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.

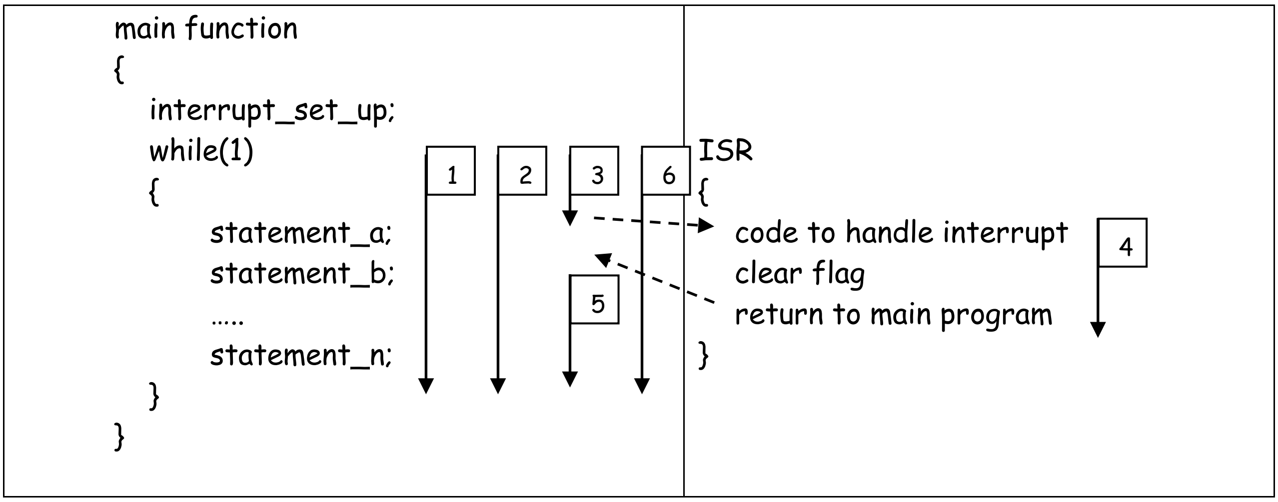
Introduction / Briefing

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 will go to a specific location (*) called the "interrupt vector" to look for the ISR (interrupt service routine).
- In the ISR, the codes to handle the interrupt will get executed (4).
- After that, the microcontroller will return to the main function to continue with statement_b (5), and the while loop will get executed over and over again (6…).

(*) 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 0x001008 and 0x001018, 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.

☐ 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				

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

INTCONbits.______ = ______;
__________________________;

☐ 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						

INTEDG0 = 0: INT0 interrupt on falling edge at RB0
INTEDGO = 1: INT0 interrupt on rising edge at RB0 (power-on reset default)

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

INTCON2bits.______ = ______;

☐ 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_b.c.

Interrupt priority (D.I.Y.)

☐ 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

☐ 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._______ = _______;
___________________________;

☐ 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.)

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-scaler 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)
{
…
}
	PIC free to do other things in while loop e.g. use slide switch to turn motor on/off

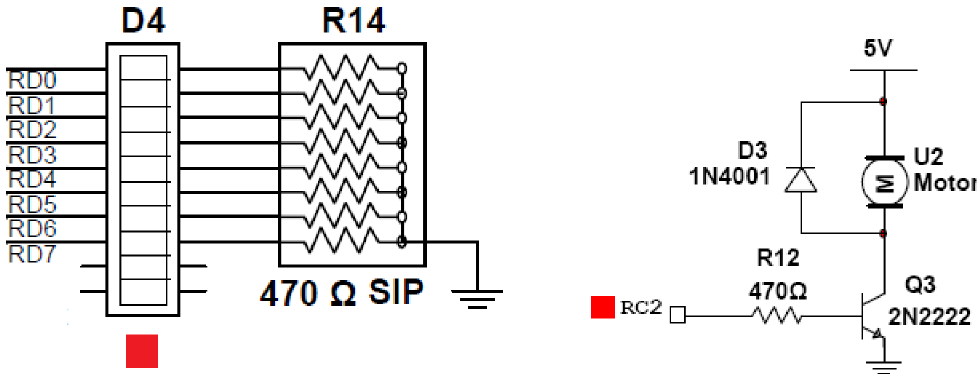
ISR	
if (INTCONbits.TMR0IF) {	Check that it is really Timer0 overflow causing the interrupt
TMR0H = 0x48; TMR0L = 0xE5; ….	Reload Timer0 with starting count value - for the next round
INTCONbits.TMR0IF = 0; }	Clear flag, to get ready for the next interrupt — Respond to interrupt and reload for next interrupt.

