# Interrupts

Hua-Hsi Tseng
曾華璽

*Networked Embedded Applications and Technologies Lab*

Department of Computer Science and Information Engineering
National Cheng Kung University, TAIWAN

## Outline

- Introduction to interrupt
- Interrupt in PIC18F4520
- Implementation in C
- Lab

## Introduction to interrupt

Interrupt in PIC18F4520
Implementation in C
Lab

## What is interrupt

- What is "Interrupt"?
  - An event that requires the CPU to stop normal process execution and perform some service.
  - Usually generated by hardware (although they can be initiated by software).
  - Often indicate that an event has occurred that needs an urgent response.
  - Interrupts may occur at any time.

## What is interrupt

- Before we have interrupts, the technique we used is:
  - Busy Waiting.(Polling)
    - A process repeatedly checks to see if a condition is true.

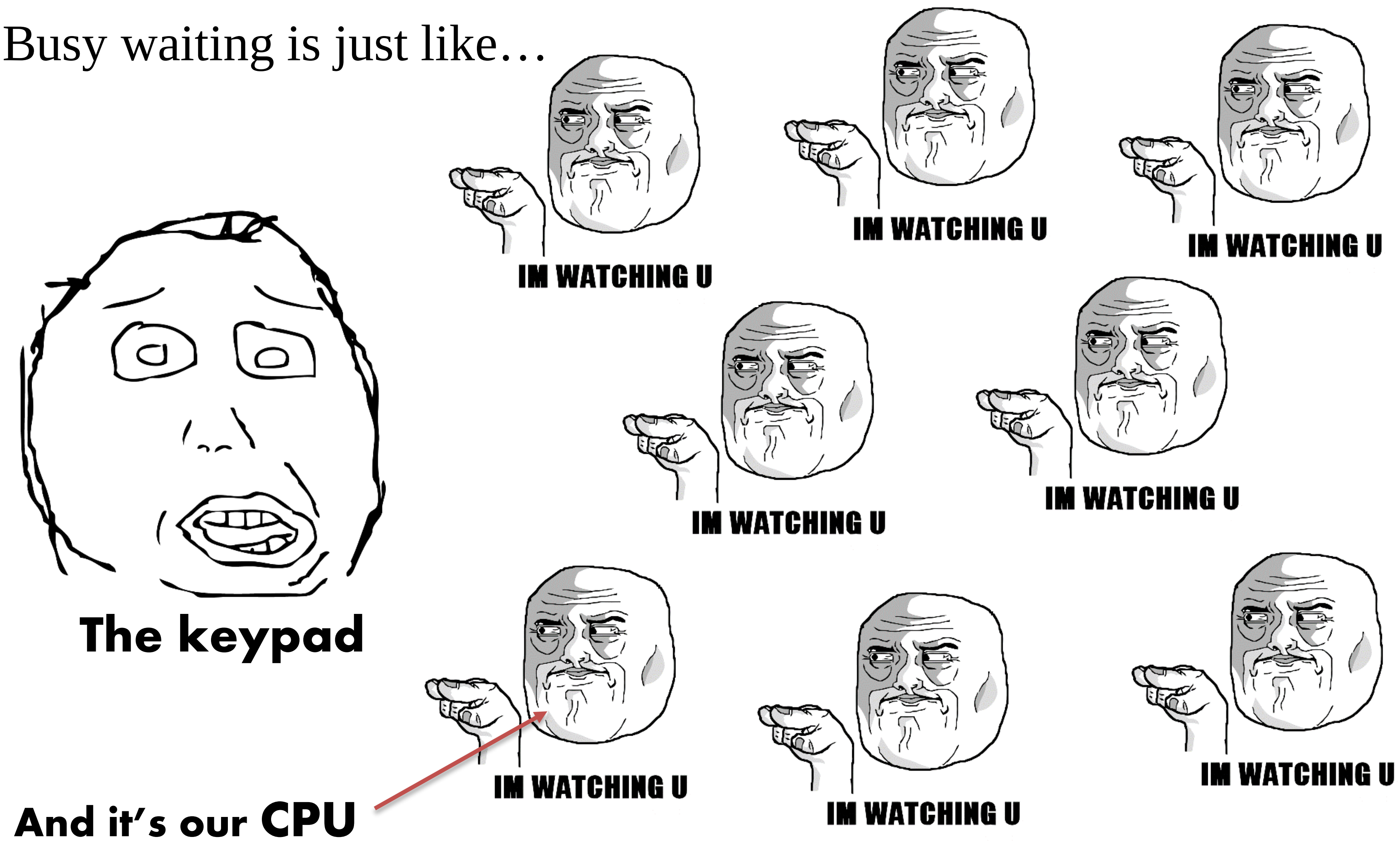
- But now we use interrupts.

