Chapter 10

Interrupts

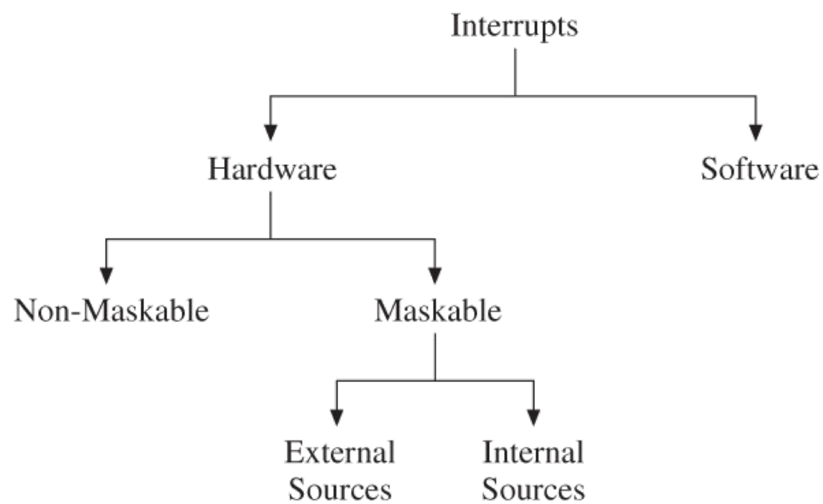
Updated: 4/19/19

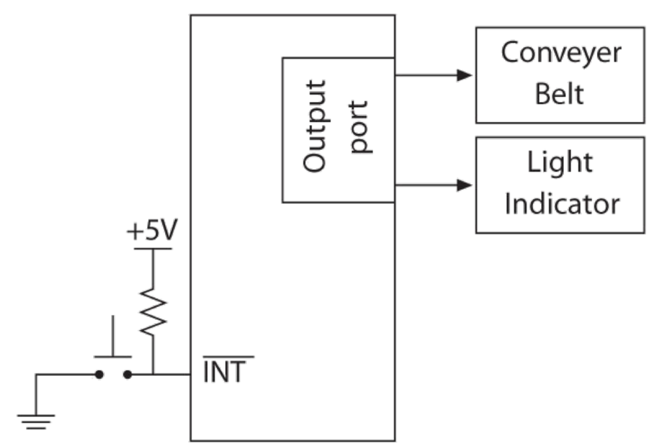
Basic Concepts in Interrupts

- An interrupt is a communication process set up in a microprocessor or microcontroller in which:
  - An internal or external device requests the MPU to stop the processing
    - The MPU acknowledges the request
    - Attends to the request
    - Goes back to processing where it was interrupted

Types of Interrupts

- Hardware interrupts
  - Maskable: can be masked or disabled
    - Two groups: external and internal interrupts
      - External through designated I/O pins
      - Internal by Timers, A/D, etc.
  - Non-maskable: cannot be disabled
- Software interrupts: generally used when the situation requires stop processing and start all over
  - Examples: divide by zero or stack overflow
  - Generally, microcontrollers do not include software interrupts





MPU Response to Interrupts (1 of 2)

- When the interrupt process is enabled, the MPU, during execution, checks the interrupt request flag just before the end of each instruction.
- If the interrupt request is present, the MPU:
  - Completes the execution of the instruction
  - Resets the interrupt flag
  - Saves the address of the program counter on the stack
    - Some interrupt processes also save contents of MPU registers on the stack.
  - Stops the execution

MPU Response to Interrupts (2 of 2)

- To restart the execution, the MPU needs to be redirected to the memory location where the interrupt request can be met.
  - Accomplished by interrupt vectors
- The set of instructions written to meet the request (or to accomplish the task) is called an interrupt service routine (ISR).
- Once the request is accomplished, the MPU should find its way back to the instruction, next memory location where it was interrupted.
  - Accomplished by a specific return instruction

```
Main code
....
Setup interrupt vectors
....
HERE: GOTO HERE


ORG 0x100
INT1_ISR: ISR code
....
....
RETFIE

END
```

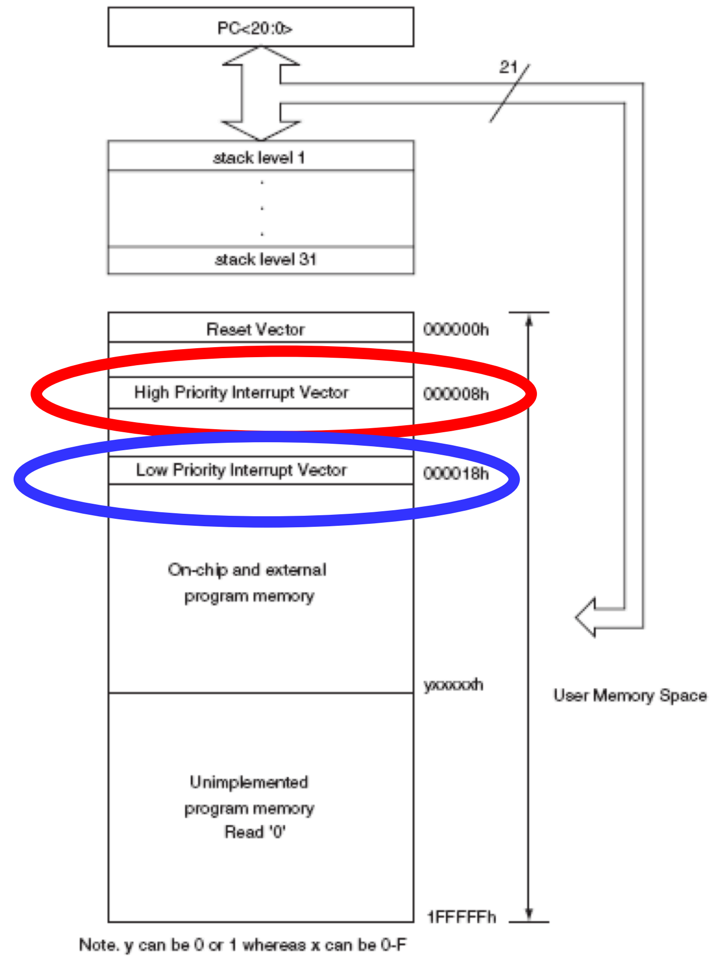
Interrupt Service Routine (ISR)

- RETFIE instruction format
  - RETFIE [s] ;Return from interrupt: s = 0 or 1
    - If s =1, the MPU also retrieves the contents of W, BSR, and STATUS register (previously saved) before enabling the global interrupt bit.
  - Format: RETFIE FAST
    - The same as RETFIE 1 except the formats are different

```
Main code
....
Setup interrupt vectors
....
HERE: GOTO HERE


ORG 0x100
INT1_ISR: ISR code
....
....
RETFIE

END
```

Interrupt Vectors

- Direct the MPU to the location where the interrupt request is accomplished.
- They are:
  - Defined memory location where a specific memory location/s is assigned to the interrupt request
  - Defined vector location where specific memory locations assigned to store the vector addresses of the ISRs
  - Specified by external hardware: The interrupt vector address (or a part of it) is provided through external hardware using an interrupt acknowledge signal.

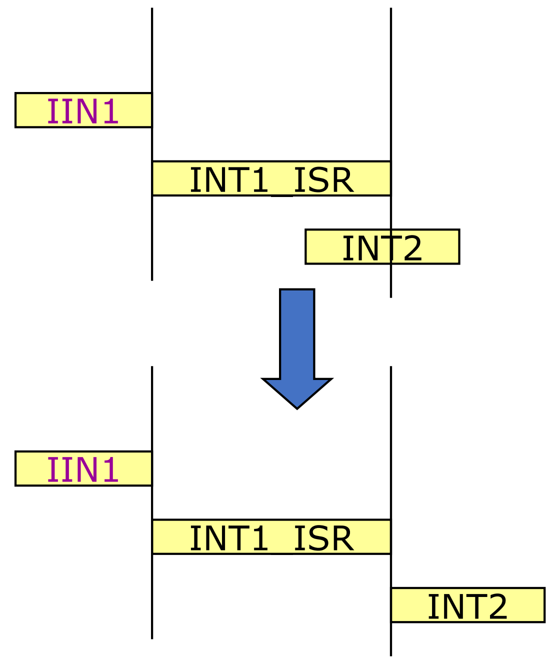


Interrupt Service Routine (ISR)

- A group of instructions that accomplishes the task requested by the interrupting source
- Similar to a subroutine except that the ISR must be terminated in a Return instruction specially designed for interrupts
  - The Return instruction, when executed, finds the return address on the stack and redirects the program execution where the program was interrupted.
  - Some Return instructions are designed to retrieve the contents of MPU registers if saved as a part of the interrupts.
  - →RETFIE FAST (1/0)

Interrupt Priorities

- Rationale for priorities
  - Multiple interrupt sources exist in a system, and more than one interrupt requests can arrive simultaneously.
    - Example: A/D and Timer0
  - When one request is being served (meaning when the MPU is executing an ISR), another request can arrive.
  - → the interrupt requests must be prioritized.
  - Most MCUs (and MPUs) include an interrupt priority scheme. Some are based on hardware and some use software.



