**MPLAB Introduction and PIC24 Assembly Language**

**4th Laboratory Report for ECE 383**

**Microcomputers**

**Submitted by**

**Yichen Huang**

**11906882**

**Shomari Thomas**

**11672867**

**The University of Alabama**

**Tuscaloosa, Alabama 35487**

**February 12, 2020**

**Abstract**

Lab 4 was an introduction to basic PIC24 assembly language and a means of practice using the MPLAB Integrated Development Environment (IDE). During this lab we used MPLAB to simulate the PIC24 assembly language program in a project, as well as implemented simple programming tasks using PIC24 assembly language. These tasks were functional practice in becoming more familiar with the MPLAB environment.

For Task 1, we followed the step-by-step instructions provided in the lab document to activate the MPLAB Simulator and watch variable values, special function register values, special function register, memory locations, and window locations. For Task 2, we watched variables aa, bb, lsp, msp, and sum and their corresponding memory locations when space is reserved for variable sum to hold the sum of lsp and msp. For task 3, we wrote an assembly language program that implemented the C program provided in the lab document monitored the memory locations corresponding to variables i, j, k, l, m, xx, and yy. Similarly to task 3, task 4 consisted of we writing an assembly language program that implemented another C program provided in the lab document and monitoring the memory locations corresponding to

variables u16\_x, u8\_a, u8\_b, u8\_c, u8\_d, u8\_e, and u8\_f.

In Lab 4, we became familiar with the MPLAB environment by translating C programs into program assembly language, and completing simulations of these programs that show the relationships between corresponding changes in data location and data memory for variables.

**Introduction**

In Lab 4, we were introduced to the basic PIC24 assembly language and the MPLAB Integrated Development Environment (IDE). The PIC24 assembly language was introduced by translating each line of a C program into the corresponding assembly language program. The MPLAB Integrated Development Environment (IDE) was used as a helpful tool to simulate the assembly language program and provide the results in the form of registers and values. Additionally an assembly language program was downloaded onto a PIC24 device, giving a physical example of the capabilities of the assembly language program and their application in the appropriate environments. This lab exemplified the sensitivity of simulation to different values (hexadecimal in this case), and showed how we can use the PIC24 assembly language and MPLAB Integrated Development Environment together to solve problems.