In systems programming, an interrupt is a signal to the processor **emitted** by **hardware** or **software** indicating an event that **needs immediate attention**.

-Wikipedia, the savior of teaching assistants

## What is interrupt

Busy waiting is just like…



## What is interrupt

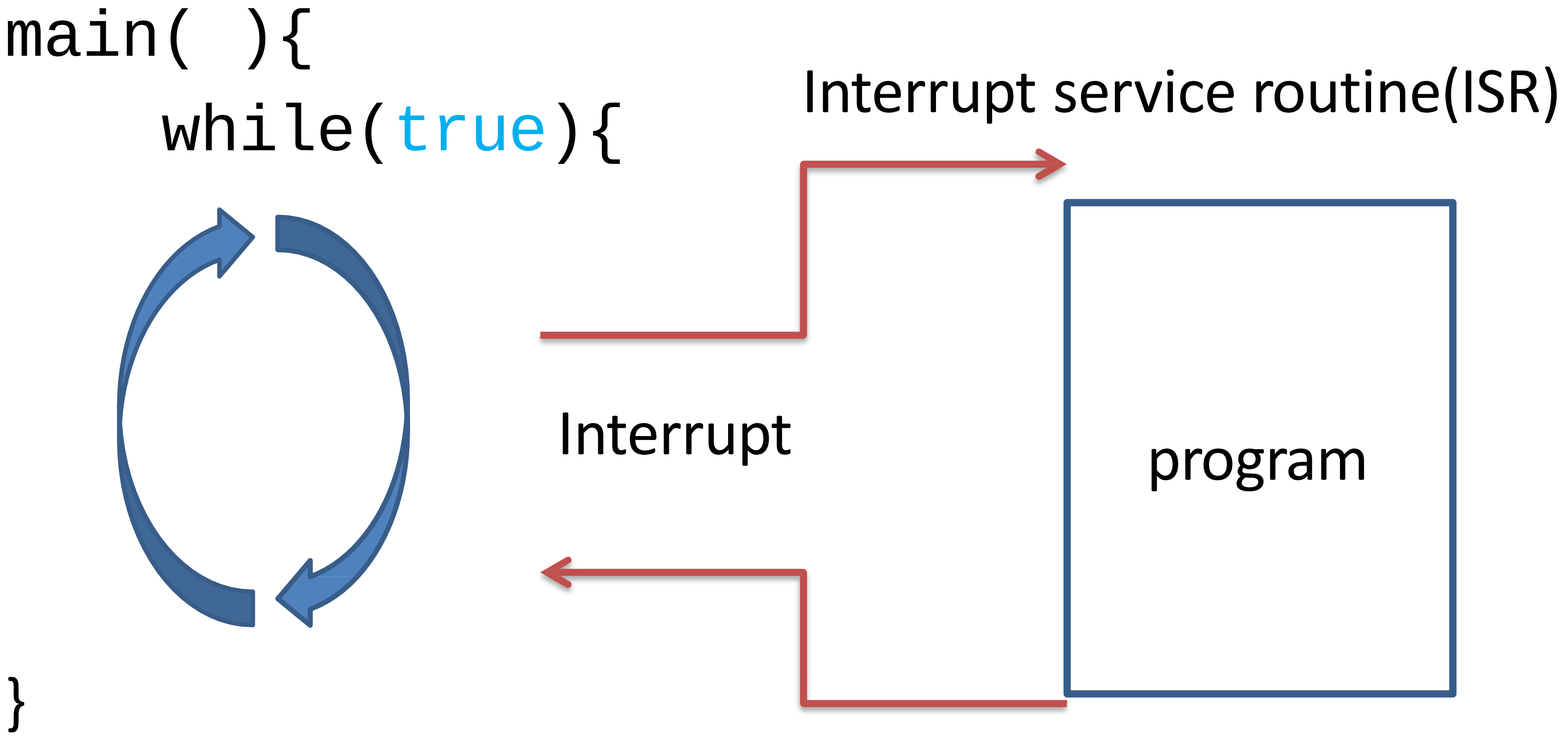
An interrupt is just like…

An emergency happened, Your Majesty
Ok, I'm going to handle it.