INT1 has higher priority than INT2

Reset as a Special Purpose Interrupt

- Reset is an external signal that enables the processor to begin execution or interrupts the processor if the processor is executing instructions.
- There are at least two types of resets in microcontroller-based systems.
  - Power-on reset and manual reset
- When the reset signal is activated, it establishes or reestablishes the initial conditions of the processor and directs the processor to a specific starting memory location.

PIC18 Interrupts

- PIC18 Microcontroller family
  - Has multiple sources that can send interrupt requests
    - Does not have any non-maskable or software interrupts; all interrupts are maskable (can be disabled)
  - Has a priority scheme divided into two groups
    - High priority and low priority
  - Uses many Special Function Registers (SFRs) to implement the interrupt process

PIC18 Interrupt Sources

- Divided into two groups
  - External sources and internal peripheral sources on the MCU chip
  - External sources
    - Three pins of PORTB -RB0/INTO, RB1/INT1,and RB2/INT2 (edge driven)
    - Change in logic levels of pins RB4-RB7 of PORTB can be recognized as interrupts
  - Internal sources
    - Use SFRs to setup the interrupt process….

PIC18 Interrupt Sources

- Internal peripheral sources
  - Examples: Timers, A/D Converter, Serial I/O, and Low-Voltage Detection Module
- SFRs
  - Used to setup the interrupt process:
    - RCON Register Control (global priority)
    - INTCON Interrupt Control
    - INTCON2 Interrupt Control2
    - INTCON3 Interrupt Control3
    - PIR1 and PIR2 Peripheral Interrupt Register1 & 2
    - PIE1 and PIE2 Peripheral Interrupt Enable 1 & 2
    - IPR1 and IPR2 Interrupt Priority Register 1 & 2

external interrupt sources (INTCON, INTCON2, INTCON3)

Handle Internal peripherals (PIR1 and PIR2, PIE1 and PIE2, IPR1 and IPR2)

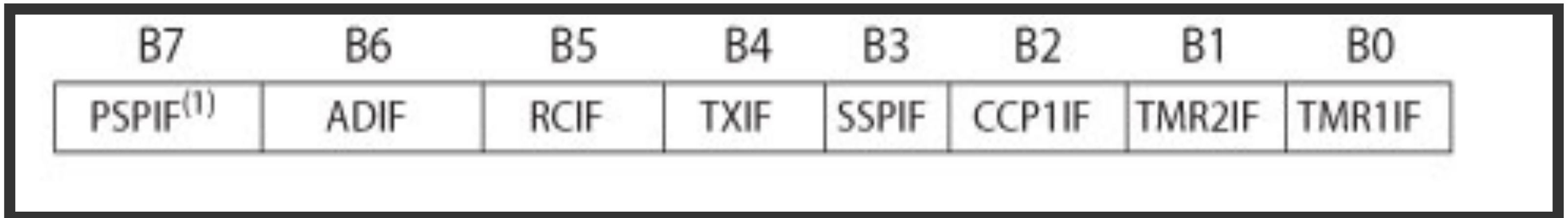
Click here: Summery of Interrupt Registers

PIC18 Interrupt Sources

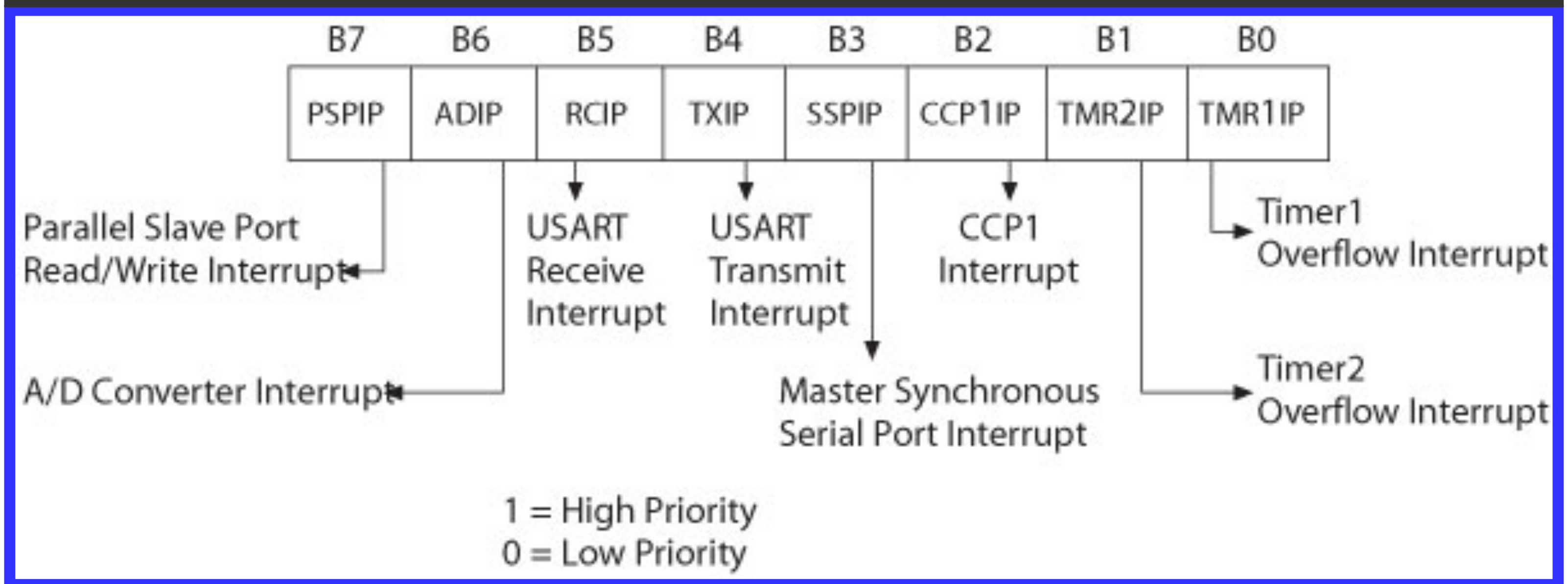
- To recognize the occurrence of an interrupt request, the MPU needs to check the following three bits:
  - The flag bit to indicate that an interrupt request is present
  - The enable bit to redirect the program execution to the interrupt vector address
  - The priority bit (if set) to select priority

