PIC Microcontroller (Lab 6)

Interrupts with Timer0 module

Leonardo Fusser, 1946995

Experiment Performed on **19 November 2020**
Report Submitted on **21 November 2020**

Department of Computer Engineering Technology
Microcontroller & Microprocessor systems
Day Yann Fong

Learning today Leading tomorrow

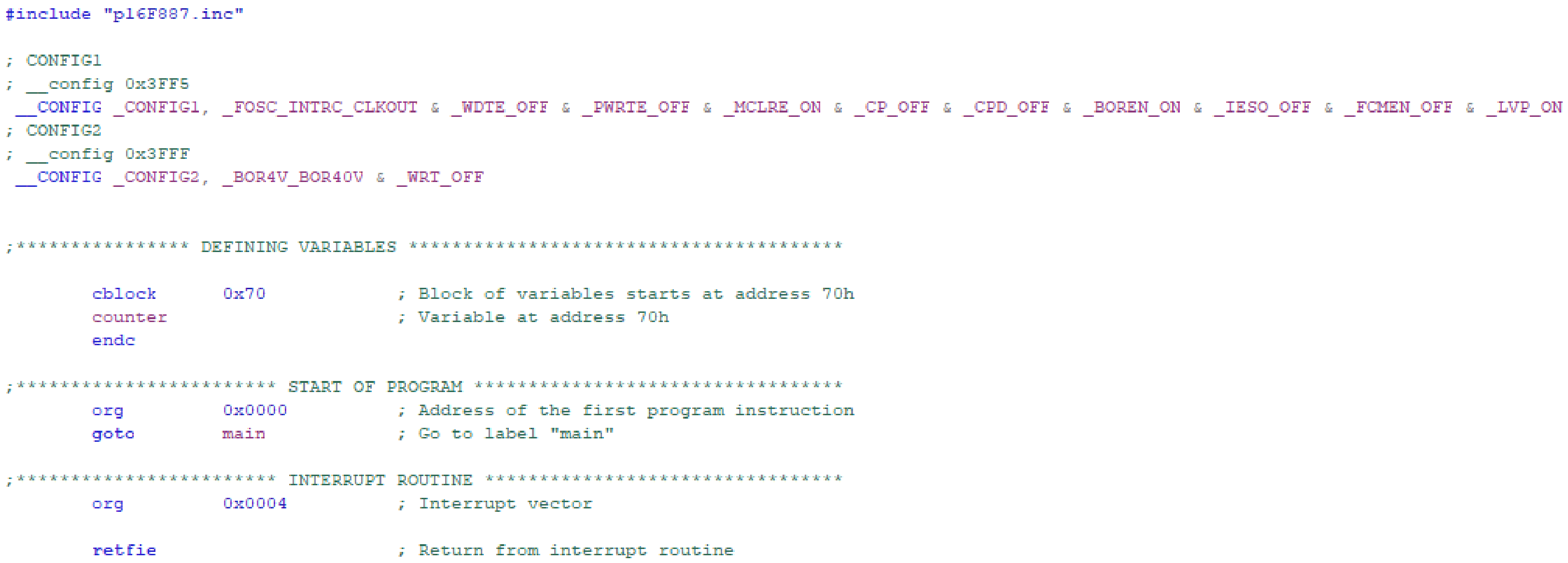
TABLE OF CONTENTS

1.0 Purpose

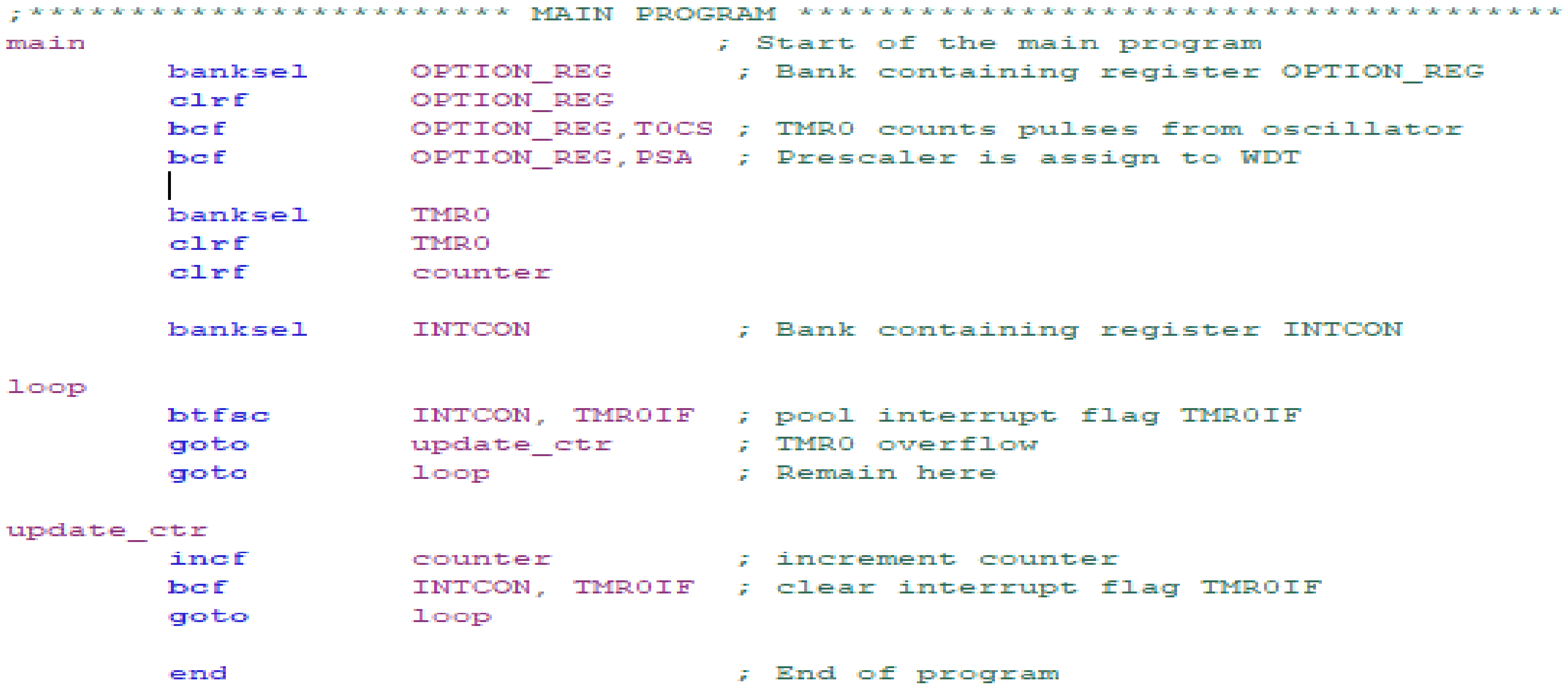
- Understand polling and interrupts and compare between the two.
- Learn how to use MPLAB IDE.
- Learn how to view TMR0 register using MPLAB IDE simulation in real time.
- Learn how to modify polling scheme to work as interrupt in MPLAB IDE simulation.