# Introduction to interrupt

- When an interrupt occurs, the processor :
  A. The global interrupt enable bit is cleared to disable further interrupts.
  B. If interrupt priority feature is enabled, high priority interrupt sources can interrupt a low priority interrupt. IPEN bit (RCON<7>).
  C. The return address is pushed onto the stack and the PC is loaded with the interrupt vector address. (0008h or 0018h).
  D. The interrupt flag bits must be cleared in software before re-enabling interrupts to avoid recursive interrupts.
  E. The "return from interrupt" instruction, RETFIE, exits the interrupt routine and sets the GIE bit (GIEH or GIEL if priority levels are used)

# Introduction to interrupt



```
main( ){
    while(true){



}
```

## Introduction to interrupt

THANKS FOR YOUR LISTENING...

Introduction to interrupt

## Interrupt in PIC18F4520

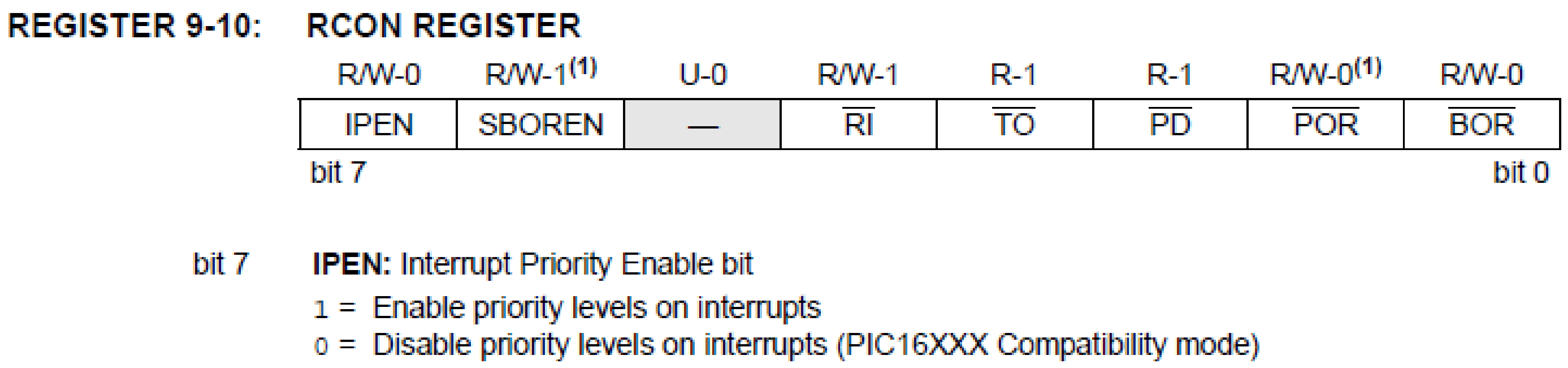
Implementation in C

Lab

## Interrupt in PIC18F4520

- In general, interrupt sources have three bits to control their operation. They are:
  - **Flag bit** to indicate that an interrupt event occurred.
  - **Enable bit** that allows program execution to branch to the interrupt vector address when the flag bit is set.
  - **Priority bit** to select high priority or low priority
- There are ten registers which are used to control interrupt operation. These registers are:
  - RCON
  - INTCON, INTCON2, INTCON3
  - PIR1, PIR2 (for the peripheral interrupts.)
  - PIE1, PIE2 (enable bits for the peripheral interrupts.)
  - IPR1, IPR2 (priority bits for the peripheral interrupts.)

