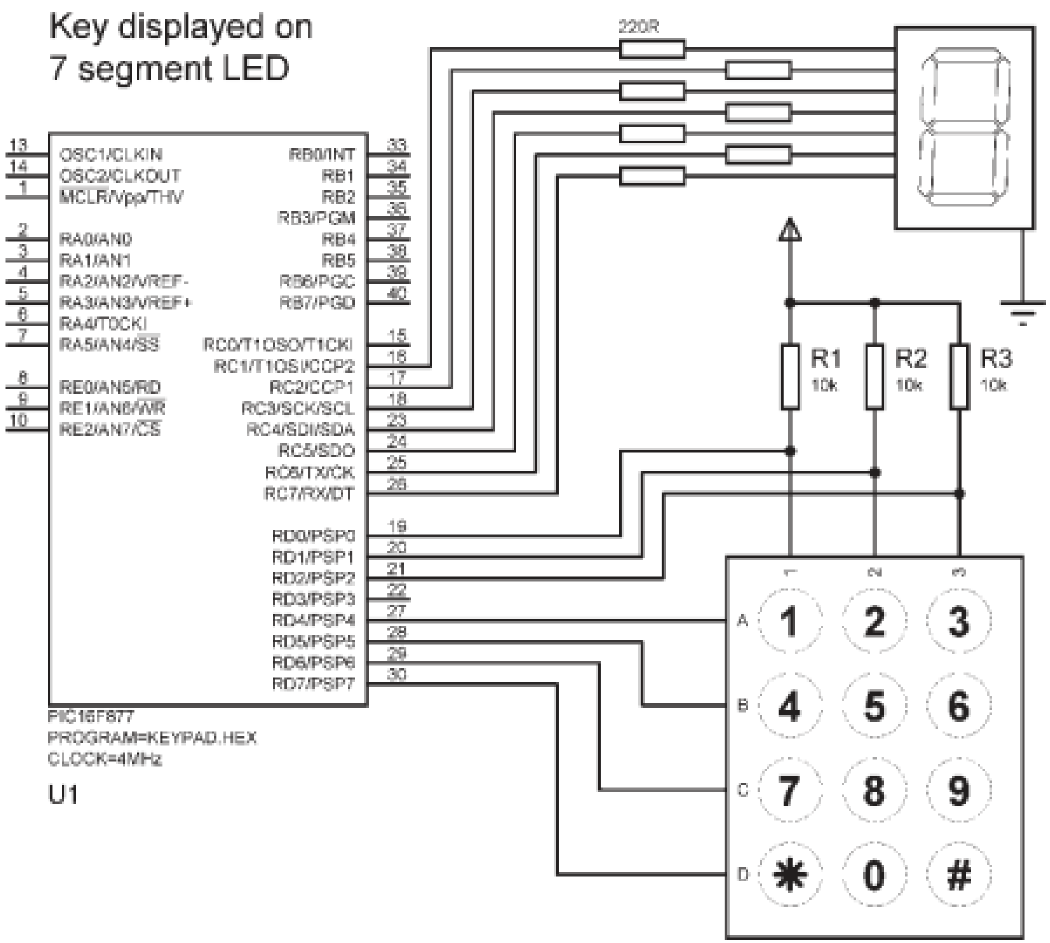
    - External reset through **MCLR** pin.
    - **WDT** wake-up.
    - Occurrence of **interrupt**.

- On wake-up, the **oscillator** circuit is **restarted**.
- The **Sleep mode** is extremely **powerful** for products that must be designed in a power conscious way.
    - Battery-based devices
    - WSN

SAMPLE PROJECT

Keypad/Display Example





When you press a key,
Display it on the 7-segment

- For more details, refer to:
  - Chapter 6, T. Wilmishurst, **Designing Embedded Systems with PIC Microcontrollers**, 2010.
- The lecture is available online at:
  - http://bu.edu.eg/staff/ahmad.elbanna-courses/12134
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg