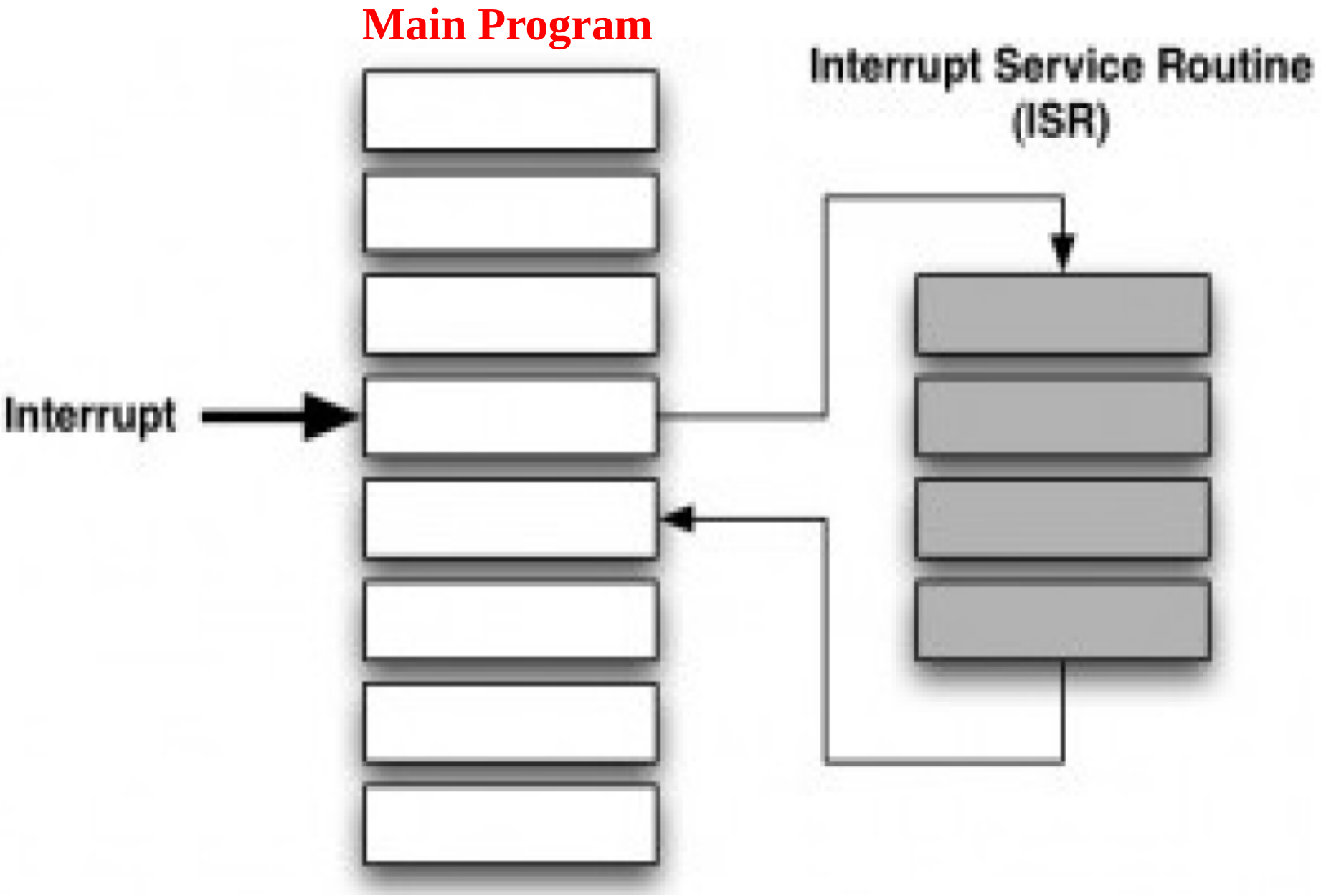
# Lec 4: Interrupt

# Interrupt

- Interrupt requests the microcontroller to stop the normal execution of program and execute a special code (interrupt service routine = ISR).
- **When interrupt occurs, the microcontroller does the following steps**:

1) Stops the regular program execution and saves its current location address (so that it can resume).
2) Jumps immediately to the ISR to execute it.
3) As soon as the ISR ends, the microcontroller restores its saved location address and resumes program execution from where it left off.


Main Program
Interrupt Service Routine
(ISR)
Interrupt

# Why Interrupt?

1. **Servicing time critical applications:** 
   - Real-time applications need **immediate attention** of the CPU.
   - For example, for safety reasons, it may be required to shut down a plant whenever there is power failure or fire.
   - In such applications, the CPU is required to stop whatever it is doing and service the interrupting device.
2. **Performing scheduled tasks:** 
   - There are many applications that require the CPU to perform scheduled tasks, such as updating the real-time clock.
   - These tasks are important and must be serviced at the **exact times**.

# Why Interrupt? *cont'd*

3. **Preventing CPU from being tied up (متكتف):**
   - In some applications, the CPU is required to perform continuous checks of I/O devices.
   - While performing these checks, the CPU cannot perform other duties.
   - By moving the checking into interrupt routines, the checking can be done in the background and the CPU is free to carry out other tasks.
   - For example, a multi- digit 7-segment display needs to be refreshed continuously.
   - If this task is done in the main program, then the CPU cannot do other tasks.
   - By moving the refreshing operation into the timer interrupt routine, the CPU is free to do other tasks.

# Common Sources of Interrupts

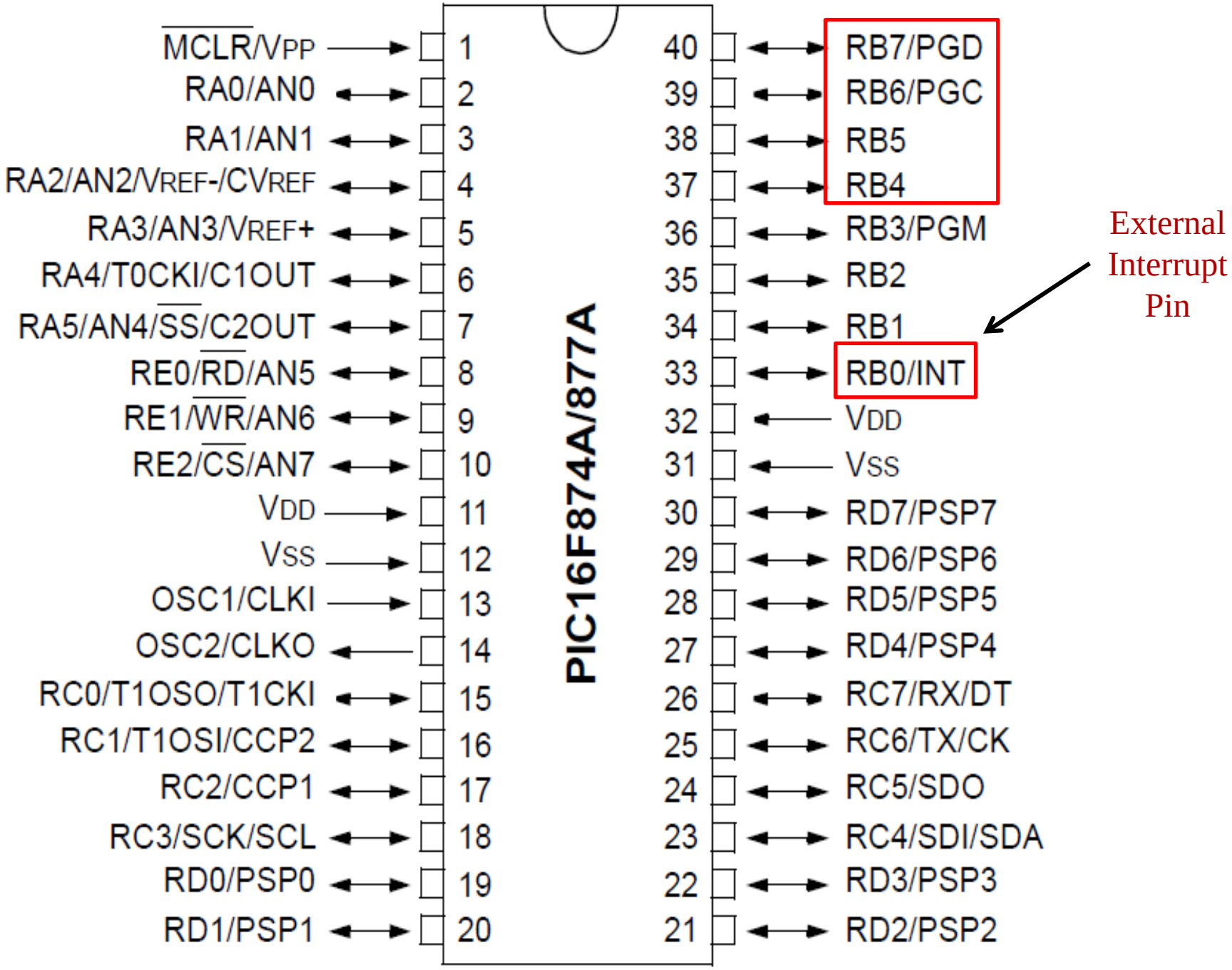
- PIC 16F877A has several interrupt sources, for example :