**Pre-Lab**

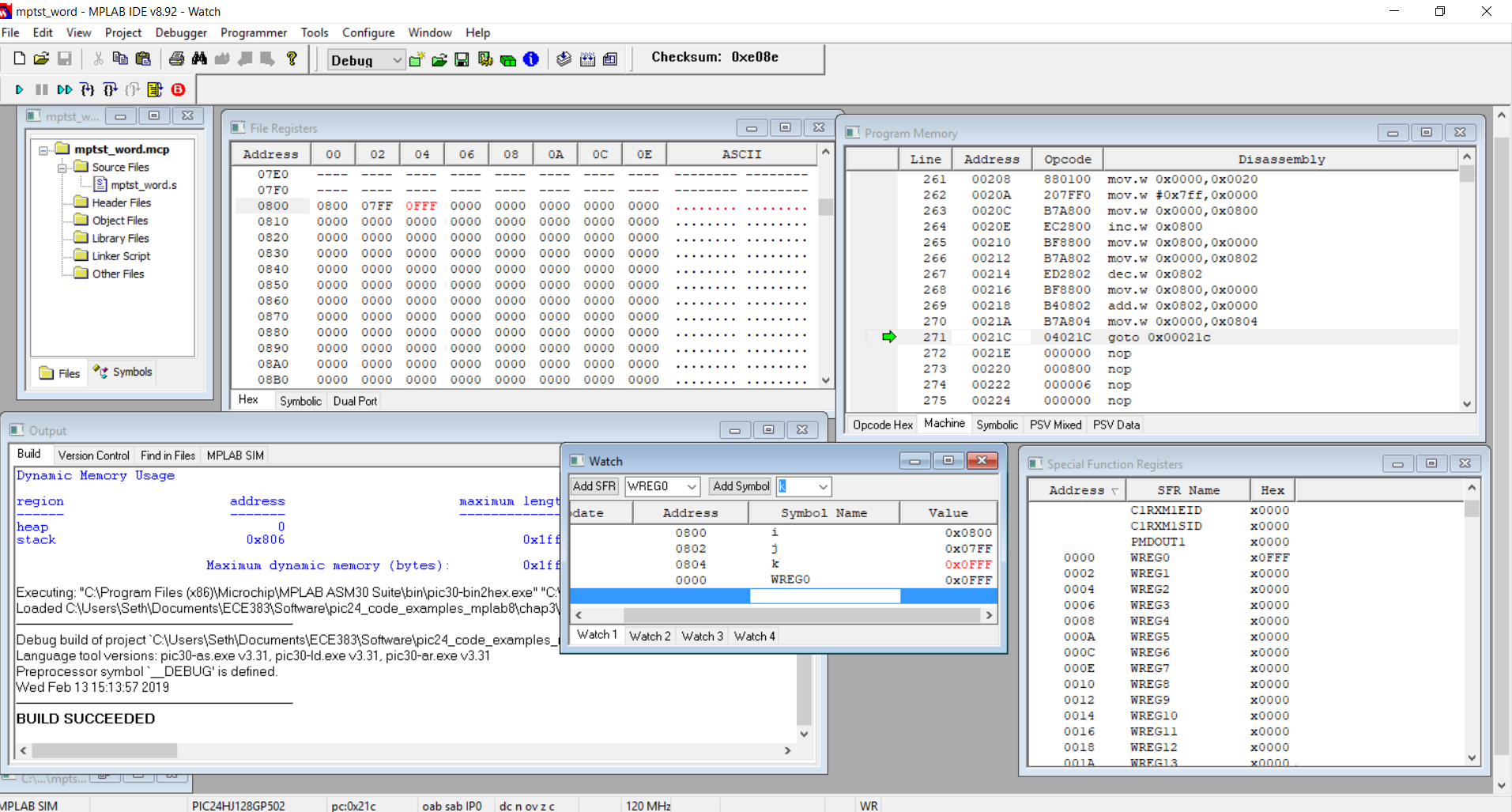
**Task 1- MPLAB Introduction**

For Task 1 of Lab 4, we first moved the files in C:\microchip\chap3\ to our custom directory. In the MPLAB IDE we opened the “mpst\_word.mcp” project and selected the PIC24HJ128GP502 device. After assembling the project, we scrolled through the program memory window to find our program in memory. Then, we opened the file registers window to view the data memory where our variables are listed, and the special function registers window. Lastly, we opened the watch window, and after adding the SFR symbol we were able to watch variable values and special function register values of the i, j, k variables and the W0 special function register. We used this information to apply it to the MPLAB Simulator. In the simulator, we were able to watch and correlate the changing values of the memory and watch window locations with the instructions causing the changes. Figure (1) shows our source assembly

language program. Pictured below in Figure (2) is a screenshot of the final state of the program, data memory, and the watch window.