2.0 Original Design

Part A assembly code

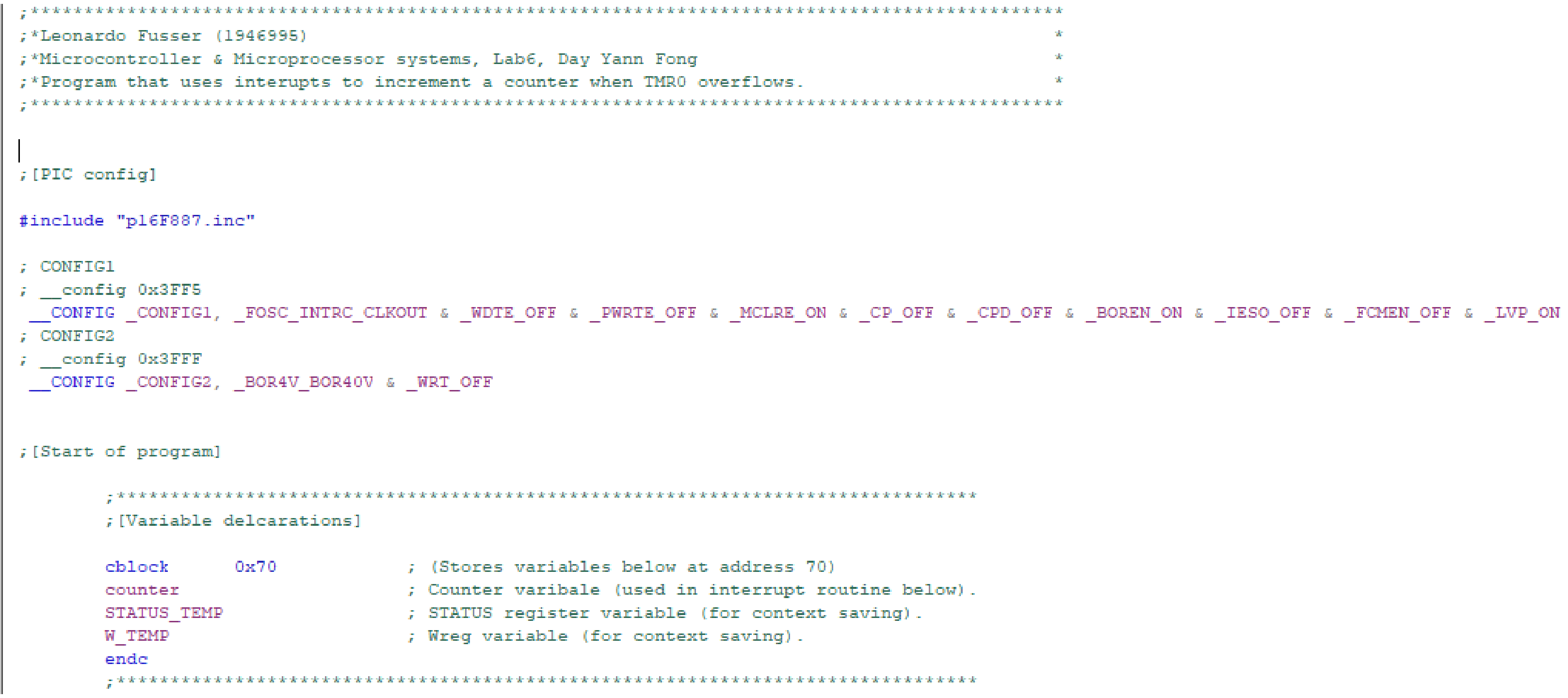


```
#include "p16F887.inc"

; CONFIG1
; __config 0x3FF5
 __CONFIG _CONFIG1, _FOSC_INTRC_CLKOUT & _WDTE_OFF & _PWRTE_OFF & _MCLRE_ON & _CP_OFF & _CPD_OFF & _BOREN_ON & _IESO_OFF & _FCMEN_OFF & _LVP_ON
; CONFIG2
; __config 0x3FFF
 __CONFIG _CONFIG2, _BOR4V_BOR40V & _WRT_OFF


;**************** DEFINING VARIABLES ****************************************

        cblock      0x70            ; Block of variables starts at address 70h
        counter                     ; Variable at address 70h
        endc

;************************ START OF PROGRAM **********************************
        org         0x0000          ; Address of the first program instruction
        goto        main            ; Go to label "main"

;************************ INTERRUPT ROUTINE *********************************
        org         0x0004          ; Interrupt vector

        retfie                      ; Return from interrupt routine
```