### 1. External interrupt

It provides a means for external hardware to generate interrupts through the RB0/INT pin (interrupt pin).

### 2. TIMER interrupts

They have related interrupts (for example TMR0 Interrupt). In most simple situation they can act like alarm clocks that can interrupt the microcontroller at predefined intervals of time.


PIC16F874A/877A
MCLR/VPP 1
RA0/AN0 2
RA1/AN1 3
RA2/AN2/VREF-/CVREF 4
RA3/AN3/VREF+ 5
RA4/T0CKI/C1OUT 6
RA5/AN4/SS/C2OUT 7
RE0/RD/AN5 8
RE1/WR/AN6 9
RE2/CS/AN7 10
VDD 11
VSS 12
OSC1/CLKI 13
OSC2/CLKO 14
RC0/T1OSO/T1CKI 15
RC1/T1OSI/CCP2 16
RC2/CCP1 17
RC3/SCK/SCL 18
RD0/PSP0 19
RD1/PSP1 20
40 RB7/PGD
39 RB6/PGC
38 RB5
37 RB4
36 RB3/PGM
35 RB2
34 RB1
33 RB0/INT
32 VDD
31 VSS
30 RD7/PSP7
29 RD6/PSP6
28 RD5/PSP5
27 RD4/PSP4
26 RC7/RX/DT
25 RC6/TX/CK
24 RC5/SDO
23 RC4/SDI/SDA
22 RD3/PSP3
21 RD2/PSP2
External Interrupt Pin

# Configuration of External Interrupt and TMR0 Interrupt

- **INTCON** (Interrupt Configuration) register and **OPTION_REG** are used to configure external Interrupt and TMR0 interrupt.

**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) | Features |
|---|---|---|---|---|---|---|---|---|
| GIE | PEIE | T0IE | INTE | RBIE | TMR0IF | INTF | RBIF | Bit name |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | |

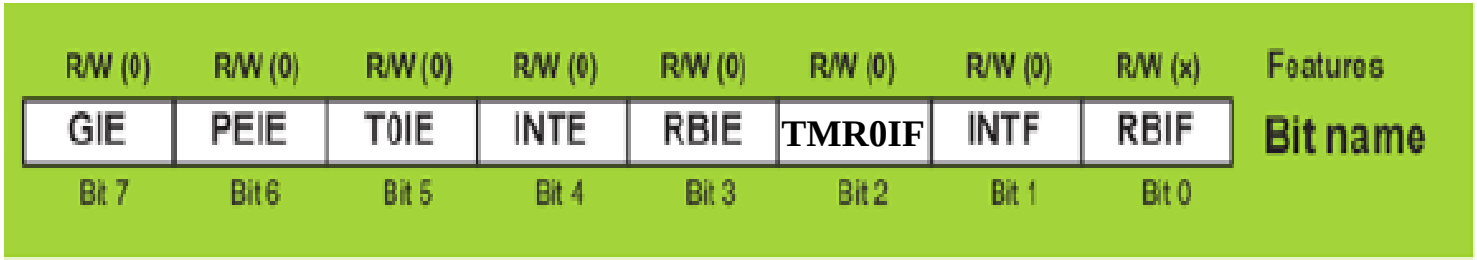
**OPTION_REG** Register

| R/W (1) | R/W (1) | R/W (1) | R/W (1) | R/W (1) | R/W (1) | R/W (1) | R/W (1) | Features |
|---|---|---|---|---|---|---|---|---|
| RBPU | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 | Bit name |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | |

1

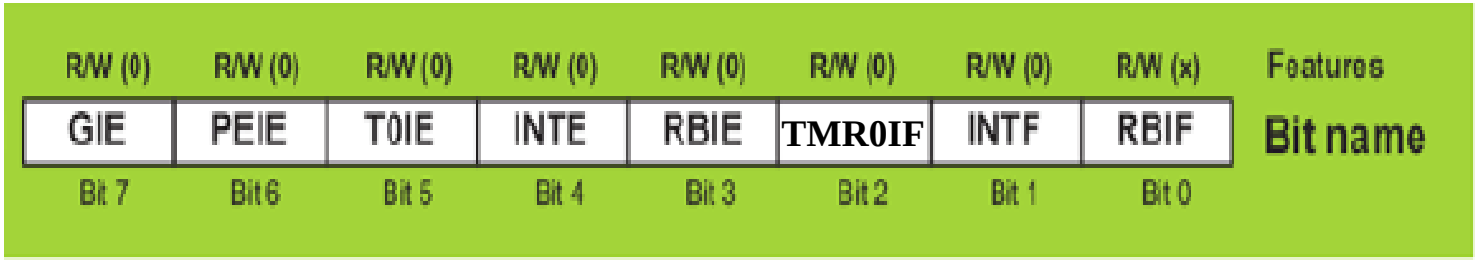
# INTCON
Register

**INTCON** Register




- **GIE** **(**Global Interrupt Enable**):**
  - **1 -** Enables all interrupts.
  - **0 -** Disables all interrupts.
- **PEIE** **(**Peripheral Interrupt Enable**):**
  - **1 -** Enables all peripheral interrupts.
  - **0 -** Disables all peripheral interrupts.
- **T0IE** **(**TMR0 Overflow Interrupt Enable**)** controls interrupt enabled by **TMR0 overflow**:
  - **1 -** Enables TMR0 interrupt.
  - **0 -** Disables TMR0 interrupt.

**INTCON**
Register




- **INTE** (RB0/INT External Interrupt Enable)

controls interrupt caused by changing the logic state of the RB0/INT pin (External Interrupt).

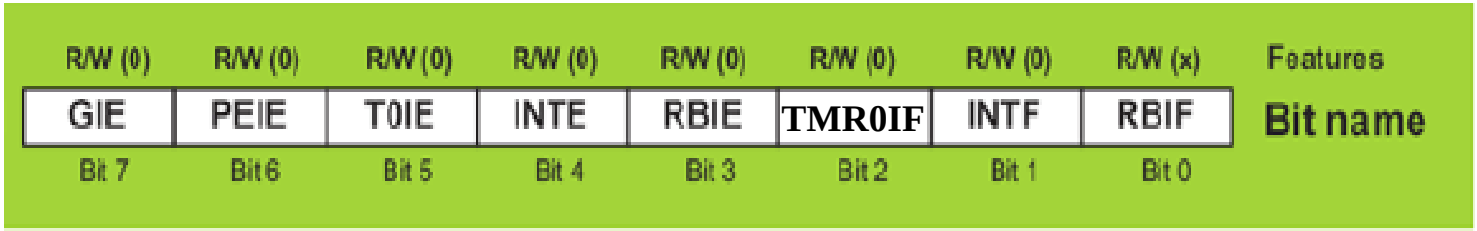
- **1 -** Enables the RB0/INT external interrupt.
- **0 -** Disables the RB0/INT external interrupt.

- **RBIE** (RB Port Change Interrupt Enable**):**

When **PORTB is configured as input port**, **RB4 – RB7** pins may cause an interrupt by changing their logic state (**no matter whether it is high-to-low transition or vice versa, the fact that something is changed only matters**). This bit determines whether an interrupt is to occur or not.

- **1 -** Enables the RB port change interrupt.
- **0 -** Disables the RB port change interrupt.

**INTCON** Register




- **TMR0IF** **(**TMR0 Overflow Interrupt Flag bit)

Indicates TMR0 overflow, when counting reaches to 255 and starts to count from 0 (255→0).

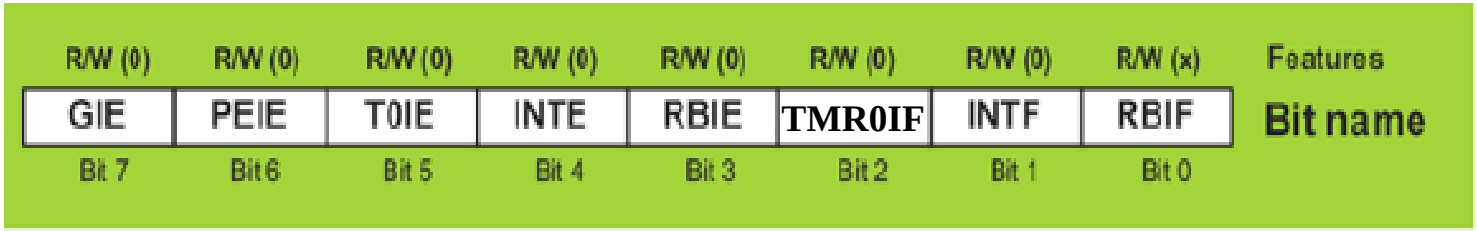
- **1 -** TMR0 register has overflowed (255→0). **This bit must be cleared in software.**
- **0 -** TMR0 register has not overflowed.

- **INTF** **(**RB0/INT External Interrupt Flag bit**):**

Indicates the change of the RB0/INT pin logic state.

- **1 -** The RB0/INT external interrupt has occurred. **This bit must be cleared in software.**
- **0 -** The RB0/INT external interrupt has not occurred.