# RCON REGISTER



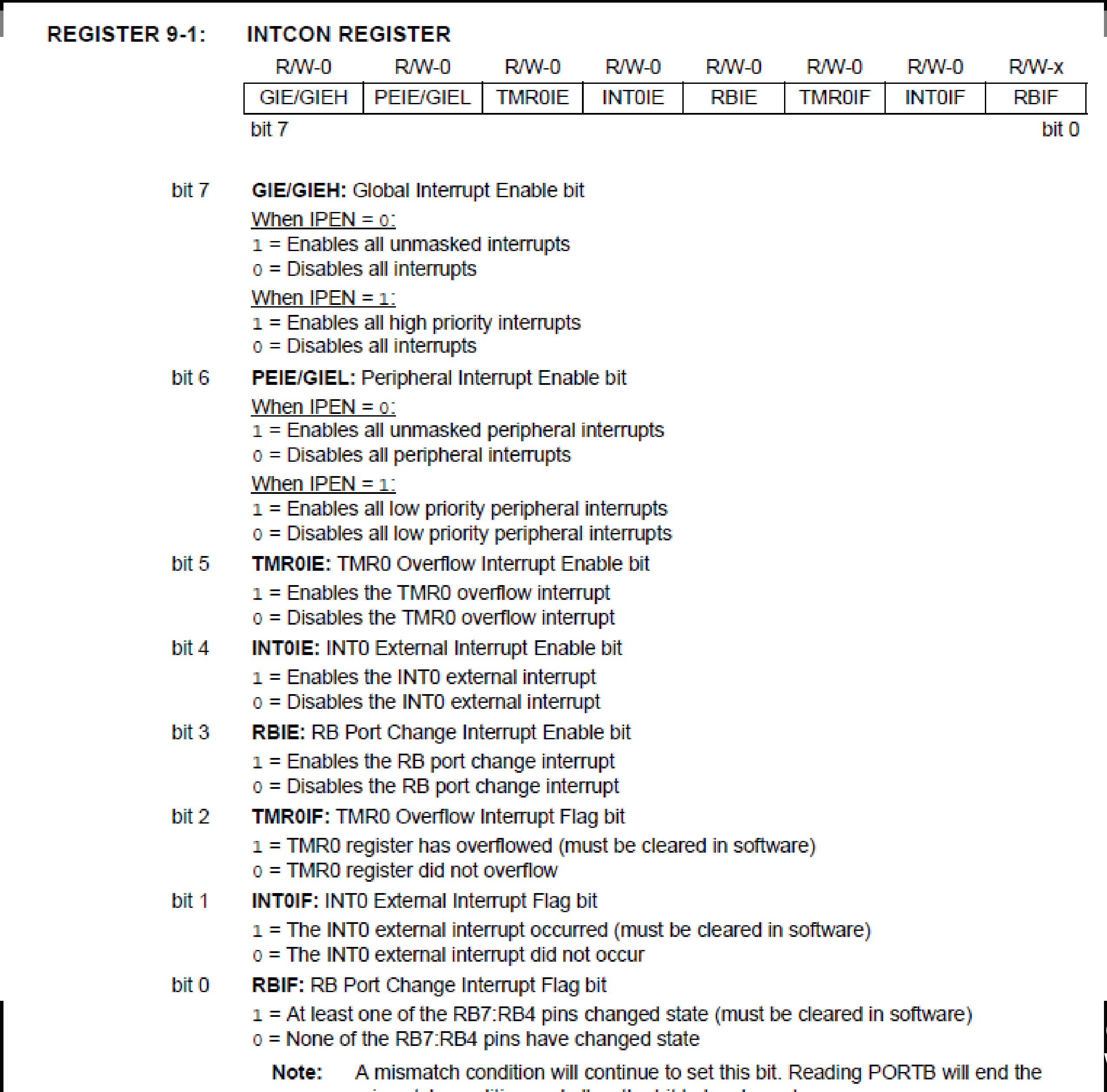
**REGISTER 9-10: RCON REGISTER**

| R/W-0 | R/W-1[1] | U-0 | R/W-1 | R-1 | R-1 | R/W-0[1] | R/W-0 |
|---|---|---|---|---|---|---|---|
| IPEN | SBOREN | — | $\overline{RI}$ | $\overline{TO}$ | $\overline{PD}$ | $\overline{POR}$ | $\overline{BOR}$ |
| bit 7 | | | | | | | bit 0 |

bit 7 **IPEN:** Interrupt Priority Enable bit

1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16XXX Compatibility mode)

# INTCON REGISTER



**REGISTER 9-1: 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/GIEH | PEIE/GIEL | TMR0IE | INT0IE | RBIE | TMR0IF | INT0IF | RBIF |
| bit 7 | | | | | | | bit 0 |

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 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