PIC18 Interrupt Sources

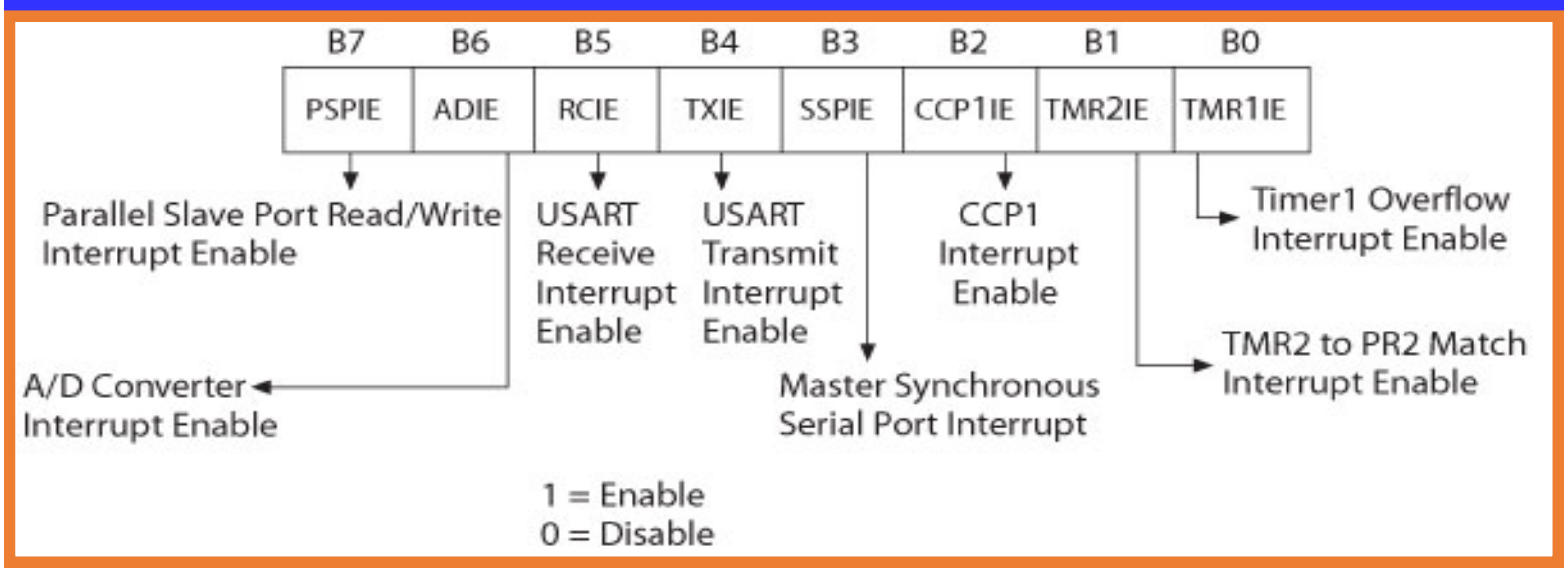
- In PIC interrupt are controlled by three bits in three different registers.
  - The IE bit is the interrupt enable bit used to enable the interrupt.
  - The IP bit is the interrupt priority bit which selects the priority (high or low).
  - The IF bit is the interrupt flag that indicates the interrupt has occurs. This bit **must be cleared** in the interrupt service function or no future interrupt will ever take effect.



PIR (flag)



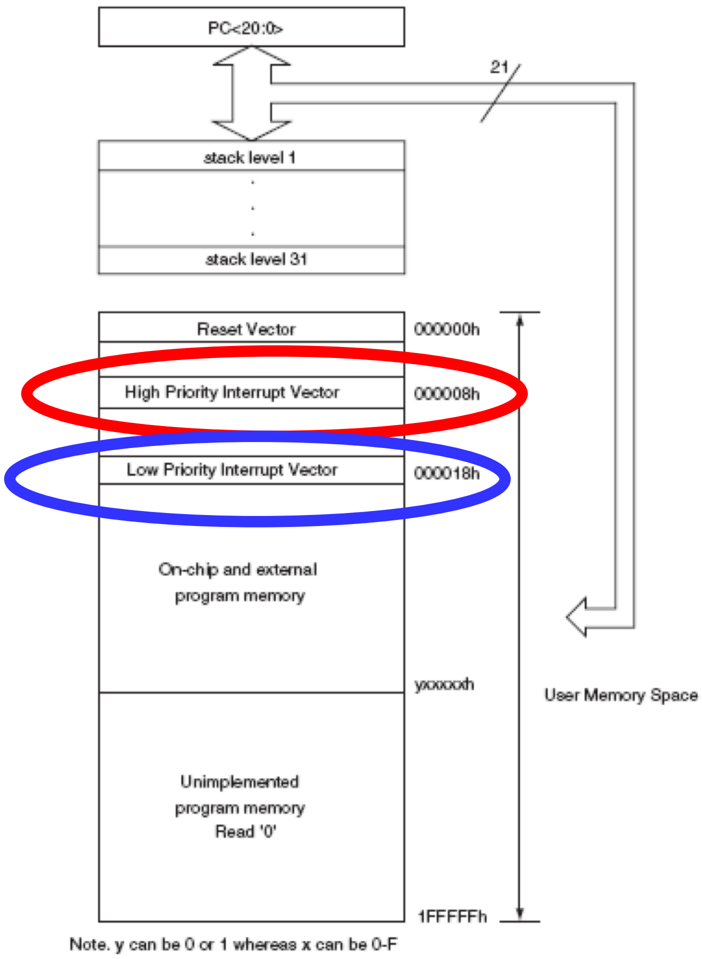
IPR (priority)
1=High Prio
0=Low Prio



PIE (peripheral)
1=Enable
0=Disable

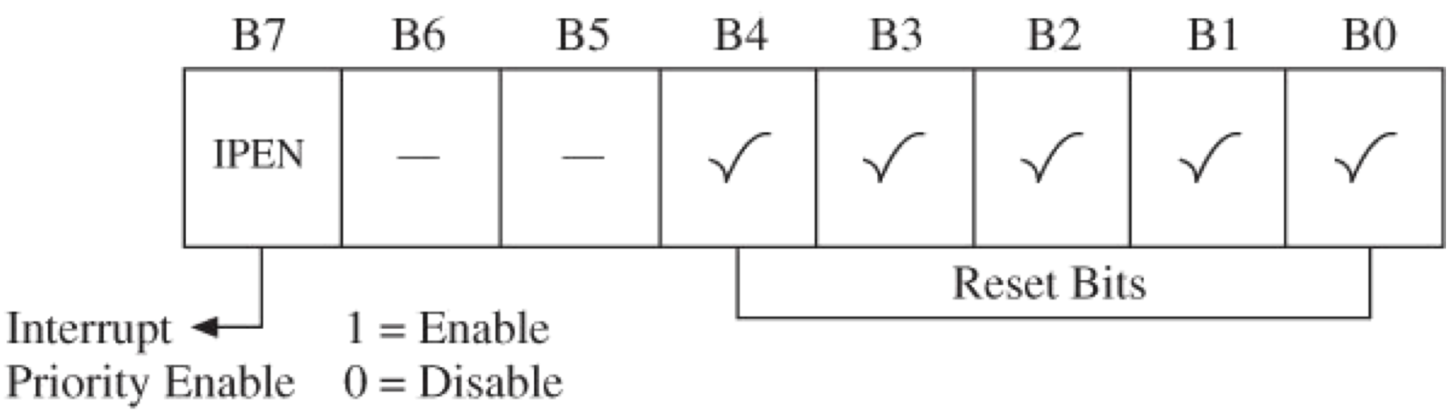
Interrupt Priorities and RCON Register (1 of 2)

- Any interrupt can be set up as high-priority or low-priority.
  - All high-priority interrupts are directed to the interrupt vector location 000008H.
  - All low-priority interrupts are directed to the interrupt vector location 000018H.
  - A high-priority interrupt can interrupt a low-priority interrupt in progress.



Interrupt Priorities and RCON Register (2 of 2)

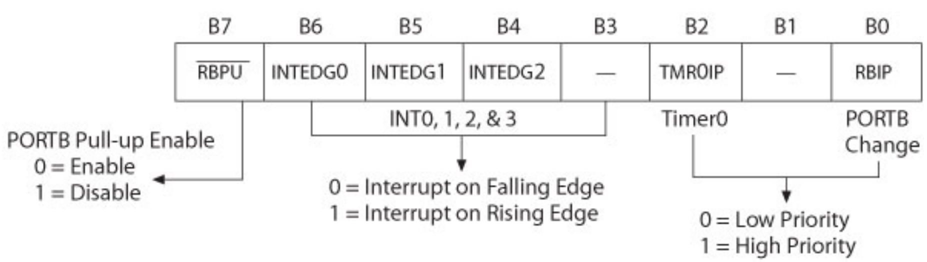
- The interrupt priority feature is enabled by Bit7 (IPEN) in RCON register.