Activites:

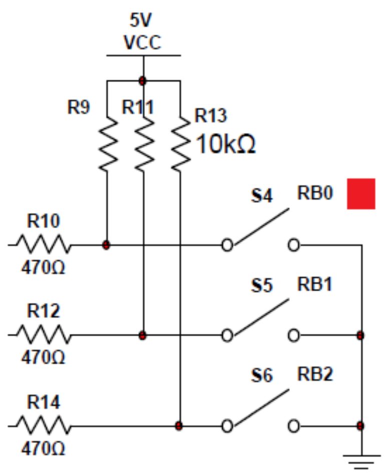
Before you begin, ensure that the Micro-controller Board is connected to the General IO Board.

External hardware interrupt

1. This part of the experiment uses the *LED bar* connected to *Port D* and the *motor* connected to *RC2*. Both are found on the *General IO Board*.



2. This part of the experiment also uses the *push button* connected to *RB0*. This is found on the *Micro-controller Board*.



3. Launch the *MPLABX IDE* and create a new project called *Lab7*.

4. Add the file *Int_INT0_a.c* to the *Lab7* project *Source File* folder. Make sure *Copy* is ticked before you click *Select*. If you have forgotten the steps, you will need to refer to the previous lab sheet.

Polling an active low switch at RB0

5. Study the code. In the main program the function "*LED_RD7_RD0*" is first called to light up the LED's in the bar, one by one, from left to right.

Then the switch at *RB0* is checked. The first time it is pressed (i.e. becomes 0), the motor at *RC2* is turned on. The next time it is pressed, the motor is turned off.

No interrupt is used and the *RB0* switch is responded to only after the LED bar sequence has completed.

6. Build, download and execute the program. Press the switch at *RB0* to control the motor on/off. Does the motor respond promptly to the switch?

   Your answer: ____

Using interrupt for the active low switch at RB0

7. Replace *Int_INT0_a.c* with *Int_INT0_b.c* .

8. Study the code. This time, *INT0* interrupt is used.

   Can you find the main function and the ISR?

   In the main function, can you find the codes that enable the interrupt, select falling edge triggering and clear the flag?

9. In the while loop in the main function, the function "*LED_RD7_RD0*" is called to light up the LED's in the bar, one by one, from left to right.

10. Pressing the switch at *RB0* causes an interrupt event to occur. The interrupt service routine for the "active low" button at *RB0* is called *ISR_PortB0_low*.

    The ISR first checks that the *INT0IF* is set.

    The first time the switch at *RB0* is pressed (i.e. becomes 0), the motor at *RC2* is turned on. The next time it is pressed, the motor is turned off.

    The ISR clears the flag at the end so that the next interrupt event can be recognised and responded to.

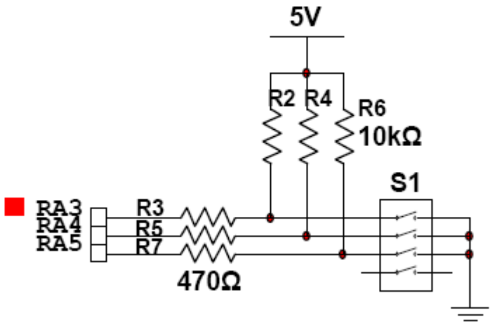
    {You may have noticed that there is no line of code "checking" if *RB0* has been pressed in the form of "*if (PORTBbits.RB0 == 0)*". This is because interrupt is used, instead of polling.}

11. Build, download and execute the program. Press the switch at *RB0* to control the motor on/off. Does the motor respond promptly to the switch?

    Your answer: ____

Timer0 interrupt

12. This part of the experiment also uses the on/off *switch* connected to *RA3*. This is found on the *General IO Board*.



Using interrupt for Timer0 overflow

13. Replace *Int_INT0_b.c* with *Int_TMR0.c*.

14. Study the code. This time, *Timer0* interrupt is used.

    Can you find the main function and the ISR?

    In the main function, can you find the codes that enable the interrupt, configure the *Timer0*, clear the flag and turn on the *Timer0*?

15. In the while loop in the main function, the switch at *RA3* is used to turn the motor on/off.

16. *Timer0* overflow causes an interrupt event to occur. The interrupt service routine for the *Timer0* overflow is called *ISR_Timer0_Int*.

    The ISR first checks if *TMR0IF* is set.

    If so, the timer is reloaded with the starting count value (to get ready for the next round). A variable *j* is incremented and then displayed on the LED bar connected *Port D*.

    At the end, the *TMR0IF* flag is cleared, so that the next interrupt event can be recognised and responded to.