**Figure (1) - APPENDIX**

**Figure (2)**



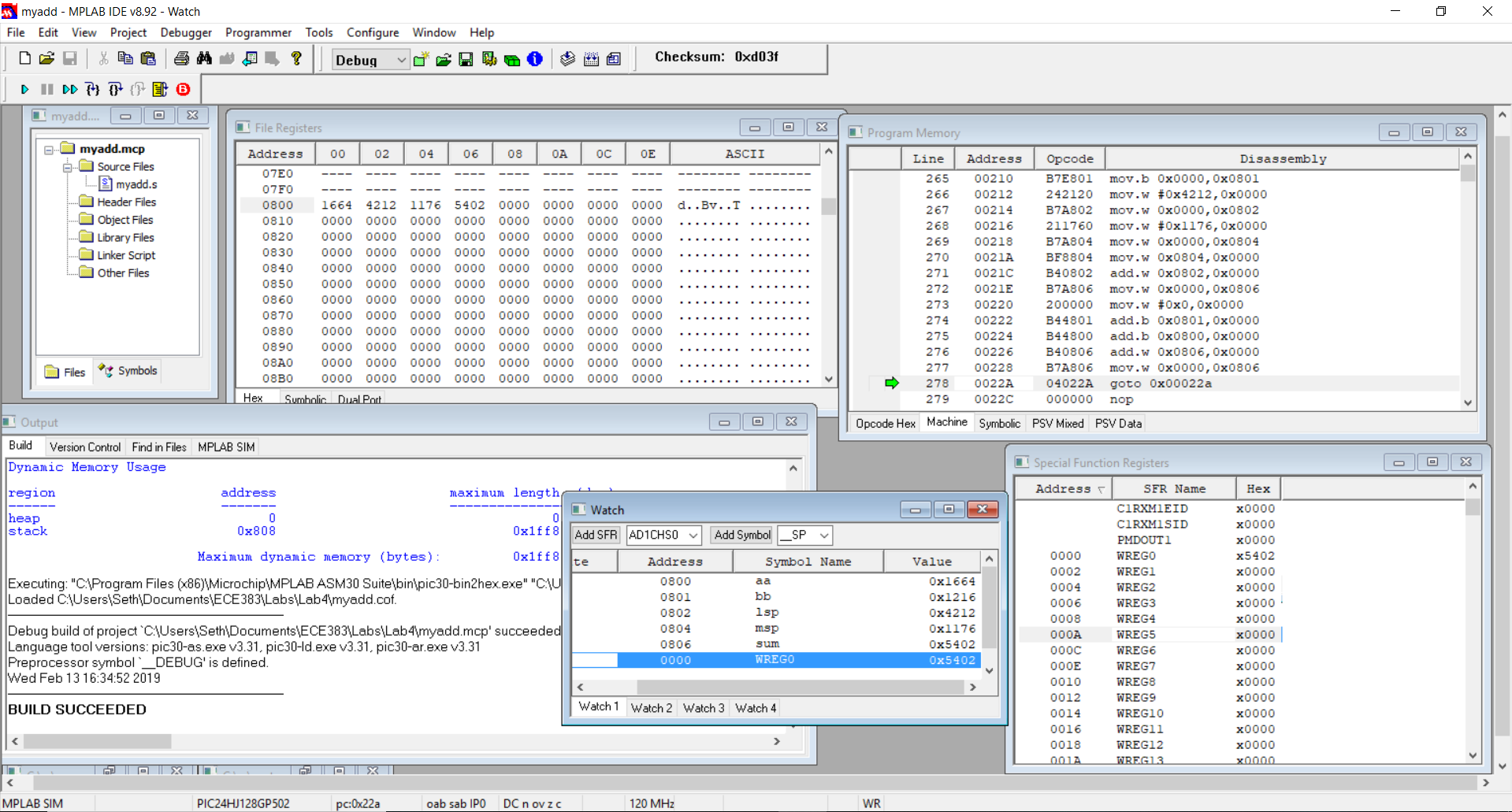
Additionally, we changed the avalue equate to be the last four digits of your student ID, 6882. The program was assembled and simulated once again to represent this.

Task 2 – myadd.s

For Task 2 of Lab 4, we used a given C program to execute assembly instructions that created changes in data memory and memory locations of variables. After removing all instructions from mov #avalue, W0 through mov WREG,k, we started using the myadd.s file as a start for a new program. Next, we converted the number 11906882 into an eight digit hexadecimal number. Using the C code provided in the lab document, we wrote a program to add the four digit hex number formed by the last four digits of the student ID number (6882) to the four-digit hex number formed by the first four digits of the student ID number (1190). We did so by translating the given C program into the appropriate assembly instructions line-by-line, reserving space for the lsp and msp variable to hold the hex values. Lastly, we opened the watch window, and watched the variable values of aa, bb, lsp, msp, and sum. As in task 1, we used the resulting information to apply it to the MPLAB Simulator. In the simulator, we were able to watch and correlate the changing values of the memory and watch window locations with the instructions causing the changes. Figure (3) shows our source assembly language program. Pictured below in Figure (4) is a screenshot of the final state of the program, data memory, and the watch window.