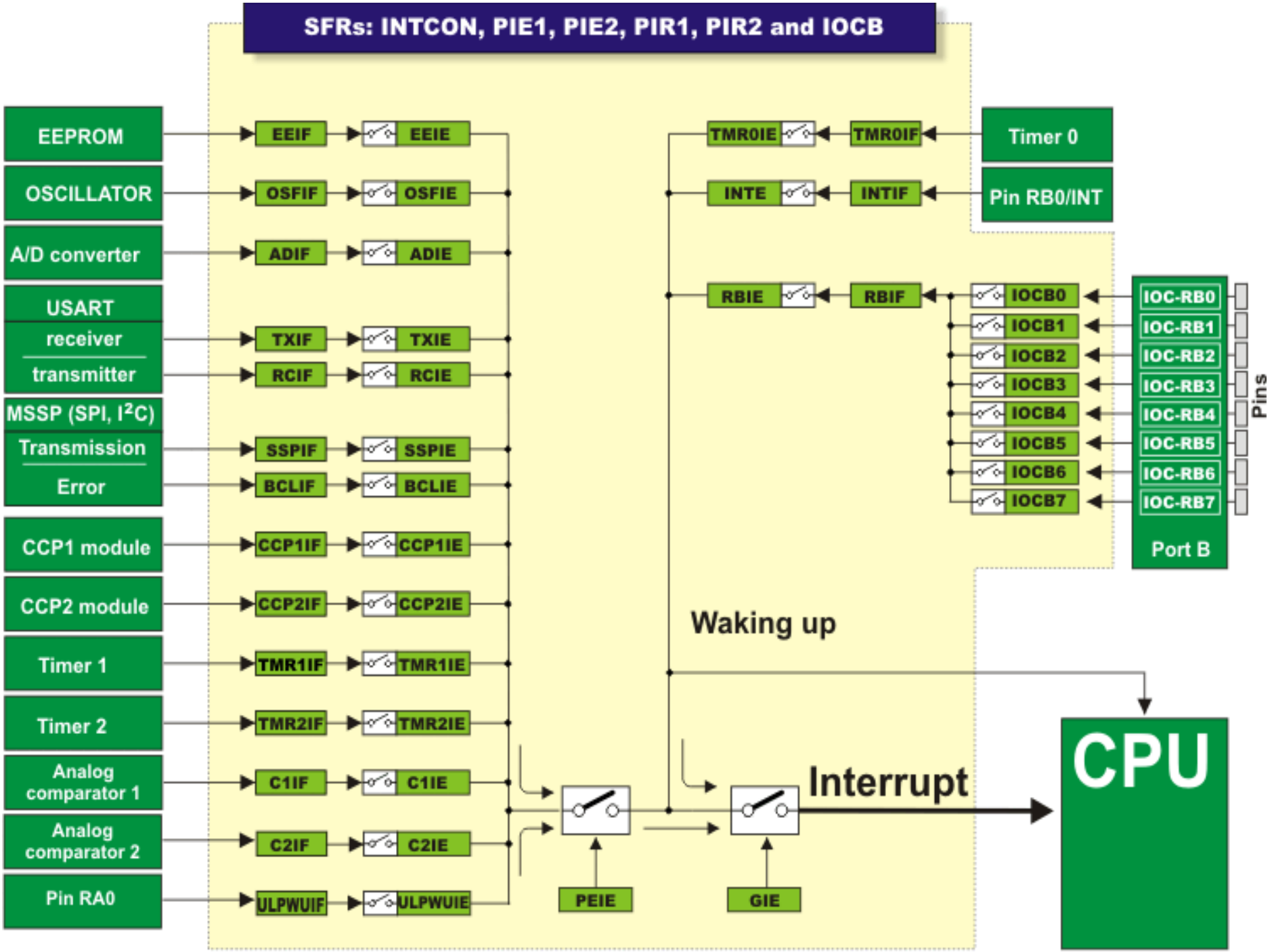
**INTCON** Register




- **RBIF** (RB Port Change Interrupt Flag bit)

Indicates **any change of logic state** of some PORTB input pins (RB4 – RB7 pins).

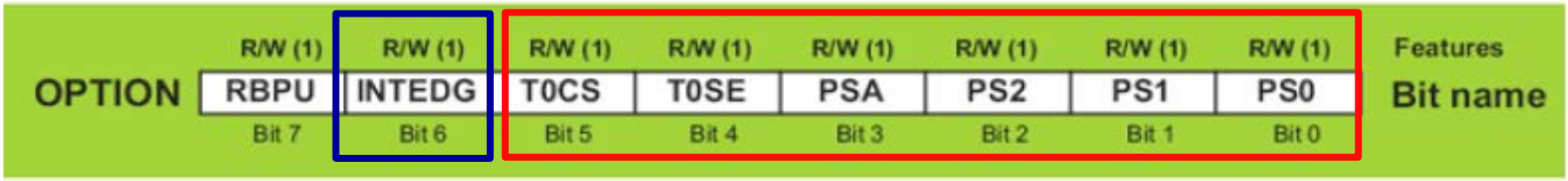
- **1 - At least one of** the RB4\RB5\RB6\RB7 pins has changed state. **This bit must be cleared in software.**
- **0** – None of the RB4 – RB7 pins has changed state.