17. Build, download and execute the program. Turn the switch at *RA3* on/off to see if the motor can be turned on/off.

    Your answer: ____

18. Next, look at the LED bar and record your observation below:

    Your answer: ____

19. There are two processes – main and ISR - in the program but the micro-controller can only run one process at a time. Do you feel that the motor is responding well to the switching at *RA3* when the LED's are counting?

    Your answer: ____

Beeping every second

20. Finally, modify the program so that the buzzer will beep every second.

    Your answer: ____

21. Build, download and execute the program to verify your coding. Debug until the program can work.

// Int_**INT0_a.c** Polling based program

```
#include <xc.h>
#include "delays.h"

unsigned char j;
unsigned char press;

void LED_RD7_RD0(void)// The function to shift a set-bit from left to right
{
        j = 0x80;             // Initialise j with B1000 0000
                              // ie the leftmost bit (or MSB)

        while(j!=0x01)        // Check that the bit has not been shifted
                              // to the right-most bit (LSB) ie B00000001
        {
          PORTD = j;          // Display j at PORTD
          delay_ms(250);      // Calling a delay function from delays.h
          j = j>>1;           // Making use of LOGICAL-RIGHT-SHIFT bit-wise
        }                     // operator to shift data to the right

        PORTD = j;            // Display j at PORTD
}                             // Stop at B00000001


void main(void)               // Main Function
{
        ADCON1 = 0x0F;
        CMCON = 0x07;

        TRISBbits.TRISB0 = 1;       // RB0 is the push button switch for INT0
        TRISCbits.TRISC2 = 0;       // RC2 connects to a DC motor
        TRISD = 0x00;         // PortD connects to a bar LEDs

        PORTD = 0x00;         // LEDs all off
        press = 0;            // Not pressing yet

        while(1)              // Main Process
        {
            LED_RD7_RD0();    // Move Port D LEDs from bit7 to bit0

            // polling the switch at RB0
            if (PORTBbits.RB0 == 0)
            {
              press++;        // To track first or second time pressing RB0 switch
```

```
            if (press == 1)             // First press
            {
              PORTCbits.RC2 = 1;        // Turn On Motor
            }
            else
            if (press == 2)             // Second press
            {
              PORTCbits.RC2 = 0;        // Else turn Off Motor
              press = 0;                // Reset the pressing counter
            }
          }
       }
}
```

// Int_INT0_b.c

```
// Int_INT0_b.c Interrupt based program
// ISR activated by INT0 from an active low switch from RB0

#include <xc.h>
#include"delays.h"

unsigned char i, j;
unsigned char press, a, b;

void interrupt ISR_PortB0_low(void)     // Interrupt Service Routine for INT0
{
  if (INTCONbits.INT0IF)// External Interrupt Flag Bit = 1 when interrupt occurs
  {
       press++;             // To track first or second time pressing RB0 switch

       if (press == 1)             // First press
       {
         PORTCbits.RC2 = 1;        // Turn On Motor
       }
       else
       if (press == 2)             // Second press
       {
         PORTCbits.RC2 = 0;        // Else turn Off Motor
         press = 0;                // Reset the pressing counter
       }

        INTCONbits.INT0IF = 0;  //Clearing the flag at the end of the ISR
  }
}


void LED_RD7_RD0(void)// The function to shift a set-bit from the MSB to LSB
{
       j = 0x80;            // Initialise j with B1000 0000
                            // ie the leftmost bit (or MSB)

       while(j!=0x01)       // Check that the bit has not been shifted
                            // to the right-most bit (LSB) ie B00000001
       {
         PORTD = j;         // Display j at PORTD
         delay_ms(250);     // Calling a delay function from delays.h
         j = j>>1;          // Making use of LOGICAL-RIGHT-SHIFT bit-wise
       }                    // operator to shift data to the right

       PORTD = j;           // Display j at PORTD
}                           // Stop at B00000001
```

```
void main(void)             // Main Function
{

       ADCON1 = 0x0F;
       CMCON = 0x07;

       TRISBbits.TRISB0 = 1; // RB0 is the push button switch for INT0
       TRISCbits.TRISC2 = 0; // RC2 connects to a DC motor
       TRISD = 0x00;         // PortD connects to a bar LEDs

       PORTD = 0x00;         // LEDs all off
       press = 0;            // Not pressing yet
       j = 0;

       RCONbits.IPEN =1;     // Bit7 Interrupt Priority Enable Bit
                             // 1 Enable priority levels on interrupts
                             // 0 Disable priority levels on interrupts

       INTCONbits.GIEH =1;   // Bit7 Global Interrupt Enable bit
                             // When IPEN = 1
                             // 1 Enable all high priority interrupts
                             // 0 Disable all high priority interrupts

       INTCON2bits.INTEDG0 = 0;// Bit4 External Interrupt2 Edge Select Bit
                             // 1 Interrupt on rising edge
                             // 0 Interrupt on falling edge

       INTCONbits.INT0IE = 1;  // Bit4 INT0 External Interrupt Enable bit
                             // 1 Enable the INT0 external interrupt
                             // 0 Disable the INT0 external interrupt

       INTCONbits.INT0IF = 0;  // Clearing the flag

       while(1)              // Main Process
       {
            LED_RD7_RD0();   // Move Port D LEDs from bit7 to bit0
       }
}
```