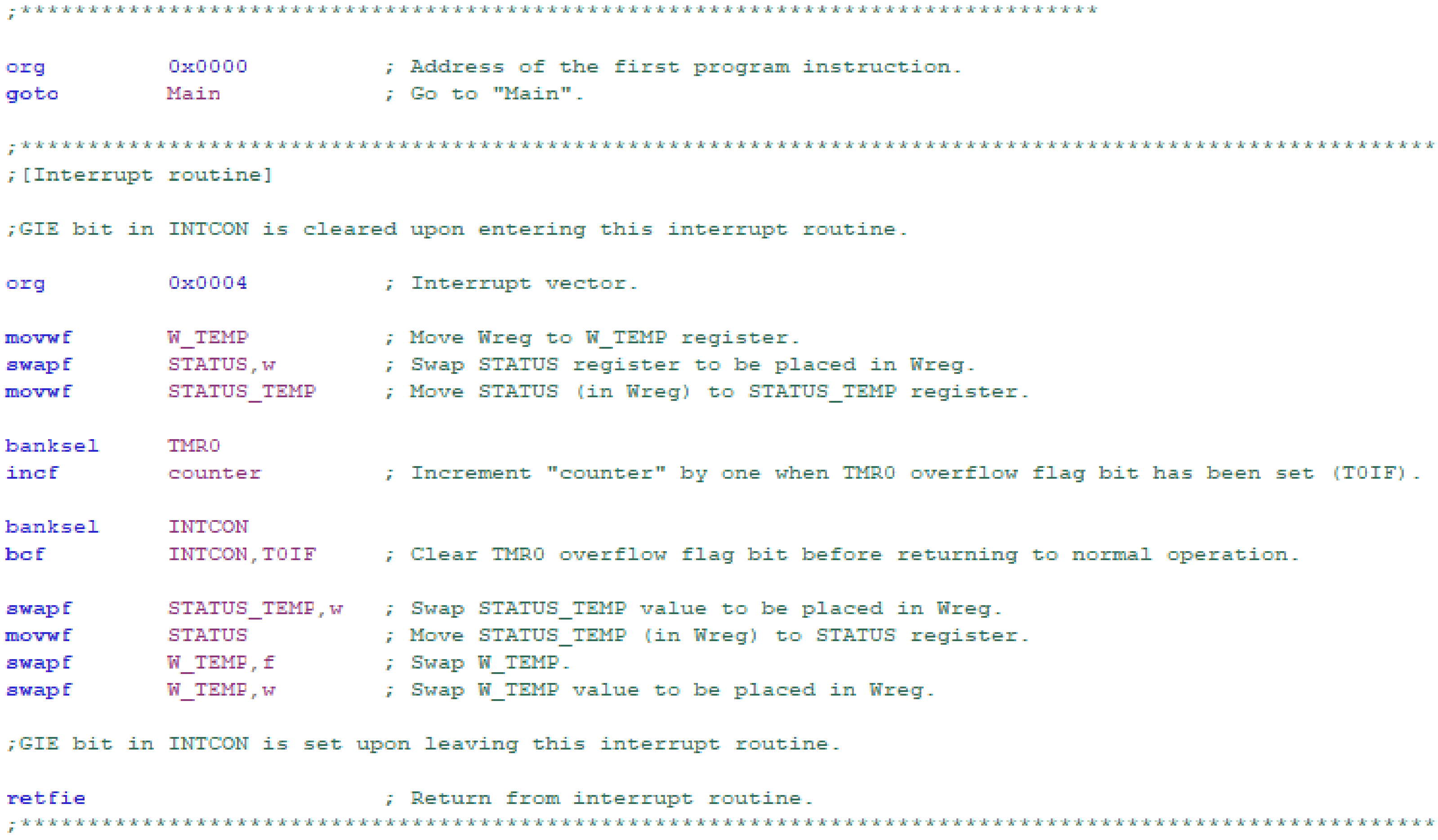


```
;*********************** MAIN PROGRAM ***********************************
main                                        ; Start of the main program
        banksel     OPTION_REG              ; Bank containing register OPTION_REG
        clrf        OPTION_REG
        bcf         OPTION_REG,T0CS         ; TMR0 counts pulses from oscillator
        bcf         OPTION_REG,PSA          ; Prescaler is assign to WDT

        banksel     TMR0
        clrf        TMR0
        clrf        counter

        banksel     INTCON                  ; Bank containing register INTCON

loop
        btfsc       INTCON, TMR0IF          ; pool interrupt flag TMR0IF
        goto        update_ctr              ; TMR0 overflow
        goto        loop                    ; Remain here

update_ctr
        incf        counter                 ; increment counter
        bcf         INTCON, TMR0IF          ; clear interrupt flag TMR0IF
        goto        loop

        end                                 ; End of program
```