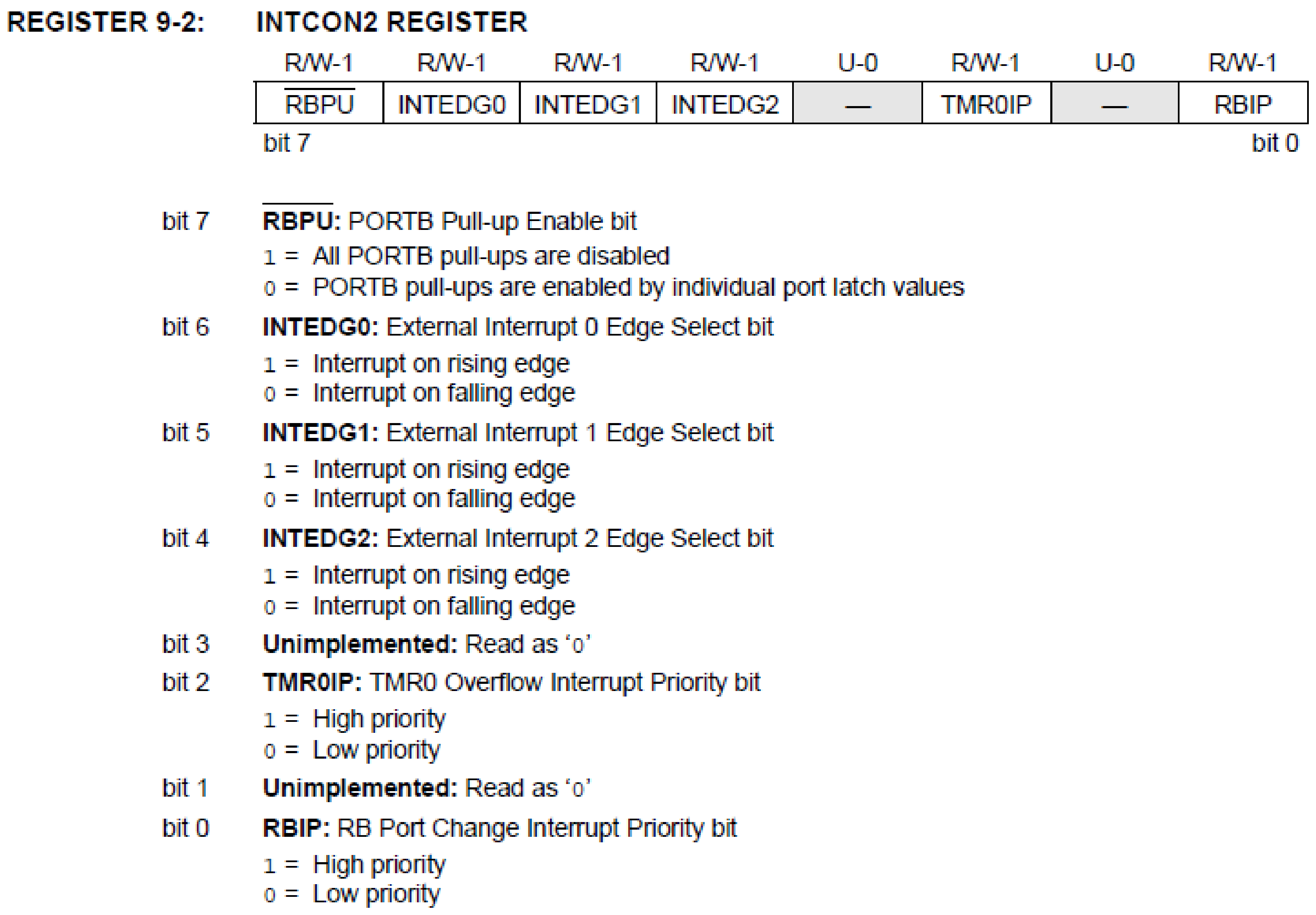
bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 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

**Note:** A mismatch condition will continue to set this bit. Reading PORTB will end the

# INTCON2 REGISTER



**REGISTER 9-2: INTCON2 REGISTER**

| R/W-1 | R/W-1 | R/W-1 | R/W-1 | U-0 | R/W-1 | U-0 | R/W-1 |
|---|---|---|---|---|---|---|---|
| $\overline{\text{RBPU}}$ | INTEDG0 | INTEDG1 | INTEDG2 | — | TMR0IP | — | RBIP |
| bit 7 | | | | | | | bit 0 |

