Lab 7 – Interrupt programming

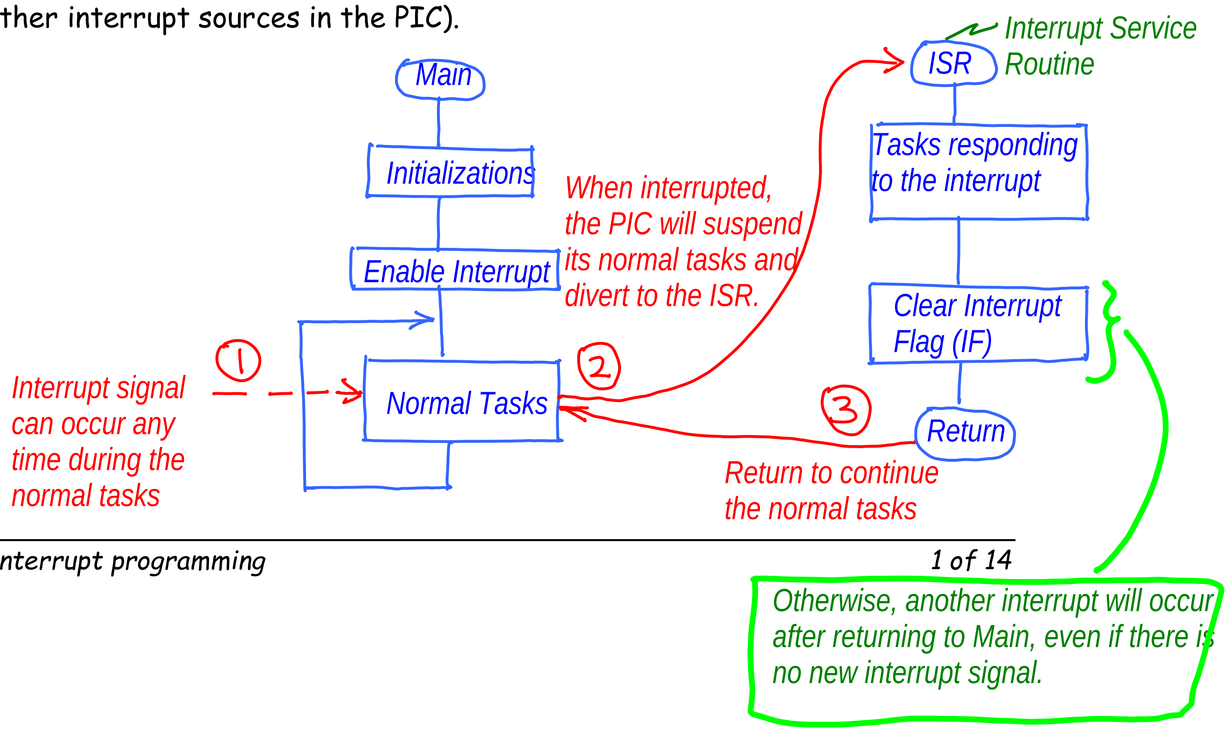
Objectives

- ☐ To learn to use PIC18F4550 microcontroller's INT0 external hardware interrupt & Timer0 interrupt.
- ☐ To learn to the sequence of code execution in an interrupt event.

Introduction / Briefing

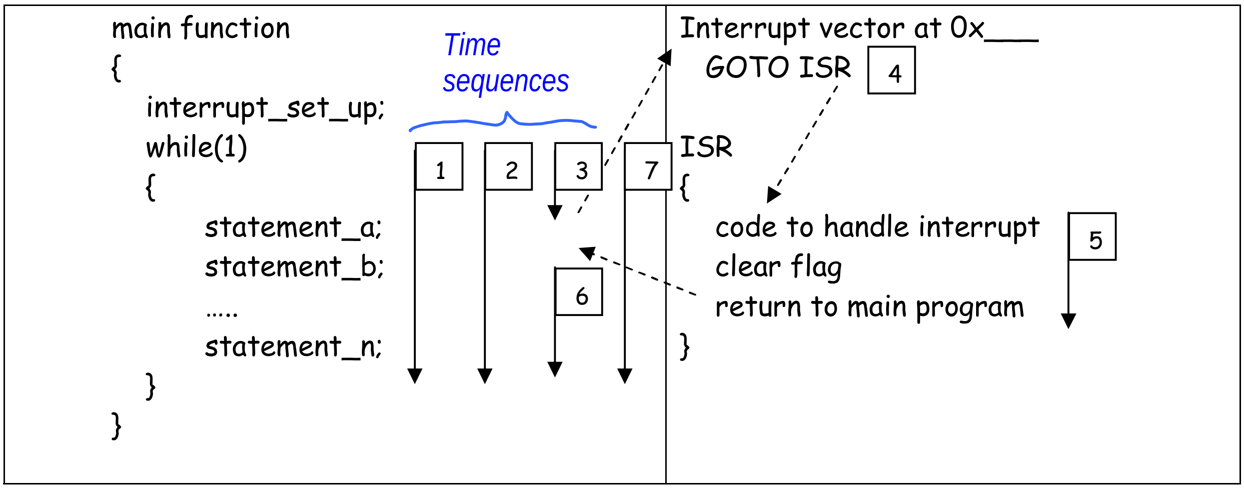
What is interrupt?