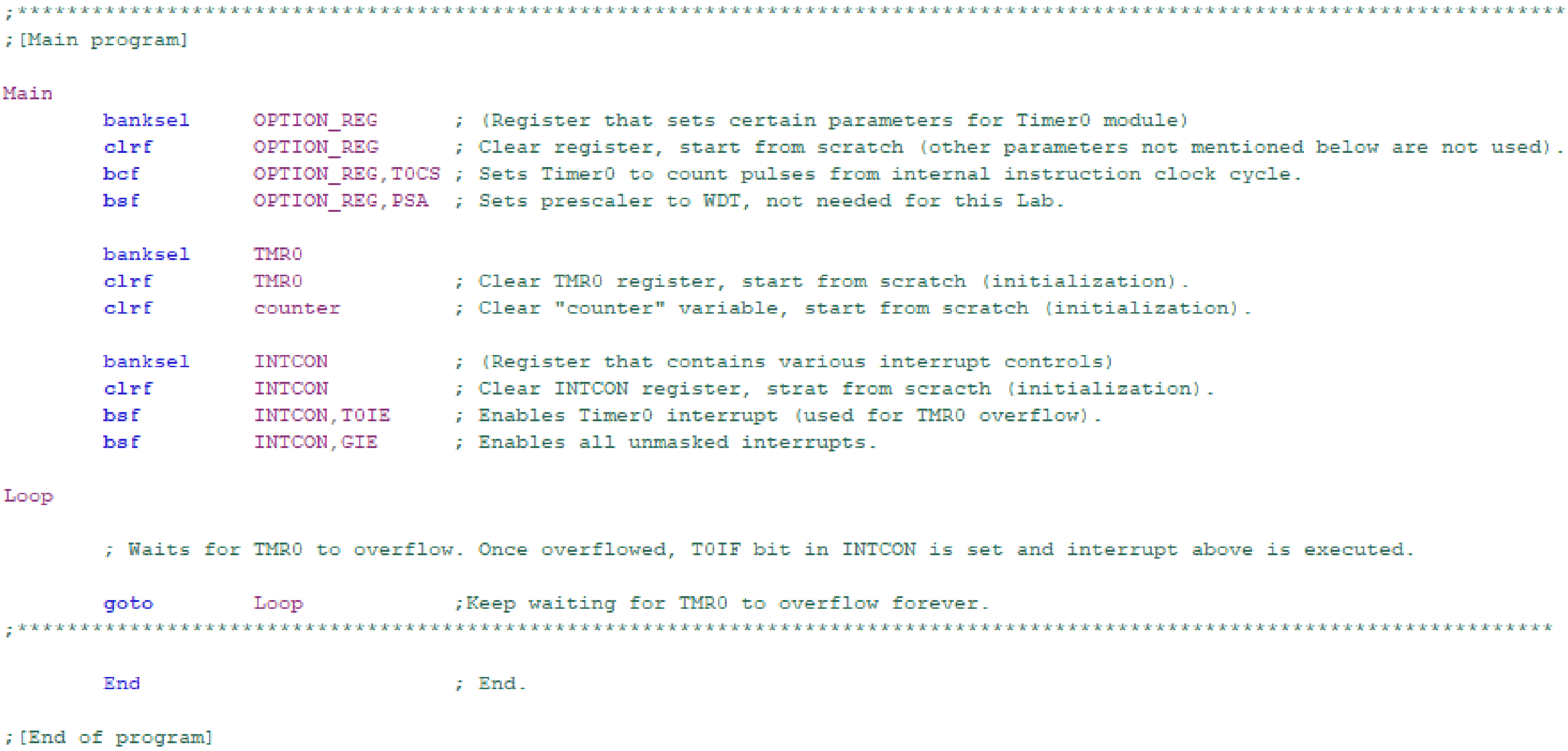
Part B assembly code



```
;***********************************************************************************************
;*Leonardo Fusser (1946995)                                                                    *
;*Microcontroller & Microprocessor systems, Lab6, Day Yann Fong                                *
;*Program that uses interupts to increment a counter when TMR0 overflows.                      *
;***********************************************************************************************


;[PIC config]

#include "p16F887.inc"

; CONFIG1
; __config 0x3FF5
 __CONFIG _CONFIG1, _FOSC_INTRC_CLKOUT & _WDTE_OFF & _PWRTE_OFF & _MCLRE_ON & _CP_OFF & _CPD_OFF & _BOREN_ON & _IESO_OFF & _FCMEN_OFF & _LVP_ON
; CONFIG2
; __config 0x3FFF
 __CONFIG _CONFIG2, _BOR4V_BOR40V & _WRT_OFF


;[Start of program]

        ;*******************************************************************************
        ;[Variable delcarations]

        cblock      0x70            ; (Stores variables below at address 70)
        counter                     ; Counter varibale (used in interrupt routine below).
        STATUS_TEMP                 ; STATUS register variable (for context saving).
        W_TEMP                      ; Wreg variable (for context saving).
        endc
        ;*******************************************************************************
```



```
;***********************************************************************

org         0x0000          ; Address of the first program instruction.
goto        Main            ; Go to "Main".

;****************************************************************************************
;[Interrupt routine]

;GIE bit in INTCON is cleared upon entering this interrupt routine.

org         0x0004          ; Interrupt vector.

movwf       W_TEMP          ; Move Wreg to W_TEMP register.
swapf       STATUS,w        ; Swap STATUS register to be placed in Wreg.
movwf       STATUS_TEMP     ; Move STATUS (in Wreg) to STATUS_TEMP register.

banksel     TMR0
incf        counter         ; Increment "counter" by one when TMR0 overflow flag bit has been set (T0IF).