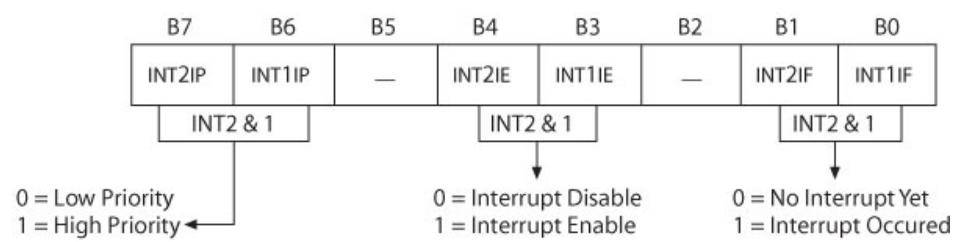


External Interrupts and INTCON Registers (1 of 3)

- Three registers with interrupt bit specifications primarily for external interrupt sources. INTCON (3)

B7	B6	B5	B4	B3	B2	B1	B0
GIE/GEIH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF





Example

- Write an instruction to setup INT1 as the high priority interrupt. (INT1 → RB1)

```
        ORG     0x00
        GOTO    MAIN

        ORG     0x0008
        GOTO    INT1_ISR

MAIN:   BSF     RCON,    IPEN
        BSF     INTCON,  GIEH
        BCF     INTCON2, INTEDG1
        BSF     INTCON3, INT1IP
        BSF     INTCON3, INT1IE
        MOVLW   D'10'
        MOVWF   REG1,0
HERE:   GOTO    HERE

        ORG     0x100
INT1_ISR
        DECF    REG1,1,0
        BNZ     GOBACK
        MOVLW   D'10'
        MOVWF   REG1,0
GOBACK: RETFIE  FAST
        END
```

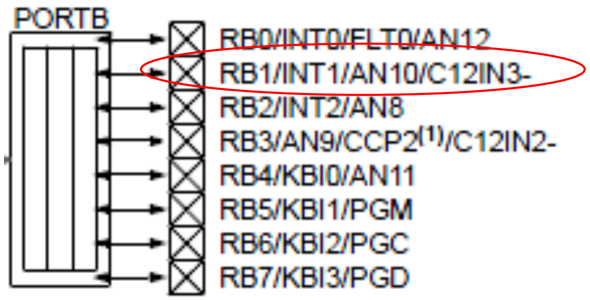
High Priority Interrupt Vector:

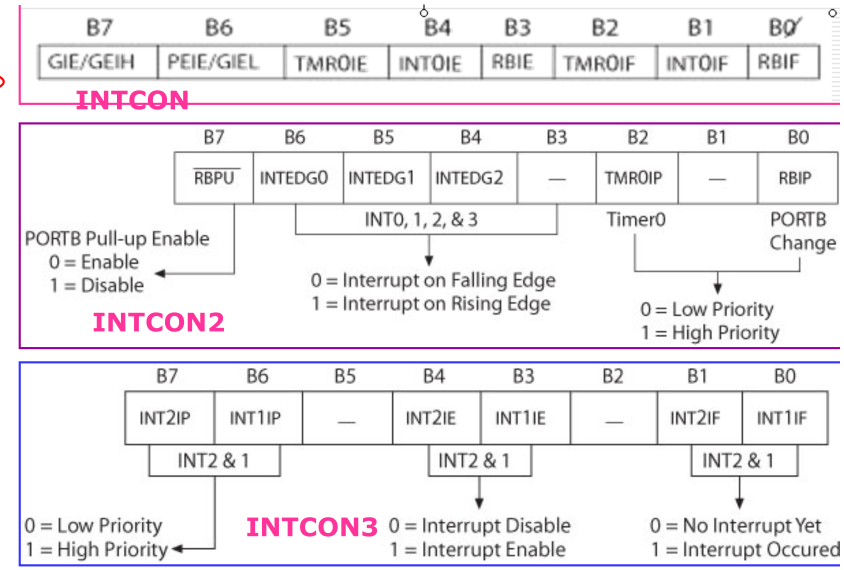
5	0008	EF80	GOTO 0x100
6	000A	F000	NOP
7	000C	8ED0	BSF 0xfd0, 0x7, ACCESS
8	000E	8EF2	BSF 0xff2, 0x7, ACCESS
9	0010	9AF1	BCF 0xff1, 0x5, ACCESS
10	0012	8CF0	BSF 0xff0, 0x6, ACCESS
11	0014	86F0	BSF 0xff0, 0x3, ACCESS
12	0016	0E0A	MOVLW 0xa
13	0018	6E01	MOVWF 0x1, ACCESS
14	001A	FFFF	NOP

ISR:

129	0100	0601	DECF 0x1, F, ACCESS
130	0102	E102	BNZ 0x108
131	0104	0E0A	MOVLW 0xa
132	0106	6E01	MOVWF 0x1, ACCESS
133	0108	0011	RETFIE 0x1
134	010A	FFFF	NOP

Example





☐Software Setting

```
        ORG     0x00
        GOTO    MAIN

        ORG     0x0008
        GOTO    INT1_ISR

MAIN:   BSF     RCON,    IPEN
        BSF     INTCON,  GIEH
        BCF     INTCON2, INTEDG1
        BSF     INTCON3, INT1IP
        BSF     INTCON3, INT1IE
        MOVLW   D'10'
        MOVWF   REG1,0
HERE:   GOTO    HERE

        ORG     0x100
INT1_ISR
        DECF    REG1,1,0
        BNZ     GOBACK
        MOVLW   D'10'
        MOVWF   REG1,0
GOBACK: RETFIE  FAST
        END
```

If ZERO no priority (early versions)

If enabled → high priority → gates all H/L interrupts to the CPU (remember: GIEL enables all low-priorities Interrupts to the CPU

If 1 → rising edge (RB1)

If 1 → High Priority

If 1 → INT1 enabled

When INT1 goes from *high to low* value of REG1 decrements!

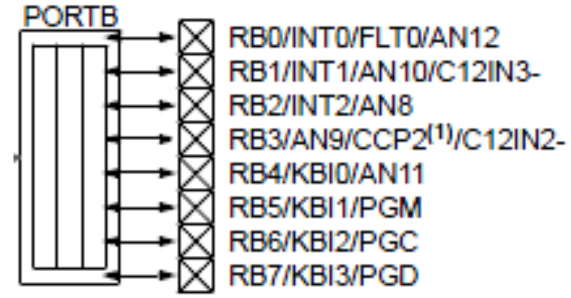
NOTE: INTCON3: FLAG bit is read only!

Convert to C code!

Interrupt Setting (INT0 high priority only)

Name	Priority Bit	Local Enable Bit	Local Flag Bit
INT0 external interrupt	*	INTCON,INT0IE	INTCON,INT0IF
INT1 external interrupt	INTCON3,INT1IP	INTCON3,INT1IE	INTCON3,INT1IF
INT2 external interrupt	INTCON3,INT2IP	INTCON3,INT2IE	INTCON3,INT2IF
RB port change interrupt	INTCON2,RBIP	INTCON,RBIE	INTCON,RBIF
TMR0 overflow interrupt	INTCON2,TMR0IP	INTCON,TMR0IE	INTCON,TMR0IF
TMR1 overflow interrupt	IPR1,TMR1IP	PIE1,TMR1IE	PIR1,TMR1IF
TMR3 overflow interrupt	IPR2,TMR3IP	PIE2,TMR3IE	PIR2,TMR3IF
TMR2 to match PR2 int.	IPR1,TMR2IP	PIE1,TMR2IE	PIR1,TMR2IF
CCP1 interrupt	IPR1,CCP1IP	PIE1,CCP1IE	PIR1,CCP1IF
CCP2 interrupt	IPR2,CCP2IP	PIE2,CCP2IE	PIR2,CCP2IF
A/D converter interrupt	IPR1,ADIP	PIE1,ADIE	PIR1,ADIF
USART receive interrupt	IPR1,RCIP	PIE1,RCIE	PIR1,RCIF
USART transmit interrupt	IPR1,TXIP	PIE1,TXIE	PIR1,TXIF
Sync. serial port int.	IPR1,SSPIP	PIE1,SSPIE	PIR1,SSPIF
Parallel slave port int.	IPR1,PSPIP	PIE1,PSPIE	PIR1,PSPIF
Low-voltage detect int.	IPR2,LVDIP	PIE2,LVDIE	PIR2,LVDIF
Bus-collision interrupt	IPR2,BCLIP	PIE2,BCLIE	PIR2,BCLIF