- ☐ A microcontroller can use the interrupt method to respond to an event.
- ☐ In this method, when a peripheral (e.g. an I/O pin or a timer) needs something to be done, it notifies the micro-controller, which stops whatever it is doing and "serves" the peripheral. After that, the micro-controller goes back to continue what it was doing.
- ☐ As an analogy, you could be reading newspaper. When there is a buzz tone on your hand phone, you are "interrupted" – you stop reading and reply an SMS. After that, you continue reading your newspaper where you left off.
- ☐ The program associated with the interrupt is aptly called the interrupt service routine or ISR.
- ☐ In this experiment, you will learn the basics of interrupt, focusing on INT0 external hardware interrupt and Timer0 interrupt, (though there are many other interrupt sources in the PIC).



Sequence of code execution

- The diagram below shows the sequence of code execution in an interrupt event:



- After interrupt has been set up, the while loop in the main function is executed over and over again (1, 2…).

- Let's say an interrupt event occurs when statement_a is being executed (3).

- The microcontroller will complete the execution of this statement. Then it goes to a specific location (*) called the interrupt vector to look for the ISR (interrupt service routine).

- Usually a single GOTO instruction can be found at the interrupt vector (4). This causes a branch to the ISR.

- In the ISR, the codes to handle the interrupt get executed (5).

- After that, the microcontroller returns to the main function to continue with statement_b (6), and the while loop gets executed over and over again (7…).

(*) In the lab, a "boot-loader" is used in the PIC18F4550. This program downloads a user program from a PC via the USB port. The "boot-loader" changes the high and low-priority interrupt-vectors to 0x000808 and 0x000818, respectively.

INT0 external hardware interrupt

☐ The PIC18F4550 has 3 external hardware interrupts: INT0, INT1 and INT2 which use pins RB0, RB1 and RB2 respectively. We will discuss INT0 (and INT1 and INT2 are similar in terms of operation).

☐ The INT0 interrupt responds to a change of voltage i.e. a transition at RB0.

Acts as the main switch for all interrupt sources

Act as individual switch for INT0 (at pin RB0)

☐ To enable the INT0 interrupt, set both the GIE (Global Interrupt Enable) and the INT0IE (INT0 Interrupt Enable) bits in the INTCON register.

INTCON (Interrupt Control Register)

GIE			INT0IE			INT0IF	
1			1				

This bit (Interrupt Flag) will become 1 if INT0 interrupt has occurred.

Q1. Give the C-code to enable the INT0 interrupt.

INTCONbits.GIE = 1;
INTCONbits . INT0IE = 1 ;

☐ The INTEDG0 bit of the INTCON2 register is used to specify whether interrupt is to occur on a falling (i.e. a high to low transition) or a rising (i.e. a low to high transition) edge at RB0:

INTCON2 (Interrupt Control Register 2)

	INTEDG0						
	0						

NGT

INTEDG0 = 0: INT0 interrupt on $\underbrace{\text{falling edge}}_{}$ at RB0
INTEDG0 = 1: INT0 interrupt on rising edge at RB0 (power-on reset default)

PGT

Q2. Give the C-code to select falling edge triggering.

INTCON2bits.INTEDG0= 0 ;

☐ Note that falling edge triggering has been chosen because on the microcontroller board, the push button switch at RB0 has been connected as "active low".

- ☐ When interrupt has been enabled and there is a falling edge at RB0, the flag INT0IF (external hardware INTerrupt 0 Interrupt Flag) in the INTCON register will be set. To clear this flag, so that future interrupt can be noticed, use the code INTCONbits.INT0IF = 0;

- ☐ With these, you should be able to understand the code in Int_INT0.c.