SFRs: INTCON, PIE1, PIE2, PIR1, PIR2 and IOCB
EEPROM
OSCILLATOR
A/D converter
USART
receiver
transmitter
MSSP (SPI, I²C)
Transmission
Error
CCP1 module
CCP2 module
Timer 1
Timer 2
Analog comparator 1
Analog comparator 2
Pin RA0
EEIF
EEIE
OSFIF
OSFIE
ADIF
ADIE
TXIF
TXIE
RCIF
RCIE
SSPIF
SSPIE
BCLIF
BCLIE
CCP1IF
CCP1IE
CCP2IF
CCP2IE
TMR1IF
TMR1IE
TMR2IF
TMR2IE
C1IF
C1IE
C2IF
C2IE
ULPWUIF
ULPWUIE
TMR0IE
TMR0IF
Timer 0
INTE
INTIF
Pin RB0/INT
RBIE
RBIF
IOCB0
IOCB1
IOCB2
IOCB3
IOCB4
IOCB5
IOCB6
IOCB7
IOC-RB0
IOC-RB1
IOC-RB2
IOC-RB3
IOC-RB4
IOC-RB5
IOC-RB6
IOC-RB7
Pins
Port B
Waking up
PEIE
GIE
Interrupt
CPU

2

# OPTION_REG
Register

## OPTION_REG Register




- **PS0, PS1, PS2, PSA, T0SE, T0CS**
- These bits control the configuration of **TMR0/WD** (as discussed before in the pervious lecture).

- **INTEDG (Interrupt Edge Select bit)**: **INTCON.INTE = 1 is must**
  - **0 -** Interrupt on **falling** edge of the INT/RB0 pin (from high to low).
  - **1 –** Interrupt on **rising** edge of the INT/RB0 pin (from low to high).

# Interrupt Configuration

## 1 TMR0 Interrupt

```
OPTION_REG = 0x07;
OR
OPTION_REG = 0b00000111;
```

```
INTCON = 0xA0;
OR
INTCON = 0b10100000; (0xA0)
OR
INTCON.GIE = 1;
INTCON.T0IE = 1;
```

## 2 External Interrupt (RB0/INT pin)

```
OPTION_REG = 0;
OR
OPTION_REG.INTEDG = 0;
```

```
INTCON = 0x90;
OR
INTCON = 0x10010000; (0x90)
OR
INTCON.GIE = 1;
INTCON.INTE = 1;
```

# EX1: Without any Interrupt

- Write a microcontroller PIC16F877A C program to toggle all bits of PORTD every 10 sec, while at the same time the microcontroller is doing AND, OR, XOR between pin5, pin6, and pin7 of PORTB continuously as follows:

  portB.f1 = portB.f5 & portB.f6 & portB.f7

  portB.f2 = portB.f5 | portB.f6 | portB.f7

  portB.f3 = portB.f5 ^ portB.f6 ^ portB.f7

- PORTD is initialized by $55_H$

```
void main( ) {

      trisB = 0b11100000;
      trisD = 0;
      portD = 0x55;



    for(  ;  ;  ) {
        portD = ~portD;
        delay_ms(10000); // any time-consuming block

        portB.f1 = portB.f5 & portB.f6 & portB.f7;
        portB.f2 = portB.f5 | portB.f6 | portB.f7;
        portB.f3 = portB.f5 ^ portB.f6 ^ portB.f7;

     }
}
```

# EX2: TMR0 Interrupt

- Write a microcontroller PIC16F877A C program to use **TMR0 interrupt** to toggle all bits of PORTD every 10 sec, while at the same time the microcontroller is doing AND, OR, XOR between pin5, pin6, and pin7 of PORTB continuously as follows:

  portB.f1 = portB.f5 & portB.f6 & portB.f7
  portB.f2 = portB.f5 | portB.f6 | portB.f7
  portB.f3 = portB.f5 ^ portB.f6 ^ portB.f7

- PORTD is initialized by $55_H$
- Assume (1:256) Prescaler is assigned to TMR0 and f = 4 MHz.