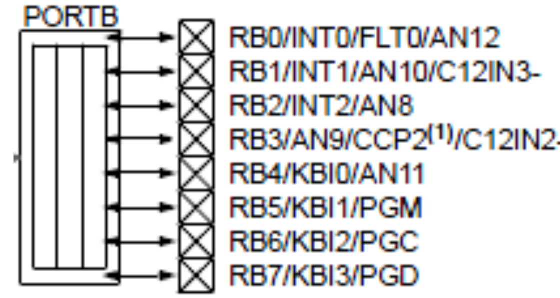
C Code Example

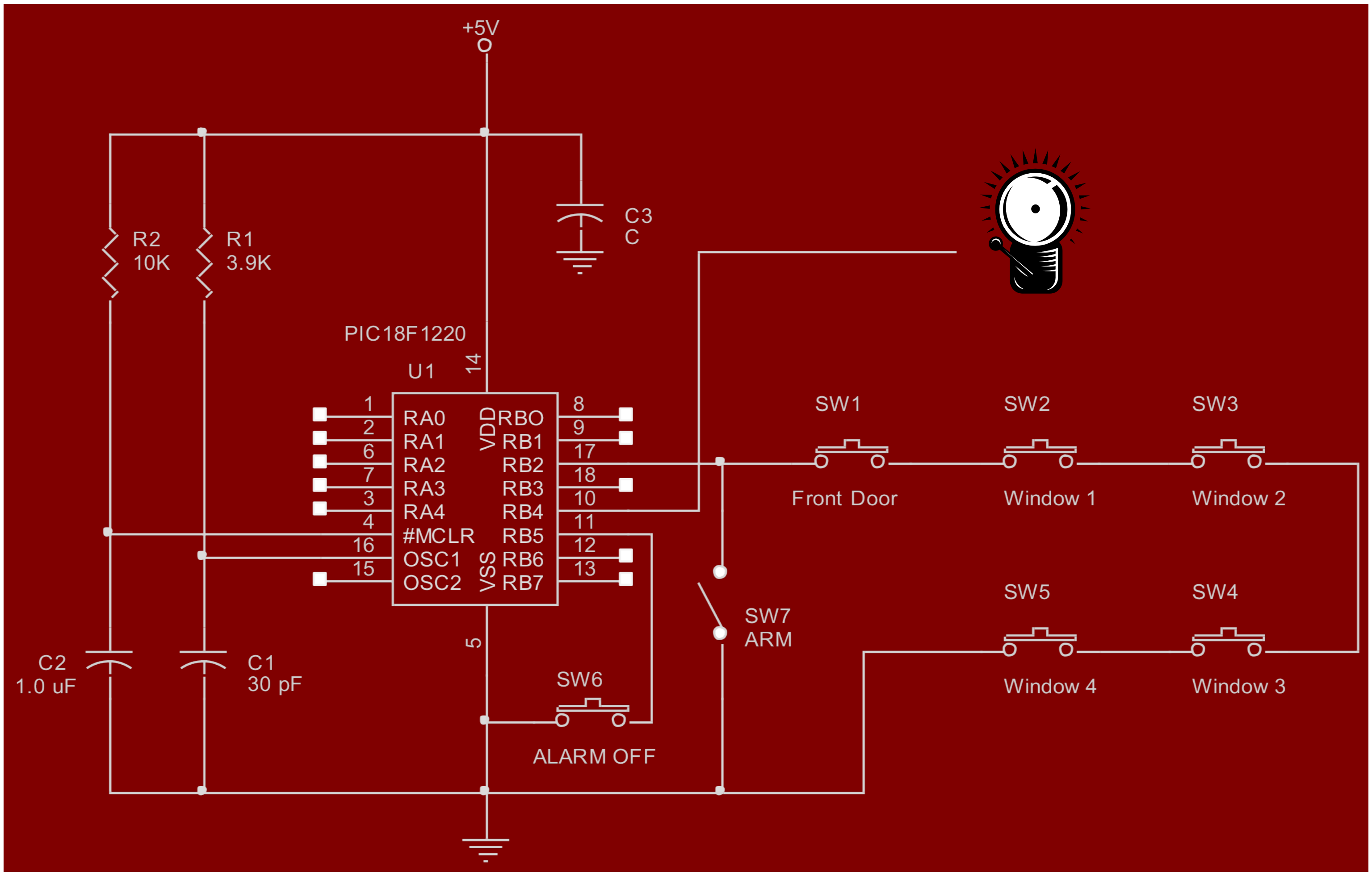


When a pulse is generated on INT0 the high priority interrupt is generated!

```
ADCON1 = 0x0F;                   // make ports pins digital

TRISB = 1;                       // make RB0 input

RCONbits.IPEN = 1;               // IPEN = 1

INTCON2bits.INTEDG0 = 0;         // make INT0 negative edge triggered
INTCONbits.INT0IE = 1;           // enable INT0
INTCONbits.GIEH = 1;             // enable high priority interrupts

// INT0 is now armed and active
```

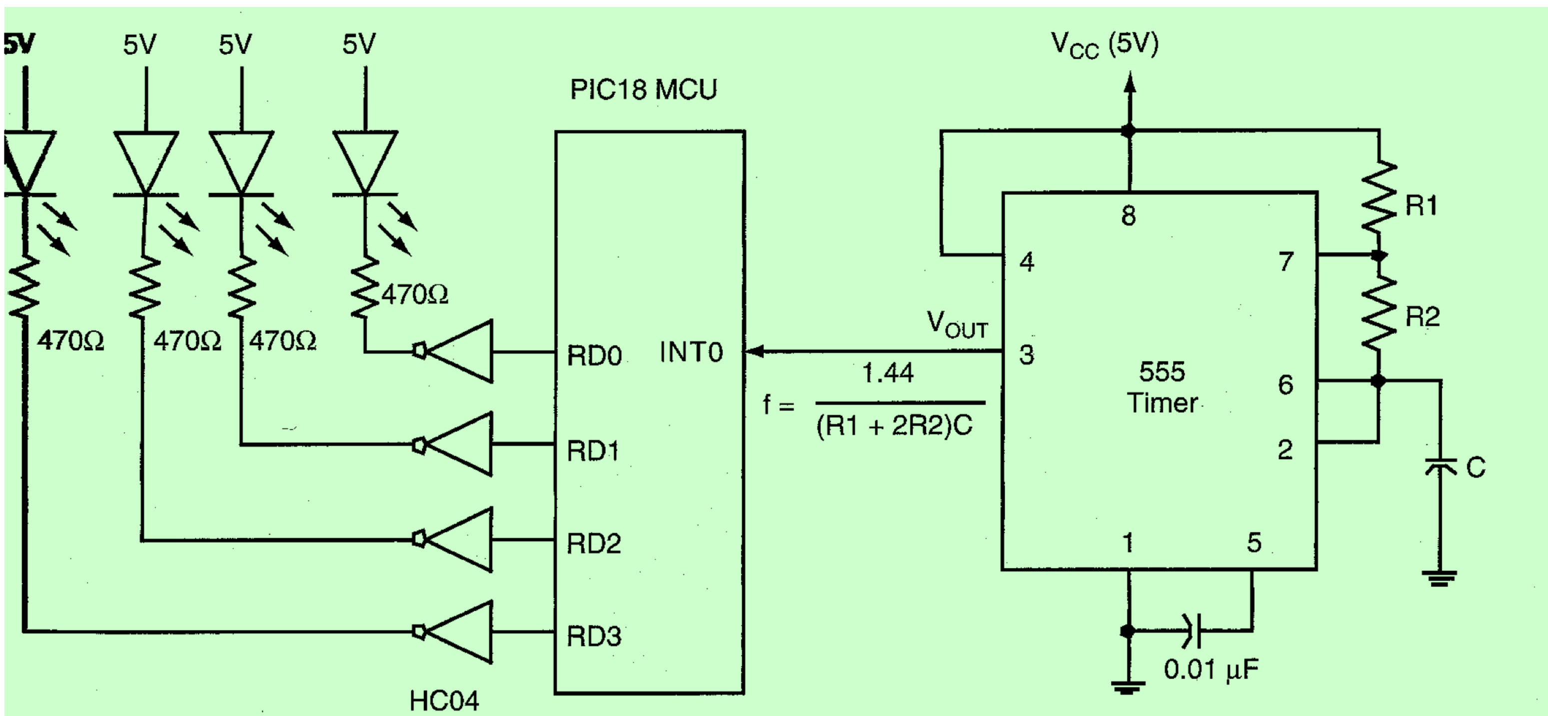
C Code Example – Burglar Alarm Circuit





Another practical Example

Suppose you are given a circuit as shown below. Write a main program and an INT0 interrupt service routine in assembly language. The main program initializes a counter to 0, enables the INTO interrupt, and then stays in a while-loop to wait forever. The INT0 interrupts service routine simply increments the counter by 1 and outputs it to the LEDs. Whenever is incremented to 15, the service routine resets it to 0. Choose appropriate component that the PIC1 8 receives an INTO interrupt roughly every second.