**Figure (3) - APPENDIX**

**Figure (4)**

****

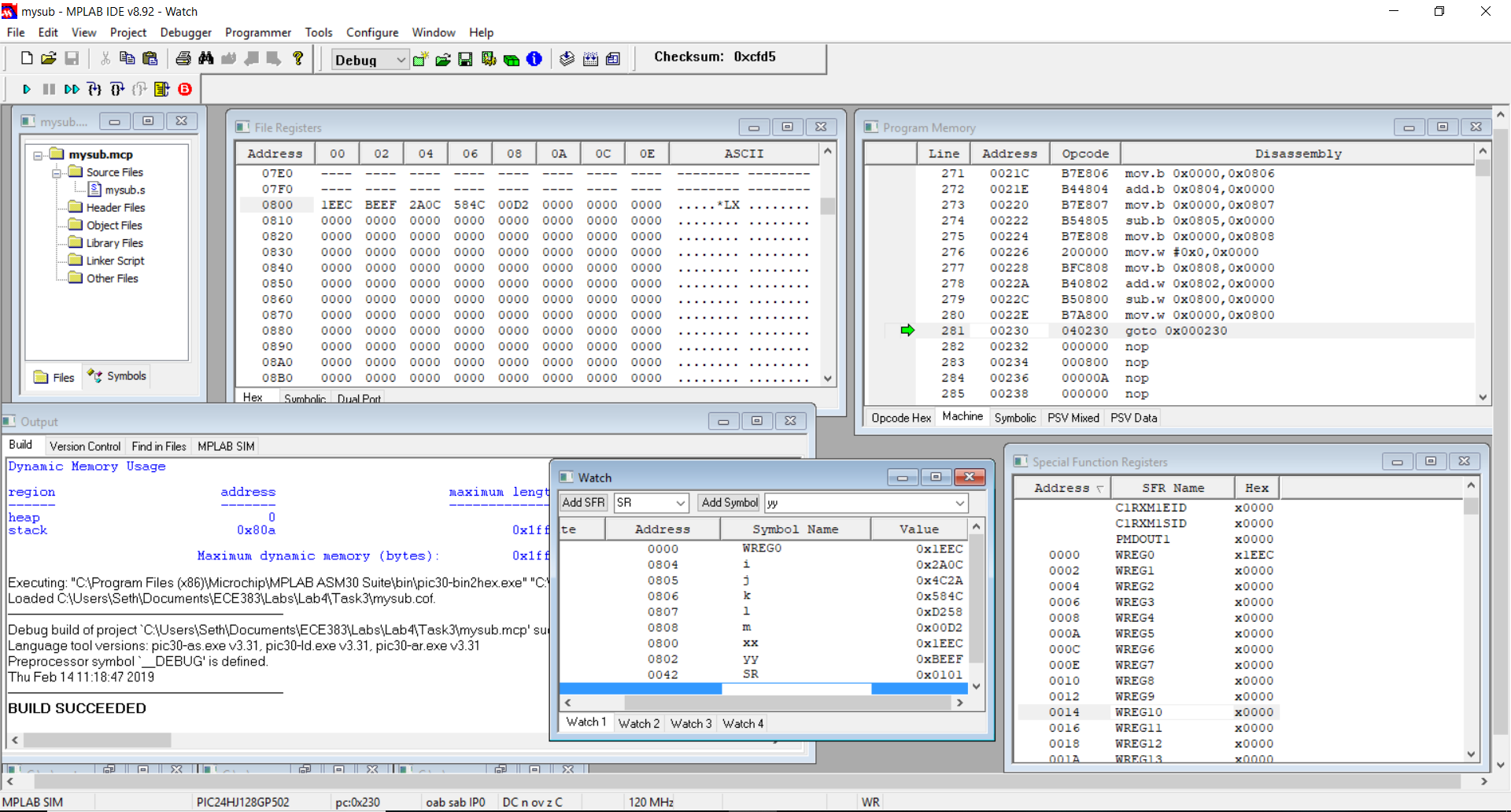
**Procedure/Results**

Task 3 – mysub.s

For task 3, we created a new project named mysub. We then wrote an assembly language program corresponding to the C program provided in the lab document, and used the last 6 digits of the student ID number (906882). Next we opened the watch window and watched the variable values of i, j, k, l, m, xx, and yy, with i, j, k, l, m being 8-bit variables and xx and yy being 16-bit variables. As in the previous tasks, we used the resulting information to apply it to the MPLAB Simulator. In the simulator, we were able to watch and correlate the changing values of the memory and watch window locations with the instructions causing the changes. Figure (5) shows our source assembly language program. Pictured below in Figure (6) is a screenshot of the final state of the program, data memory, and the watch window.