banksel     INTCON
bcf         INTCON,T0IF     ; Clear TMR0 overflow flag bit before returning to normal operation.

swapf       STATUS_TEMP,w   ; Swap STATUS_TEMP value to be placed in Wreg.
movwf       STATUS          ; Move STATUS_TEMP (in Wreg) to STATUS register.
swapf       W_TEMP,f        ; Swap W_TEMP.
swapf       W_TEMP,w        ; Swap W_TEMP value to be placed in Wreg.

;GIE bit in INTCON is set upon leaving this interrupt routine.

retfie                      ; Return from interrupt routine.
;****************************************************************************************
```



```
;****************************************************************************************************************************
;[Main program]

Main
        banksel     OPTION_REG      ; (Register that sets certain parameters for Timer0 module)
        clrf        OPTION_REG      ; Clear register, start from scratch (other parameters not mentioned below are not used).
        bcf         OPTION_REG,T0CS ; Sets Timer0 to count pulses from internal instruction clock cycle.
        bsf         OPTION_REG,PSA  ; Sets prescaler to WDT, not needed for this Lab.

        banksel     TMR0
        clrf        TMR0            ; Clear TMR0 register, start from scratch (initialization).
        clrf        counter         ; Clear "counter" variable, start from scratch (initialization).

        banksel     INTCON          ; (Register that contains various interrupt controls)
        clrf        INTCON          ; Clear INTCON register, strat from scracth (initialization).
        bsf         INTCON,T0IE     ; Enables Timer0 interrupt (used for TMR0 overflow).
        bsf         INTCON,GIE      ; Enables all unmasked interrupts.

Loop

        ; Waits for TMR0 to overflow. Once overflowed, T0IF bit in INTCON is set and interrupt above is executed.

        goto        Loop            ;Keep waiting for TMR0 to overflow forever.
;***************************************************************************************************************************

        End                         ; End.

;[End of program]
```