<u>Interrupt priority (D.I.Y.)</u>

- ☐ By default, all interrupts are "high priority".

- ☐ It is also possible to make some interrupts "high priority" and others "low priority". This is done by setting the **IPEN** (Interrupt Priority ENable) bit in the RCON register.

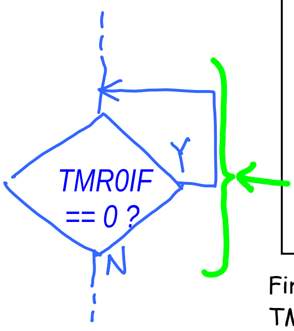
- ☐ When interrupt priority is enabled, we must classify each interrupt source as high priority or low priority. This is done by putting 0 (for low priority) or 1 (for high priority) in the **IP** (interrupt priority) bit of each interrupt source.

- ☐ The IP bits for the different interrupt sources are spread across several registers – INTCON, INTCON2, INTCON3, IPR1 and IPR2. We will not go into the details of all these.

- ☐ A higher priority interrupt can interrupt a low priority interrupt but NOT vice-versa.

Timer0 interrupt

Polling method
Disadvantage:
Spend most of the time waiting.



☐ In the last experiment, you used Timer0 to introduce a delay of 1 second. The C code was:

```
T0CON=0b00000111;          // Off Timer0, 16-bits mode, Fosc/4, prescaler to 256

TMR0H=0X48;                // Starting count value
TMR0L=0XE5;

INTCONbits.TMR0IF=0;       // Clear flag first
T0CONbits.TMR0ON=1;        // Turn on Timer 0

while(INTCONbits.TMR0IF==0);  // Wait for time is up when TMR0IF=1
T0CONbits.TMR0ON=0;        // Turn off Timer 0 to stop counting
```

First, Timer0 is configured but turned OFF. Then, the starting count value is written to TMR0H, followed by TMR0L. Next, the flag is cleared and Timer0 turned ON. After that, the while loop is used to wait for 1 second to elapse i.e. for the TMR0IF interrupt flag to be set. Finally, Timer0 is turned OFF.

☐ Instead of "polling" for the timer overflow (from FFFF to 0000) using
 while (INTCONbits.TMR0IF == 0);

the "interrupt" method can be used. The advantages of the interrupt method over the polling method are discussed in the lecture.

☐ The Timer0 interrupt responds to Timer0 overflow.

☐ To enable the Timer0 interrupt, set both the GIE (Global Interrupt Enable) and the Timer0IE (Timer0 Interrupt Enable) bits in the INTCON register.

INTCON (Interrupt Control Register)

GIE		TMR0IE			TMR0IF		
1		1					

Q3. Give the C-code to enable the Timer0 interrupt.

INTCONbits._*GIE*__ = _*1*____;
_*INTCONbits . TMR0IE = 1*____;

- ☐ When interrupt has been enabled and there is a Timer0 overflow, the flag Timer0IF (TiMeR0 overflow Interrupt Flag) in the INTCON register will be set. To clear this flag, so that future interrupt can be noticed, use the code INTCONbits.TMR0IF = 0;

- ☐ With these, you should be able to understand the Timer0 Interrupt code outlined as follows. (This is similar to, but not exactly the same as, the Int_TMR0.c used in the experiment.)

Initializations

main function	
INTCONbits.GIEH =1;	Global Interrupt Enable
INTCONbits.TMR0IE = 1;	Timer0 Interrupt Enable — Enable interrupt
T0CON = 0b00000111;	Stop Timer0, 16-bit, Fosc/4, pre-scalar 256
TMR0H = 0x48; TMR0L = 0xE5;	Starting count value for a 1 second delay
INTCONbits.TMR0IF = 0;	Clear Flag
T0CONbits.TMR0ON = 1;	Turn on Timer0 — Configure Timer0 for 1 sec delay

while (1)
{
… *Normal Tasks*
}
	PIC free to do other things in while loop e.g. use slide switch to turn motor on/off

ISR	*(Needed if other interrupt sources have also been enabled.)*
if (INTCONbits.TMR0IF) *==1* {	Check that it is really Timer0 overflow causing the interrupt
TMR0H = 0x48; TMR0L = 0xE5;	Reload Timer0 with starting count value - for the next round
…*Other tasks (if any)*	
INTCONbits.TMR0IF = 0; }	Clear flag, to get ready for the next interrupt — Respond to interrupt and reload for next interrupt.