Handling Multiple Interrupt Sources

- In PIC18 MCU, all interrupt requests are directed to one of two memory locations:
  - 000008H (high-priority) or 000018 (low-priority)
- When multiple requests are directed to these locations, the interrupt source must be identified by checking the interrupt flag through software instructions.

Example

IPR1							
PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP

PIE1							
PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

```
BSF     INTCON, GIEL      ;Enable global low-priority - INTCON ,6>
BCF     IPR1, TMR1IP      ;Set Timer1 as low-priority
BSF     PIE1, TMR1IE      ;Enable Timer1 overflow interrupt
BCF     IPR1, TMR2IP      ;Set Timer2 as low-priority
BSF     PIE1, TMR2IE      ;Enable Timer2 match interrupt


BCF     PIR1, TMR1IF      ;Clear TMR1 flag
CALL    TMR1L             ;Call service subroutine

BCF     PIR1,TMR2IF       ;Clear TMR2 flag
CALL    TMR2              ;Call service subroutine
```

Name	Priority Bit	Local Enable Bit	Local Flag Bit
RB port change interrupt	INTCON2,RBIP	INTCON,RBIE	INTCON,RBIF
TMR0 overflow interrupt	INTCON2,TMR0IP	INTCON,TMR0IE	INTCON,TMR0IF
TMR1 overflow interrupt	IPR1,TMR1IP	PIE1,TMR1IE	PIR1,TMR1IF
TMR3 overflow interrupt	IPR2,TMR3IP	PIE2,TMR3IE	PIR2,TMR3IF
TMR2 to match PR2 int.	IPR1,TMR2IP	PIE1,TMR2IE	PIR1,TMR2IF

Example of using multiple interrupts

INT1=High Priority / TMR1 and TMR2 Low Priority

Assign the High Priority Interrupt Vector
Assign the Low Priority Interrupt Vector
Setup the interrupt registers for external interrupt
Setup the interrupt registers for internal interrupts

Example of using multiple interrupts

INT1=High Priority / TMR1 and TMR2 Low Priority

```
        ORG     0x00
        GOTO    MAIN

        ORG 0x0008
INTCK:
        GOTO    INT1_ISR
```

High priority

```
        ORG 0x00018
TIMERCK:
        BTFSC   PIR1,TMR1IF
        GOTO    TMR1_ISR
        BTFSC   PIR1,TMR2IF
        GOTO    TMR2_ISR
```

Low priority
Two interrupts
Check Flag

```
MAIN:

        BSF RCON,     IPEN
        BSF INTCON,   GIEH
        BSF INTCON2,  INTEDG1
        BSF INTCON3,  INT1IP
        BSF INTCON3,  INT1IE
```

INT is enabled
Edge driven

```
        BSF INTCON, GIEL
        BCF IPR1,  TMR1IP
        BSF PIE1,  TMR1IE
        BCF IPR1,  TMR2IP
        BSF PIE1,  TMR2IE
```

INT is enabled
Edge driven

```
0000    EF12    GOTO 0x24
0002    F000    NOP
0004    FFFF    NOP
0006    FFFF    NOP
0008    EF80    GOTO 0x100
000A    F000    NOP
000C    FFFF    NOP
000E    FFFF    NOP
0010    FFFF    NOP
0012    FFFF    NOP
0014    FFFF    NOP
0016    FFFF    NOP
0018    B09E    BTFSC 0xf9e, 0, ACCESS
001A    EF87    GOTO 0x10e
001C    F000    NOP
001E    B29E    BTFSC 0xf9e, 0x1, ACCESS
0020    EF91    GOTO 0x122
0022    F000    NOP
        0E3F    MOVLW 0x3f
        6E93    MOVWF 0xf93, ACCESS
0028    8ED0    BSF 0xfd0, 0x7, ACCESS
002A    8EF2    BSF 0xff2, 0x7, ACCESS
002C    8AF1    BSF 0xff1, 0x5, ACCESS
002E    8CF0    BSF 0xff0, 0x6, ACCESS
0030    86F0    BSF 0xff0, 0x3, ACCESS
0032    8CF2    BSF 0xff2, 0x6, ACCESS
0034    909F    BCF 0xf9f, 0, ACCESS
0036    809D    BSF 0xf9d, 0, ACCESS
0038    929F    BCF 0xf9f, 0x1, ACCESS
003A    829D    BSF 0xf9d, 0x1, ACCESS
003C    0E0A    MOVLW 0xa
003E    6E01    MOVWF 0x1, ACCESS
0040    EF20    GOTO 0x40
```

MAIN

XC8 Interrupt Handling

```
void __interrupt("high_priority") CheckButtonPressed(){
        int k=0;
        if(INTCONbits.INT0IF){
            for(k=0;k<10;k++){
                PORTDbits.RD0=!PORTDbits.RD0;
                __delay_ms(1000);
            }
            INTCONbits.INT0IF=0;
            PORTDbits.RD0=0;
        }
}
```

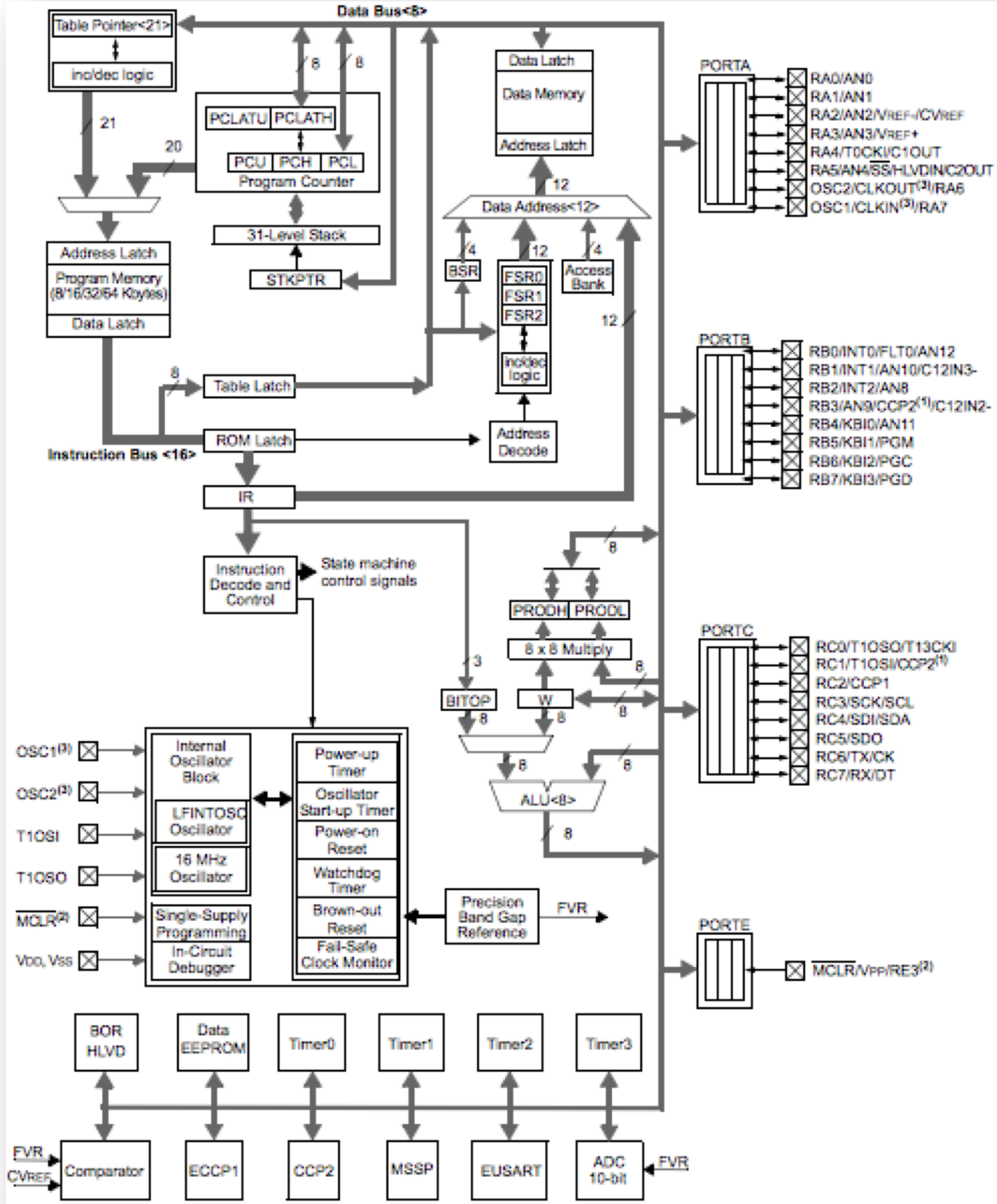
The ISR can be defined as high_priority or low_priority