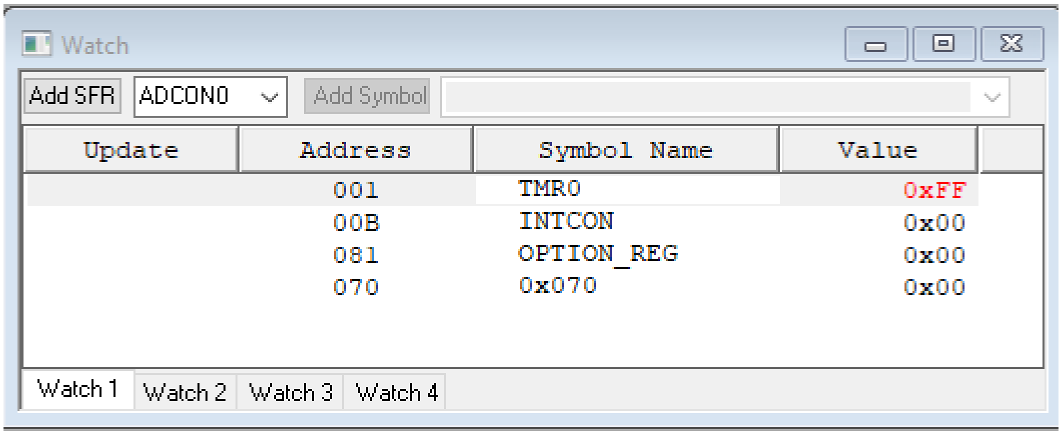
<u>Note</u>: *.asm* source file for Part B is attached to this submission for your convenience