**Figure (5) - APPENDIX**

**Figure (6)**

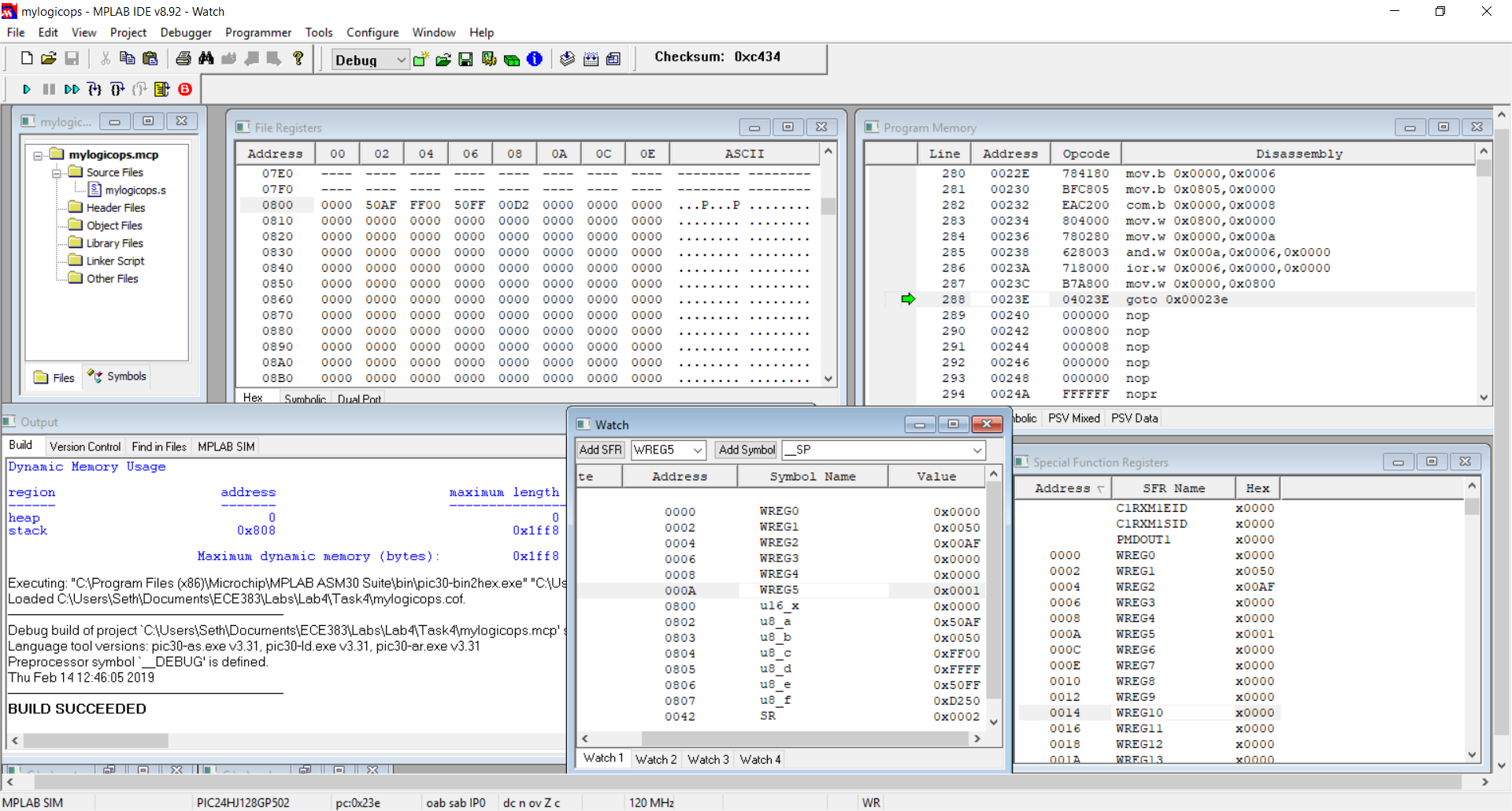
****

Task 4 – mylogicops.s

In the 4th and final task of lab 4, we created a new project named mylogicops. We then wrote an assembly language program corresponding to the C program provided in the lab document. Next we opened the watch window and watched the variable values of u16\_x, u8\_a, u8\_b, u8\_c, u8\_d, u8\_e, and u8\_f . We then used the MPLAB Simulator to simulate our program. In the simulator, we were able to watch and correlate the changing values of the memory and watch window locations with the instructions causing the changes. Following our simulation, we downloaded our program onto the PIC24 device and showed our demonstration to the TA. Figure (7) shows our source assembly language program. Pictured below in Figure (8) is a screenshot of the final state of the program, data memory, and the watch window.

**Figure (7) - APPENDIX**

**Figure (8)**

****

**Conclusion**

We are now confident in our ability to translate C code into its corresponding assembly language

program and complete simulations and device downloads using MPLAB. In addition, we were successful in our simulations for tasks 1, 2, 3, and 4, as well as the program download onto the PIC24 device. The results of simulations provided the expected flag values and memory locations of the values. This lab provided an introduction to the PIC24 assembly language and MPLAB and allowed for us to be confident in their use for testing components in future labs.

**Appendix**

**Task 1: Figure (1)**

;  
; Just check out MPLAB  