// Int_TMR0.c

```
/* Int_TMR0.c Timer Interrupt based program
 * ISR activated by Timer0 interrupt when it over-flow
 * Set up a Timer0 interrupt-driven program to count up the LEDs at PORTD at
 * 1 sec interval
 *
 * Timer0 is configured for 16 bit Timer Mode operation.
 * The TMR0 interrupt is generated when the TMRO register
 * overflows from FFFFh to 0000h.
 * This overflow sets the TMR0IF bit.
 * The TMR0IF bit must be cleared in software by the Timer0 module
 * ISR before re-enabling this interrupt.
 *
 * Timer0 starting value is set by writing to TMR0H and TMR0L.
 * For 1 sec, the starting value is 0x48E5
 * Main Process - configure external interrupt and use RA3 switch to control
motor at RC2
*/

#include <xc.h>

unsigned char j;

void interrupt ISR_Timer0_Int()  // Timer0 Interrupt Service Routine (ISR)
{
  if (INTCONbits.TMR0IF)         // TMR0IF:- Timer0 Overflow Interrupt Flag Bit
                                 // 1 = TMR0 reg has overflowed
                                 // 0 = TMR0 reg has not overflowed
    {
        TMR0H = 0x48; // Timer0 start value = 0x48E5 for 1 second
        TMR0L = 0xE5;

        j++; // Increase count by 1
        PORTD = j; // Show count value at Port D Leds

        INTCONbits.TMR0IF = 0; // Reset TMR0IF to "0" since the end of
                          // the interrupt function has been reached
    }
}


void main(void)                  // Main Function
{

       ADCON1 = 0x0F;
       CMCON = 0x07;

       TRISAbits.TRISA3 = 1;     // RA3 is the On/Off switch
       TRISCbits.TRISC2 = 0;     // RC2 connects to a DC motor
       TRISD = 0x00;             // PortD connects to a bar LEDs

       PORTD = 0x00;             // LEDs all off
       j = 0;                    // Start count from 0

       RCONbits.IPEN =1;         // Bit7 Interrupt Priority Enable Bit
                                 // 1 Enable priority levels on interrupts
                                 // 0 Disable priority levels on interrupts


       INTCONbits.GIEH =1;       // Bit7 Global Interrupt Enable bit
                                 // When IPEN = 1
                                 // 1 Enable all high priority interrupts
                                 // 0 Disable all high priority interrupts

       T0CON = 0b00000111;       // bit7:0 Stop Timer0
                                 // bit6:0 Timer0 as 16 bit timer
                                 // bit5:0 Clock source is internal
                          // bit4:0 Increment on lo to hi transition on TOCKI
```

```
                                    // bit3:0 Prescaler output of Timer0
                                    // bit2-bit0:111 1:256 prescaler


       INTCON2 = 0b10000100;        // bit7 :PORTB Pull-Up Enable bit
                                    //      1 All PORTB pull-ups are disabled
                                    // bit2 :TMR0 Overflow Int Priority Bit
                                    //   1 High Priority

       TMR0H = 0x48;                // Initialising TMR0H
       TMR0L = 0xE5;                // Initialising TMR0L for 1 second interrupt

       T0CONbits.TMR0ON = 1;        // Turn on timer
       INTCONbits.TMR0IE = 1;       // bit5 TMR0 Overflow Int Enable bit
                                    // 0 Disable the TMR0 overflow int

       INTCONbits.TMR0IF = 0;       // bit2 TMR0 Overflow Int Flag bit
                                    // 0 TMR0 register did not overflow

    while (1) // Main Process
    {
        if (PORTAbits.RA3 == 0)     // If RA3 switch is ON

            PORTCbits.RC2 = 1;      // Turn On Motor

        else

            PORTCbits.RC2 = 0;      // Else turn Off Motor

    }

}
```