## ➢ OPTION_REG Configuration:

OPTION_REG Register


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |

**OPTION_REG = 0x07;**

## ➢ Calculate the required no. of overflows to get time equal to 10 sec

- Assume The initial value of TMR0 = 0 (T0 =0)
- Overflow Time of TMR0 = (256-0) x prescale x 4 x clock_period

= 256 x 256 x 4 x $0.25*10^{-6}$ sec = 65536 $\mu Sec$

1 overflow $\longrightarrow$ 65536 $\mu$Sec

X overflows $\longrightarrow$ 10 Sec

$$X = \frac{10}{65536 * 10^{-6}} = 152.59 = \mathbf{153}$$

The required no. of overflows to get time equal to 10 sec = **153** overflows.

```
int overflows_no = 0;
void interrupt( ) {
     if(INTCON.TMR0IF == 1) {
        overflows_no++;     // Interrupt causes overflows_no to be incremented by 1
        INTCON.TMR0IF = 0;  //Clear the interrupt flag   or    INTCON.B2 = 0;  or   INTCON.F2 = 0
        TMR0 = 0;   //TMR0 returns to its initial value (T0)
     }
     if (overflows_no = = 153) {    // 10 sec delay
          PORTD = ~PORTD;       // Toggle PORTD
          overflows_no = 0;
      }
}
```

```
void main( ) {
   TRISB = 0b11100000;     // pin7, pin6, and pin5 are input
   TRISD = 0;     // PORTD O/P
   PORTD = 0x55;
   OPTION_REG = 0x07; // Prescaler (1:256) is assigned to the timer TMR0
   INTCON = 0xA0;     // Enable interrupt TMR0 and Global Interrupts
   TMR0 = 0;        // Timer TMR0 counts from 0 to 255
   while(1){
        portB.f1 = portB.f5 & portB.f6 & portB.f7;
        portB.f2 = portB.f5 | portB.f6 | portB.f7;
        portB.f3 = portB.f5 ^ portB.f6 ^ portB.f7;
    }
}
```

# EX3: RB0/INT Interrupt

- Write a microcontroller PIC16F877A C program to use external interrupt (RB0/INT pin) to toggle all bits of PORTD, while at the same time the microcontroller is doing AND, OR, XOR between pin5, pin6, and pin7 of PORTB continuously as follows:

  portB.f1 = portB.f5 & portB.f6 & portB.f7

  portB.f2 = portB.f5 | portB.f6 | portB.f7

  portB.f3 = portB.f5 ^ portB.f6 ^ portB.f7

- PORTD is initialized by $55_H$

```
void interrupt( ) {
  If ( INTCON.INTF = = 1 ) {
     INTCON.INTF = 0;     //Clear the interrupt flag   or    INTCON.B1 = 0;
     PORTD = ~PORTD;    // Toggle PORTD
    }
}
```

```
void main( ) {
    TRISB = 0b11100001;
    TRISD = 0;     // PORTD is O/P
    PORTD = 0x55;
    OPTION_REG = 0;         // Interrupt on falling edge of the RB0/INT pin
    INTCON = 0x90;          // Enable RB0/INT pin interrupt and Global Interrupts
    while(1)  {
            portB.f1 = portB.f5 & portB.f6 & portB.f7;
            portB.f2 = portB.f5  |  portB.f6  |  portB.f7;
            portB.f3 = portB.f5  ^  portB.f6  ^  portB.f7;
    }
}
```