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

bit 6 **INTEDG0:** External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1:** External Interrupt 1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2:** External Interrupt 2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 3 **Unimplemented:** Read as '0'

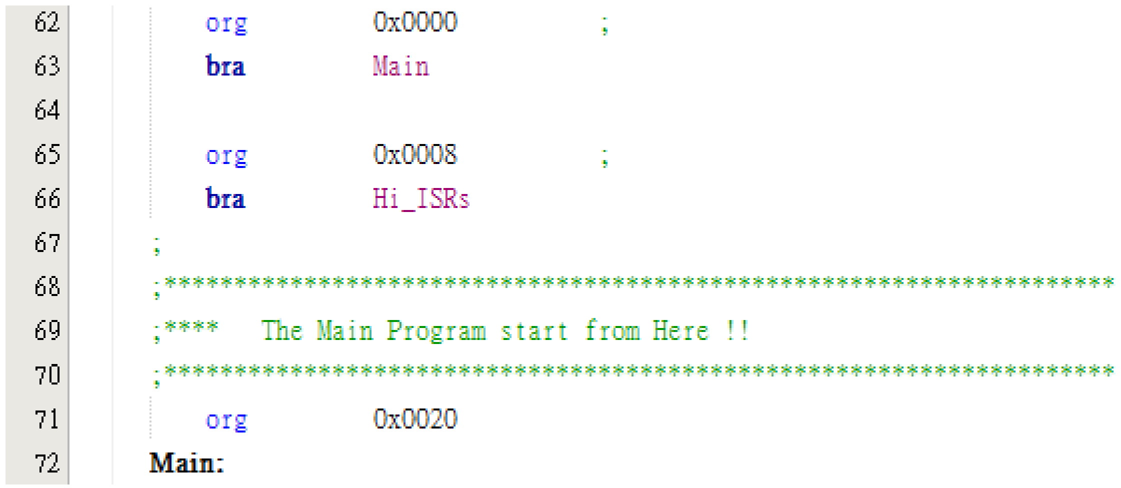
bit 2 **TMR0IP:** TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **RBIP:** RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

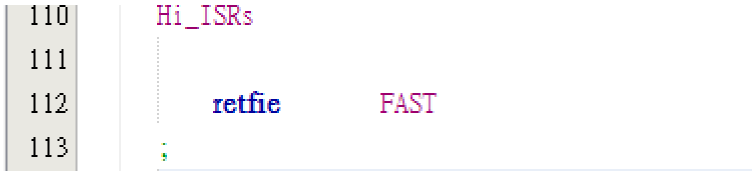
# Interrupt service routine(ISR)

- During interrupts, the return PC address is saved on the stack.
- Additionally, the WREG, Status and BSR registers are saved on the fast return stack.



```
        org         0x0000          ;
        bra         Main

        org         0x0008          ;
        bra         Hi_ISRs
;
;*********************************************************************
;****    The Main Program start from Here !!
;*********************************************************************
        org         0x0020
Main:
```

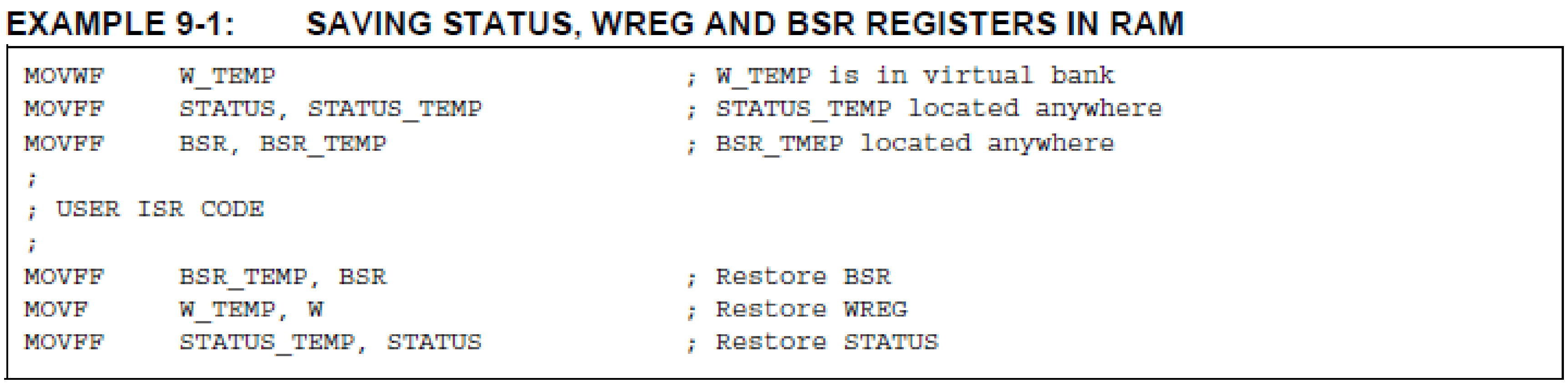


```
Hi_ISRs

        retfie          FAST
;
```