 .include "p24Hxxxx.inc"  
 .global \_\_reset ;The label for the first line of code.   
 .bss ;unitialized data section  
;;These start at location 0x0800 because 0-0x07FF reserved for SFRs  
i: .space 2 ;Allocating space (in bytes) to variable.  
j: .space 2 ;Allocating space (in bytes) to variable.  
k: .space 2 ;Allocating space (in bytes) to variable.  
;.........................................................................  
;Code Section in Program Memory  
;.........................................................................  
 .text ;Start of Code section  
\_\_reset: ; first instruction located at \_\_reset label  
 mov #\_\_SP\_init, w15 ;Initalize the Stack Pointer  
 mov #\_\_SPLIM\_init,W0   
 mov W0, SPLIM ;Initialize the stack limit register  
;\_\_SP\_init set by linker to be after allocated data   
;User Code starts here.  
; C Program equivalent  
; #define avalue 2047  
; uint16\_t i,j,k;  
;  
; i = avalue; /\* myvalue = 2047 (0x7FF) \*/  
; i = i + 1; /\* i++, i = 2048 (0x800) \*/  
; j = i; /\* j is 2048 (0x0800) \*/  
; j = j - 1; /\* j--, j is 2047 \*/  
; k = j + i; /\* k = 4095 (0x0FFF) \*/  
;  
 .equ avalue, 2047  
;i = avalue; /\* myvalue = 2047 \*/  
 mov #avalue, w0 ; w0 = 2047 (w0 is wreg)  
 mov wreg,i ; i = 2047  
; i = i + 1;  
 inc i ; i = i + 1  
; j = i  
 mov i,wreg ; w0 = i  
 mov wreg,j ; j = w0  
; j = j - 1; /\* j--, j is 100 \*/  
 dec j ; j= j - 1  
; k = j + i  
 mov i,wreg ; w0 = i  
 add j,wreg ; w0 = w0+j   
 mov wreg,k ; k = 0  
  
done:  
 goto done ;Place holder for last line of executed code  
.end ;End of program code in this file