3.0 Observations

Observations from the lab

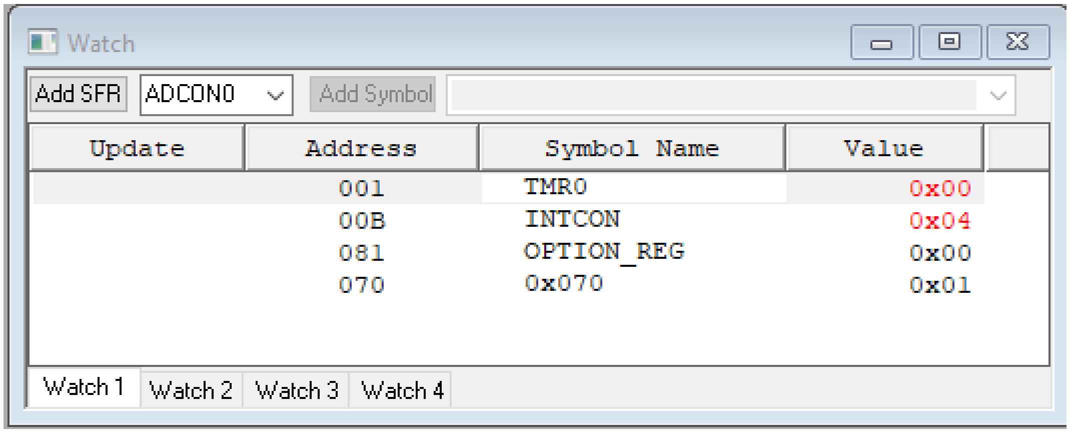
Part A:

- In this part, the loop “loop” keeps checking if the TMR0 register has overflowed. If the TMR0 register hasn’t overflowed (T0IF = 0), the program will keep looping through the loop “loop” until the TMR0 register has overflowed. When the TMR0 register has overflowed (T0IF = 1; set briefly), the program goes to another place (update_ctr) to update the counter by 1, then the TMR0 interrupt flag is reset (T0IF = 0) before returning to the beginning of the loop “loop” to repeat (forever). This is a typical polling mechanism. No interrupts are used for this part. This behavior can be observed from the screenshots below.



Update	Address	Symbol Name	Value
	001	TMR0	0xFF
	00B	INTCON	0x00
	081	OPTION_REG	0x00
	070	0x070	0x00