This **Interrupt Vector** is called CheckButtonPressed
The ISR for IV is defined here

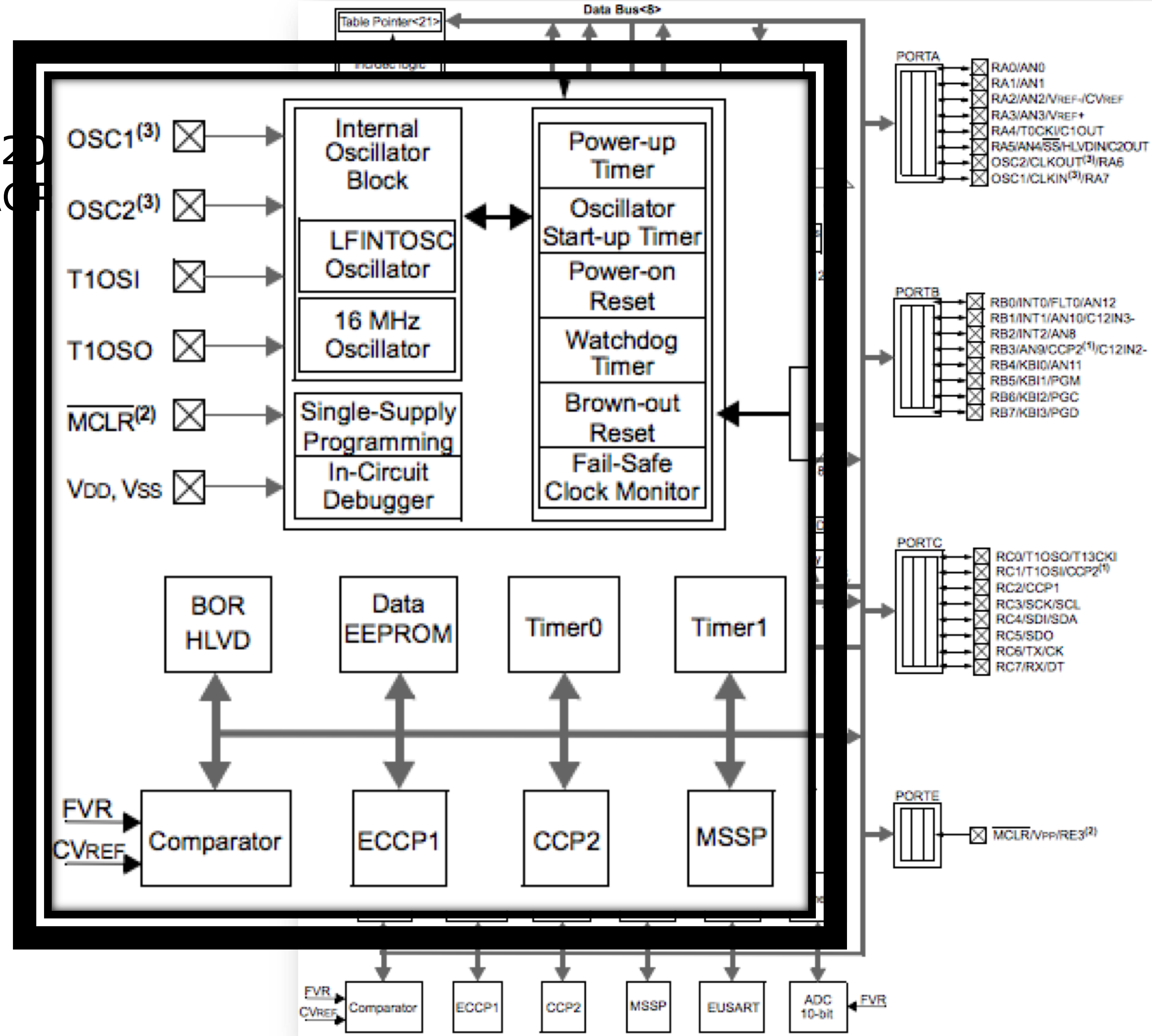
With in the Interrupt definition
It is important to clear the Interrupt FLAG!

```
void main(void) {

    ANSELH = 0x00; //Set RB<4:0> as digital I/O pins
    INTCON2bits.RBPU = 1; // Port B pull-ups on
    TRISD = 0x0; //set port D as output

    ADCON1 = 0x0F; // make ports pins digital
    TRISB = 1; // make RB0 input
    RCONbits.IPEN = 1; // IPEN = 1
    INTCON2bits.INTEDG0 = 1; // make INT0 negative edge triggered
    INTCONbits.INT0IE = 1; // enable INT0
    INTCONbits.GIEH = 1; // enable high priority interrupts
    // INT0 is now armed and active
     INTCONbits.INT0IF = 0;


    PORTDbits.RD1 = 1;          // pin 19
    __delay_ms(500);
    PORTDbits.RD1 = 0;

    while(1)
    {
        asm ("NOP");
    }
}
```

PIC18F2XK20 (28-PIN) BLOCK DIAGRAM



PIC18F2XK2
BLOCK DIAG



SLEEP and WDT

The SLEEP mode can reduce the power!
The CLRWDT is used to reset the WDT
Refer to the SPEC sheet.

```
#define ON        1
#define OFF       0
#define LED_0     PORTDbits.RD0
#define LED_1     PORTDbits.RD1
```

```
while(1)
{
        //SLEEP();
    asm ("SLEEP");
    LED_0 = ON;
    __delay_ms(500);
    LED_0 = OFF;
    asm ("CLRWDT");
    LED_1 = ON;
    __delay_ms(2000);
}
```

REGISTER 23-3: CONFIG2H: CONFIGURATION REGISTER 2 HIGH

U-0	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN
bit 7							bit 0

Legend:
R = Readable bit   P = Programmable bit   U = Unimplemented bit, read as '0'
-n = Value when device is unprogrammed   x = Bit is unknown

bit 7-5 **Unimplemented:** Read as '0'