# Interrupt service routine(ISR)

- If a fast return from interrupt is not used, the user may need to save the WREG, Status and BSR registers on entry to the Interrupt Service Routine.



**EXAMPLE 9-1: SAVING STATUS, WREG AND BSR REGISTERS IN RAM**

```
MOVWF    W_TEMP                        ; W_TEMP is in virtual bank
MOVFF    STATUS, STATUS_TEMP           ; STATUS_TEMP located anywhere
MOVFF    BSR, BSR_TEMP                 ; BSR_TMEP located anywhere
;
; USER ISR CODE
;
MOVFF    BSR_TEMP, BSR                 ; Restore BSR
MOVF     W_TEMP, W                     ; Restore WREG
MOVFF    STATUS_TEMP, STATUS           ; Restore STATUS
```

## Try it

- 1. 觀察Lab7，加入movlw #20並將中斷點設在ISR內並仔細看WREG的變化，特別是在離開ISR後。

- 2. 修改Lab7的程式碼從
  - org 0x0008
  - bra Hi_ISRs
- 更改成如下:
  - org 0x0008
  - call Hi_ISRs,FAST
- 程式執行的順序將如何跳動？為什麼會那樣跳?

Introduction to interrupt

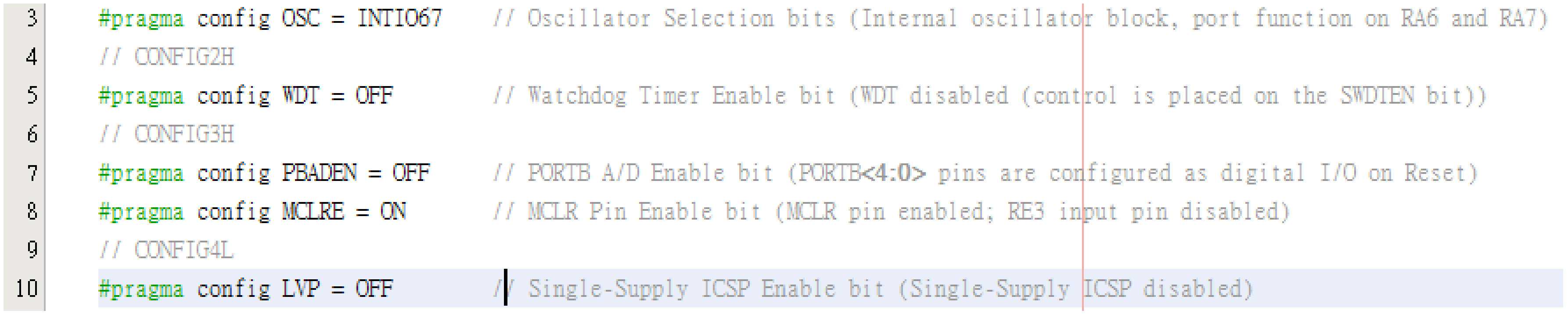
Interrupt in PIC18F4520

## Implementation in C

Lab

# Implementation in C

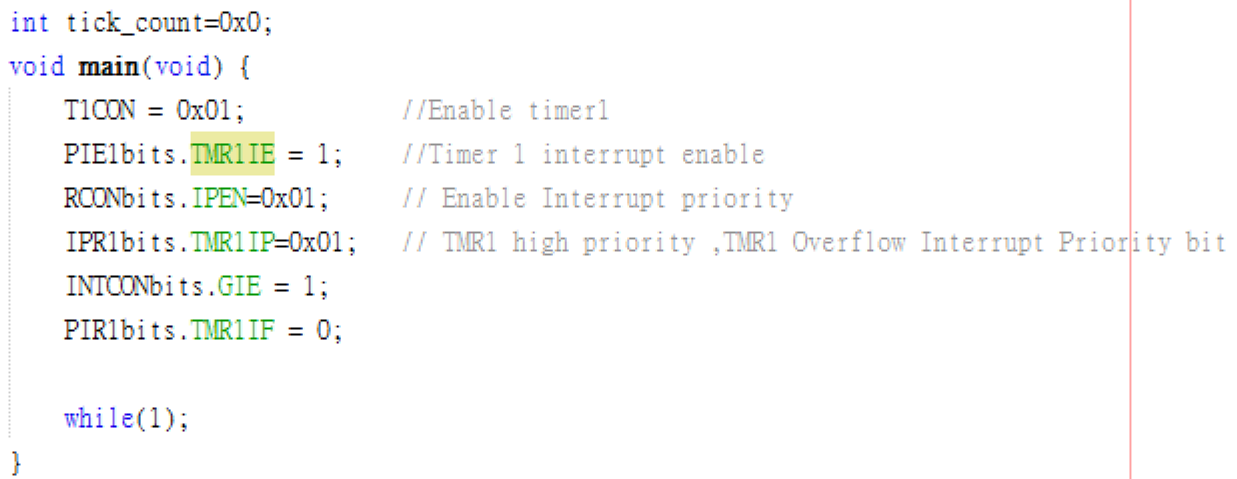
- Configuration Bits setting:
  - In MPLABXIDE >>Window >> PIC Memory Views >> Configuration Bits



```
#pragma config OSC = INTIO67    // Oscillator Selection bits (Internal oscillator block, port function on RA6 and RA7)
// CONFIG2H
#pragma config WDT = OFF        // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))