TMR0 reached 255

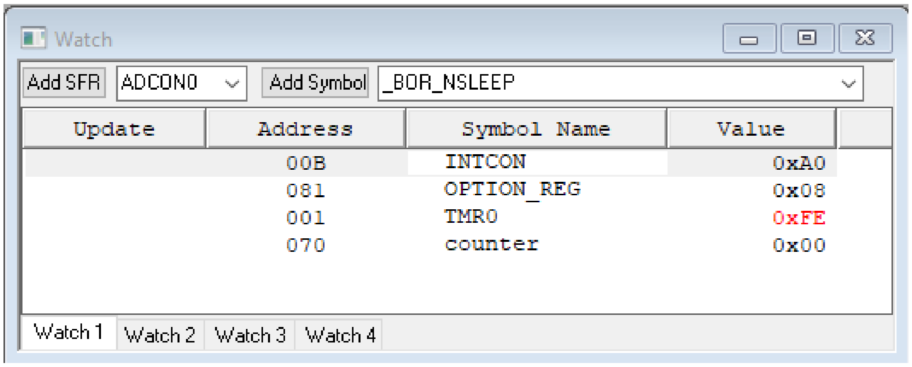


Update	Address	Symbol Name	Value
	001	TMR0	0x00
	00B	INTCON	0x04
	081	OPTION_REG	0x00
	070	0x070	0x01

After overflow, T0IF flag is set briefly and “counter” is incremented by 1

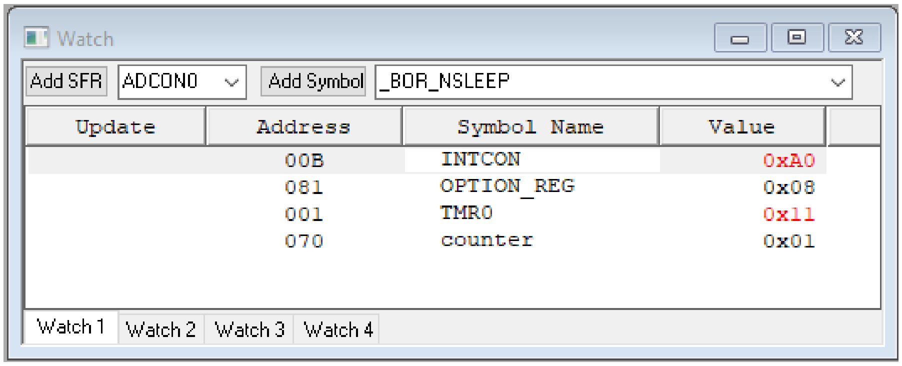
Part B:

- Here, interrupts are used instead of the polling mechanism shown above. In this part, instead of using a loop to keep checking if the TMR0 overflow flag bit has been set (T0IF = 1 set briefly; indicating that TMR0 register has overflowed), the process is done completely in the background by using interrupts. By setting the Timer0 overflow interrupt enable bit (T0IE = 1) in INTCON register, the microcontroller (simulator) will execute an interrupt when it detects that the Timer0 overflow interrupt flag bit is set (T0IF = 1 set briefly; indicating that the TMR0 register has overflowed). Note: no other interrupts can be serviced during this time (GIE = 0; GIE cleared upon entering interrupt). Instead of jumping to a few lines down in code to the incrementation of the counter, the microcontroller (simulator) will perform it while servicing the interrupt. The Timer0 overflow interrupt flag bit is reset (T0IF = 0) after the incrementation of the counter. Once the microcontroller (simulator) is done servicing the interrupt, the program will go back to normal execution and other interrupts can be serviced (GIE = 1). Context saving (STATUS and Wreg) is put in place during the interrupt service routine. This is a typical interrupt mechanism. This behavior can be observed from the screenshots below.



Update	Address	Symbol Name	Value
	00B	INTCON	0xA0
	081	OPTION_REG	0x08
	001	TMR0	0xFE
	070	counter	0x00

TMR0 reached 255



Update	Address	Symbol Name	Value
	00B	INTCON	0xA0
	081	OPTION_REG	0x08
	001	TMR0	0x11
	070	counter	0x01

After overflow, T0IF flag is set briefly and “counter” is incremented by 1

4.0 CONCLUSION

- Purpose of this lab has been achieved.
- Understood differences/operation between polling and interrupt mechanism.
- Understood how to use MPLAB IDE.
- Understood how to use MPLAB IDE simulator.
- Understood how to view TMR0 in real time (animation) using MPLAB IDE simulator.
- Problem: (for Part B) one of the variable declarations was misspelled (STAUS instead of STATUS), creating syntax errors upon assembling code.
- Solution: corrected syntax mistake in variable declarations in code. Code assembled afterwards and operation worked as intended (like in Part A but using interrupts instead).