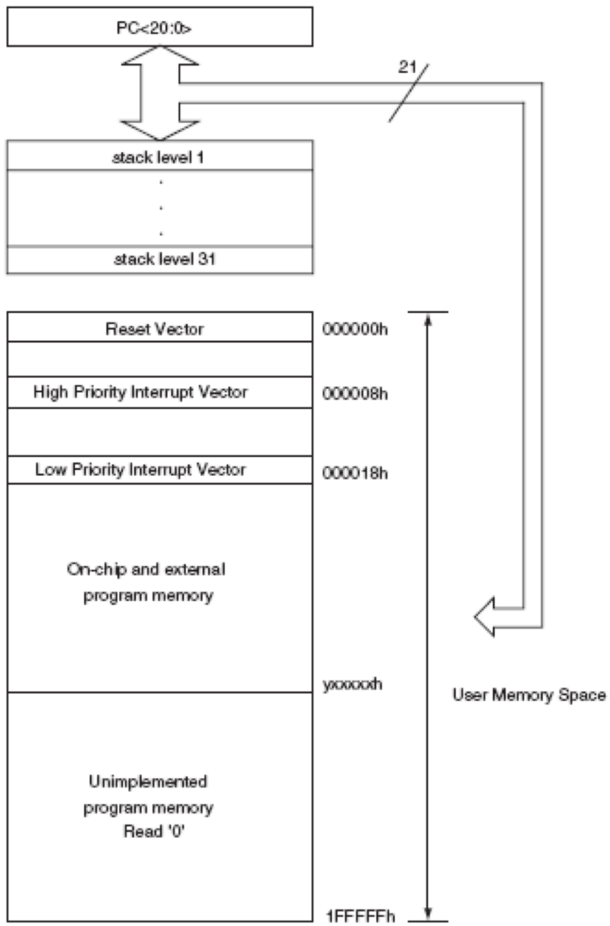
bit 4-1 **WDTPS<3:0>:** Watchdog Timer Postscale Select bits
1111 = 1:32,768
1110 = 1:16,384
1101 = 1:8,192
1100 = 1:4,096
1011 = 1:2,048
1010 = 1:1,024
1001 = 1:512
1000 = 1:256
0111 = 1:128
0110 = 1:64
0101 = 1:32
0100 = 1:16
0011 = 1:8
0010 = 1:4
0001 = 1:2
0000 = 1:1

bit 0 **WDTEN:** Watchdog Timer Enable bit
1 = WDT is always enabled. SWDTEN bit has no effect
0 = WDT is controlled by SWDTEN bit of the WDTCON register

C statement	Assembly Language	Scaling factor	Time to Reset
#pragma config WDTPS = 1	_WDTPS_1_2H	1:1	4 ms
#pragma config WDTPS = 32768	_WDTPS_32768_2H	1:32768	131.072 sec

PIC18 Resets

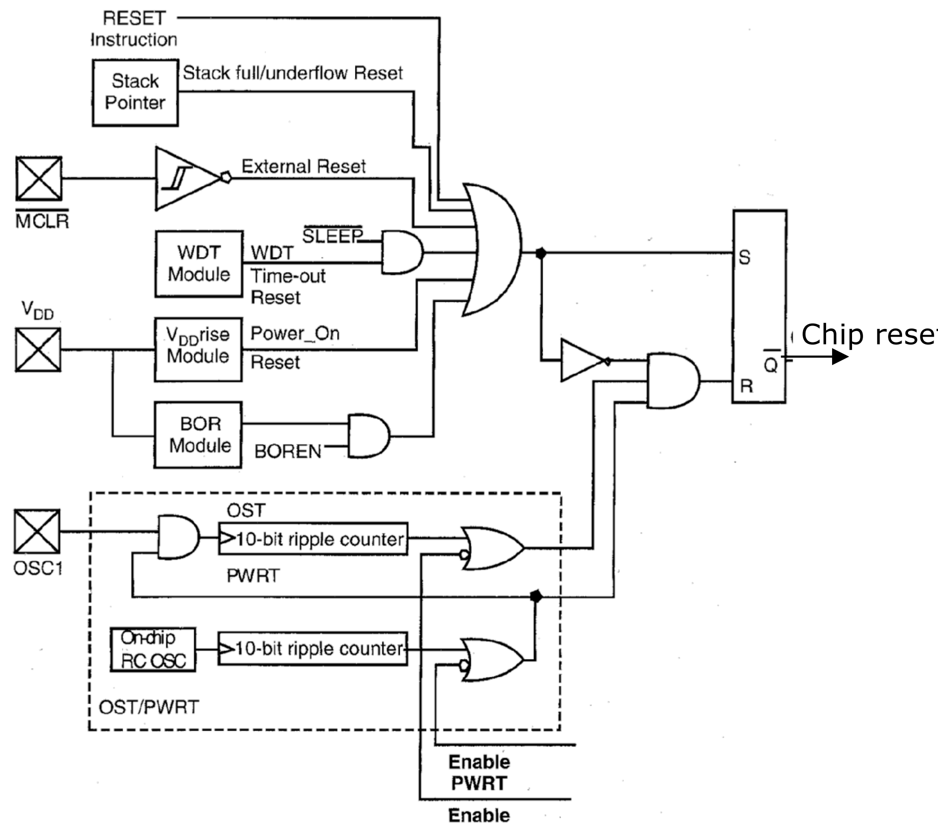
- When the reset signal is activated:
  - The MPU goes into a reset state during which the initial conditions are established.
  - The program counter is cleared to 000000 which is called the reset vector.
  - The MPU begins the execution of instructions from location 000000.



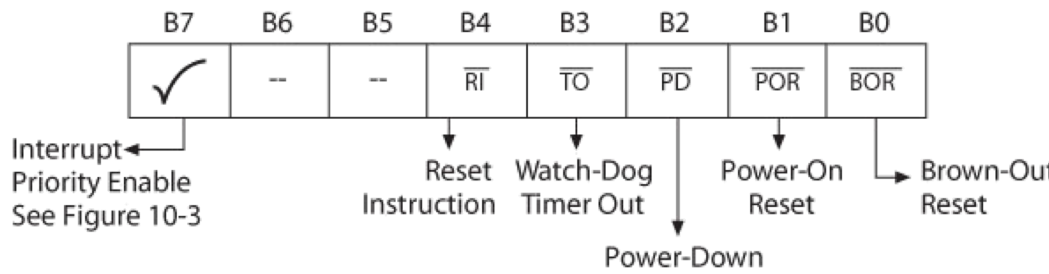
Note. y can be 0 or 1 whereas x can be 0-F

On Chip reset circuit for PIC18

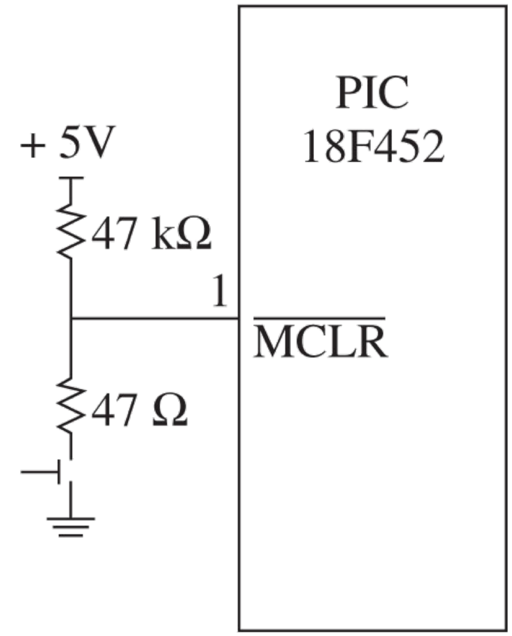
- Power-on reset (POR)
- $\overline{MCLR}$ pin reset during normal operation
- $\overline{MCLR}$ pin reset during SLEEP
- Watchdog timer (WDT) reset (during normal operation)
- Programmable brown-out reset (BOR)
- RESET instruction
- Stack full reset
- Stack underflow reset



PIC18 Resets



- PIC18 MCU can be reset by external source such as the push-button key, or when power is turned-on, or by various internal sources.
  - Resets categorized as follows:
    - External Manual Reset Key
    - Power-on Reset (POR)
    - Watchdog Timer Reset (WDT)
    - Programmable Brown-Out Reset (BOR)
    - RESET and SLEEP Instructions
    - Stack Full and Underflow Reset



Find MCLR pin!

Example of Reset Programming

- Identifying a power-on reset
- IF_ RCON,NOT_POR == 0 ;POR has occurred
- setf RCON ;Reinitialize all reset flags after power on
- <take action particular to power—on reset>
- ENDIF_
- Identifying a reset due to execution of a “reset” instruction
- IF_ RCON,NOT_RI == 0 ;reset’ instruction has been executed
- bsf RCON.NOT_RI ;Set bit to distinguish froe other resets
- <take appropriate action in response to “reset” instruction>
- ENDIF