// CONFIG3H
#pragma config PBADEN = OFF     // PORTB A/D Enable bit (PORTB<4:0> pins are configured as digital I/O on Reset)
#pragma config MCLRE = ON       // MCLR Pin Enable bit (MCLR pin enabled; RE3 input pin disabled)
// CONFIG4L
#pragma config LVP = OFF        // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
```

## Implementation in C

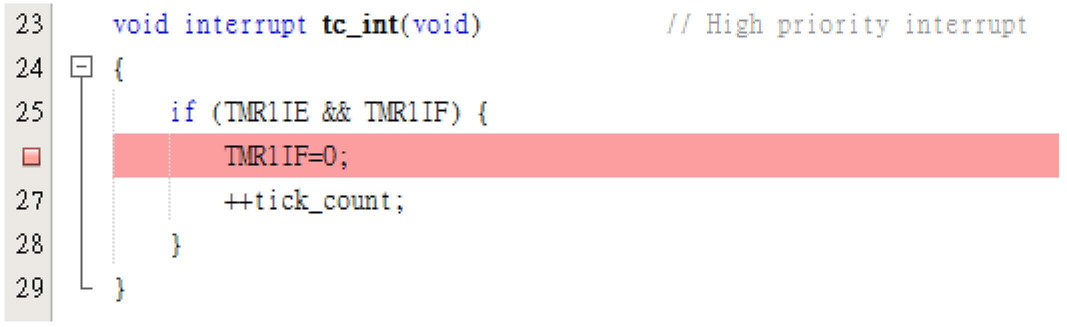
- Main function: (Timer interrupt example next week.)



```
int tick_count=0x0;
void main(void) {
    T1CON = 0x01;            //Enable timer1
    PIE1bits.TMR1IE = 1;     //Timer 1 interrupt enable
    RCONbits.IPEN=0x01;      // Enable Interrupt priority
    IPR1bits.TMR1IP=0x01;    // TMR1 high priority ,TMR1 Overflow Interrupt Priority bit
    INTCONbits.GIE = 1;
    PIR1bits.TMR1IF = 0;

    while(1);
}
```

- Interrupt function:



```
void interrupt tc_int(void)          // High priority interrupt
{
    if (TMR1IE && TMR1IF) {
        TMR1IF=0;
        ++tick_count;
    }
}
```

## 參考資料

- PIC18F4520 datasheet
  - http://ww1.microchip.com/downloads/en/devicedoc/39631a.pdf
- Microchip 教材 102ASP Example code
  - http://www.microchip.com.tw/Data_CD/Workshop/8-Bits/102ASP%20PIC18F452.zip
- How do I write interrupt routines in XC8?
  - http://microchipdeveloper.com/faq:31