**Task 2: Figure (3)**

;  
; Just check out MPLAB  
 .include "p24Hxxxx.inc"  
 .global \_\_reset ;The label for the first line of code.   
 .bss ;unitialized data section  
;;These start at location 0x0800 because 0-0x07FF reserved for SFRs  
aa: .space 1 ;Allocating space (in bytes) to variable.  
bb: .space 1 ;Allocating space (in bytes) to variable.  
lsp: .space 2 ;Allocating space (in bytes) to variable.  
msp: .space 2 ;Allocating space (in bytes) to variable.  
sum: .space 2 ;Allocating space (in bytes) to variable.  
;.........................................................................  
;Code Section in Program Memory  
;.........................................................................  
 .text ;Start of Code section  
\_\_reset: ; first instruction located at \_\_reset label  
 mov #\_\_SP\_init, w15 ;Initalize the Stack Pointer  
 mov #\_\_SPLIM\_init,W0   
 mov W0, SPLIM ;Initialize the stack limit register  
;\_\_SP\_init set by linker to be after allocated data   
  
;User Code starts here.  
; C Program equivalent  
; #define avalue 4212  
;uint8 aa=100, bb=22;  
;uint16 lsp, msp, sum;  
;lsp = 0xY3Y2Y1Y0; // Four digits of CWID treated as hex  
;msp = 0xY7Y6Y5Y4; // Four digits of CWID treated as hex  
;sum = lsp + msp;  
;sum = sum + aa + bb;  
 .equ avalue, 4212  
;uint8 aa=100, bb=22;  
mov #100, w0 ; w0 = 100  
mov.b wreg, aa ; a = 100  
  
mov #22, w0 ; w0 = 22  
mov.b wreg, bb ; b = 22  
  
;lsp = 0xY3Y2Y1Y0; // Four digits of CWID treated as hex  
mov #0x4212, w0 ; w0 = 0x4212  
mov wreg, lsp ; lsp = 0x4212  
  
;msp = 0xY7Y6Y5Y4; // Four digits of CWID treated as hex  
mov #0x1176, w0 ; w0 = 0x1176  
mov wreg, msp ; msp = 0x1176  
  
;sum = lsp + msp;  
mov msp, wreg ; w0 = msp  
add lsp, wreg ; w0 = w0 + lsp  
mov wreg, sum ; sum = w0  
  
;sum = sum + aa + bb;  
mov #0x0000, w0 ; w0 = 0x0000 (To clear from last use)  
add.b bb, wreg ; w0 = w0 + bb (since w0 is still = sum)  
add.b aa, wreg ; w0 = w0 + aa  
add sum, wreg ; w0 = sum + w0  
mov wreg, sum ; sum = w0

done:  
 goto done ;Place holder for last line of executed code  
.end ;End of program code in this file

**Task 3: Figure (5)**

;

; Just check out MPLAB

.include "p24Hxxxx.inc"

.global \_\_reset ;The label for the first line of code.

.bss ;unitialized data section

;;These start at location 0x0800 because 0-0x07FF reserved for SFRs

xx: .space 2 ;Allocating space (in bytes) to variable.

yy: .space 2 ;Allocating space (in bytes) to variable.

i : .space 1 ;Allocating space (in bytes) to variable.

j : .space 1

k : .space 1

l : .space 1

m : .space 1

;.........................................................................;Code Section in Program Memory

;.........................................................................

.text ;Start of Code section

\_\_reset: ;first instruction located at \_\_reset label

mov #\_\_SP\_init, w15 ;Initalize the Stack Pointer

mov #\_\_SPLIM\_init,W0

mov W0, SPLIM ;Initialize the stack limit register

;\_\_SP\_init set by linker to be after allocated data

;User Code starts here.

; C Program equivalent

; uint16 xx=0xDEAD, yy=0xBEEF;

; uint8 i, j, k, l, m;

; i = Y1Y0;

; j = Y3Y2;

; k = Y5Y4;

; l = i + k;

; m = j ñ l;

; xx=xx-yy-m;

;

; uint16 xx=0xDEAD, yy=0xBEEF;

mov #0xDEAD, w0

mov w0, xx

mov #0xBEEF, w0

mov w0, yy

; i = Y1Y0;

mov.b #38, w0

mov.b wreg, i

; j = Y3Y2;

mov.b #26, w0

mov.b wreg, j

; k = Y5Y4;

mov.b #66, w0

mov.b wreg, k

; l = i + k;

add.b i, wreg

mov.b wreg, l

; m = j ñ l;

sub.b j, wreg

mov.b wreg, m

; xx=xx-yy-m;

mov yy,w0

sub xx

mov.b m, wreg

sub.b xx

done:

goto done ;Place holder for last line of executed code

.end ;End of program code in this file

**Task 4: Figure (7)**

;

; Just check out MPLAB

.include "p24Hxxxx.inc"

.global \_\_reset ;The label for the first line of code.

.bss ;unitialized data section

;;These start at location 0x0800 because 0-0x07FF reserved for SFRs

u16\_x: .space 2 ;Allocating space (in bytes) to variable.

u8\_a : .space 1 ;Allocating space (in bytes) to variable.

u8\_b : .space 1 ;Allocating space (in bytes) to variable.

u8\_c : .space 1

u8\_d : .space 1

u8\_e : .space 1

u8\_f : .space 1

;.........................................................................

;Code Section in Program Memory

;.........................................................................

.text ;Start of Code section

\_\_reset: ;first instruction located at \_\_reset label

mov #\_\_SP\_init, w15 ;Initalize the Stack Pointer

mov #\_\_SPLIM\_init,W0

mov W0, SPLIM ;Initialize the stack limit register

;\_\_SP\_init set by linker to be after allocated data

;User Code starts here.

; C Program equivalent

; uint8 u8\_a, u8\_b, u8\_c, u8\_d, u8\_e, u8\_f;

; uint16 u16\_x=0x0001;

; u8\_a=0xAF;

; u8\_b=0x50;

; u8\_c= u8\_a & u8\_b;

; u8\_d= u8\_a | u8\_b;

; u8\_e= u8\_a ^ u8\_b;

; u8\_f=~u8\_a;

; u16\_x=~u8\_d | (u16\_x & u8\_c);

;

; u16\_x=0x0001;

mov #0x1, w0

mov w0, u16\_x

; u8\_a=0xAF;

mov.b #0xAF, w0

mov.b wreg, u8\_a

; u8\_b=0x50;

mov.b #0x50, w0

mov.b wreg, u8\_b

; u8\_c= u8\_a & u8\_b;

mov.b wreg, u8\_c

mov.b u8\_a, wreg

and.b u8\_c

; u8\_d= u8\_a | u8\_b

mov.b wreg, u8\_d

mov.b u8\_b, wreg

ior.b u8\_d

; u8\_e= u8\_a ^ u8\_b;

mov.b wreg, u8\_e

mov.b u8\_a, wreg

xor.b u8\_e

; u8\_f=~u8\_a;

com.b u8\_a, wreg

mov.b wreg, u8\_f

; u16\_x=~u8\_d | (u16\_x & u8\_c);

mov u16\_x, w0

and.b u8\_c, wreg

mov w0, u16\_x

mov.b u8\_d, wreg

com.b w0,w0

ior.b u16\_x, wreg

;

Done:

goto done ;Place holder for last line of executed code

.end ;End of program code in this file

