Laboratory Assignment #1

Autumn 2013 TCES 430 – Microprocessor System Design

by

Alvin Baldemeca and Edward Bassan

Date: 10/15/2013

Table of Contents

I. Objective

The objective of this lab is to get familiar with the MPLAB/MPLABX IDE, in creating and debugging a project. We also aim to get some experience writing assembly code for the PIC 18f4520. Finally we want our assembly code to do the following requirements for Lab 1:

Write assembly code for the PIC18F4520 microprocessor; and the code you write should carry out the following calculation.

Consider three positive integers A, B, and C where both A and B are fixed and satisfy B < A. The third integer C is initially a random value so that C < B. As C increments by one through the range from B to A, you want to find the sum of all the values of C when B ≤ C ≤ A.

Your code should assign values to A and B (fixed) and C (initially < B), and then find the sum S. Make all values two-byte unsigned integers.

Be efficient in execution time, not necessarily in total program lines.

II. Procedure

For this lab the first thing we did is to create a new project. We chose a Microchip Embedded - Standalone Project. The device we selected is the PIC18f4520, we choose the simulator to compile using the mpasm to assemble our code. We copied the template for the micro-controller that we have and coded our program. The final code is included in this report under Appendix A.

We ran the program in the IDE under debug and checked for the execution time for one loop while C < B and for one loop while B < C < A. This was done by setting breakpoints and the stopwatch. We also selected the oscillator frequency to be 20 MHz according to the lab instructions. Note: In MPLABX this is under the menu “Run > Set Project Configuration > default “, then choosing the simulator and changing the frequency.

III. Data & Analysis

We measured the execution time for one loop while C < B to be 7μs and 13μs for one loop while B < C < A. We also observed that most instructions took 1μs to execute. The whole program took 402μs to complete. Note: This is using the default values assigned and could be different if the values to A, B and C were changed.

From the problem we face we develop an algorithm in a high level language like C or pseudo code that does the following:

```
;   while(True)
;   {
;      if(myC == B)
```

```
;         break;
;     else
;         myC++;
;  }
;
;  while(True)
;  {
;      myS = myS + myC;
;      if(myC == myA)
;          break;
;      else
;          myC++;
;  }
```

We are able to do this because of the knowledge that we know that once myC==myB we can start adding myC to the sum myS. This code is similar to the following below:

```
;  while(C < B)
;  {
;         myC++;
;  }
;
;  while(A>=C)
;  {
;      myS = myS + myC;
;      myC++;
;  }
```

However the first code is much closer to the way the assembly code is written. The check if A == C or A!=C is more straight forward in assembly compared to A >=C. And while the second code seems cleaner the implementation of the first one in assembly is more efficient in terms of speed(i.e. lines of assembly code).

There is a relationship between lines of assembly code to the amount of time it takes a program to run because most program execute in one instruction cycle. There are few exceptions to this like instructions that are 4 byte and also some branch instructions. We could have easily done the following:

```
while(A >=C )
{
        C++;
        if(C>=B)
```

```
        S = S + C;
}
```

which is also equivalent but implementing this in assembly produces more lines in assembly code compared with the first. Appendix B contains a flow chart which compares these 3 implementation.

We often write assembly code so that we can optimize our logic the way we want to instead of relying on a compiler to optimize our code for us. Obviously this is one advantage of writing in such a low level but we really have to test that our program behaves the way we intended. A better question to ask is how a compiler would have written an assembly code for the three codes we highlighted compared to the assembly code that we had written. We did not attempt to do this for this lab although it is a good exercise that we might attempt in the future.

To debug our code we relied on the debug tool for the MPLAB/MPLABX and watch the WREG (write register) and the STATUS (status register) to verify that our program is behaving correctly. We also verified the contents of the registers we used.

IV. Observation & Conclusion

The learning curve in using the MPLAB/MPLABX IDE is not that steep, so its fairly easy to set up a new project and debug. A few challenges we face is getting the template for the particular micro-controller that we are using, and some of the debugging utilities like getting the register to display gave some trouble. Writing in assembly is fairly difficult or rather time consuming even for a simple program. We got our assembly program to behave the way we intended. The total program took 402μs to complete with the default values given. The loop C < B and C < A took 7μs and 13μs respectively for one iteration. We satisfied the objective we set up for this lab.

Appendix A : Code for Lab1.asm

```
; Alvin Baldemeca & Edward Bassan
; TCES 430 10/10/2013
; UWT, Prof. Sheng
;
; Lab 1
; For this lab we were asked to write an assembly code for the PIC18f4520 to do
; the following:
;
; Consider three positive integers A, B, and C where both A and B are fixed and
; satisfy B < A. The third integer C is initially a random value so that C < B.
; As C increments by one through the range from B to A, you want to find the sum
; of all the values of C when B ? C ? A.
; Your code should assign values to A and B (fixed) and C (initially < B), and
; then find the sum S. Make all values two-byte unsigned integers.
;
; Expected Result
; We expect the these values to be in the register when the program finishes
; Address   000   001   002   003   004   005   006   007
; Values     25    01    15          01     25    01    ED    12

#include <p18f4520.inc>             ;#include derective includes additional code. This
                    ; is a header file which defines configurations,
                    ;registers, and other useful bits of information for
                    ;the PIC18F4520 microcontroller

#define F 1
;first we want to use a macro to hold 4 variables myA myB myC and myS
 CBLOCK 0x000
  myA:2, myB:2, myC:2, myS:2
 ENDC

;We can change the random number that we picked in this block of code.  These are
;two bytes unsinged ints little endian so A is LSB and A1 is MSB same for B, C and S
VAL_OF_A equ H'25'
VAL_OF_A1 equ H'1'
VAL_OF_B equ H'15'
VAL_OF_B1 equ H'1'
VAL_OF_C equ H'FE'
VAL_OF_S equ D'0'
;The ORG directive is used to set the program or register address.  In this case
```

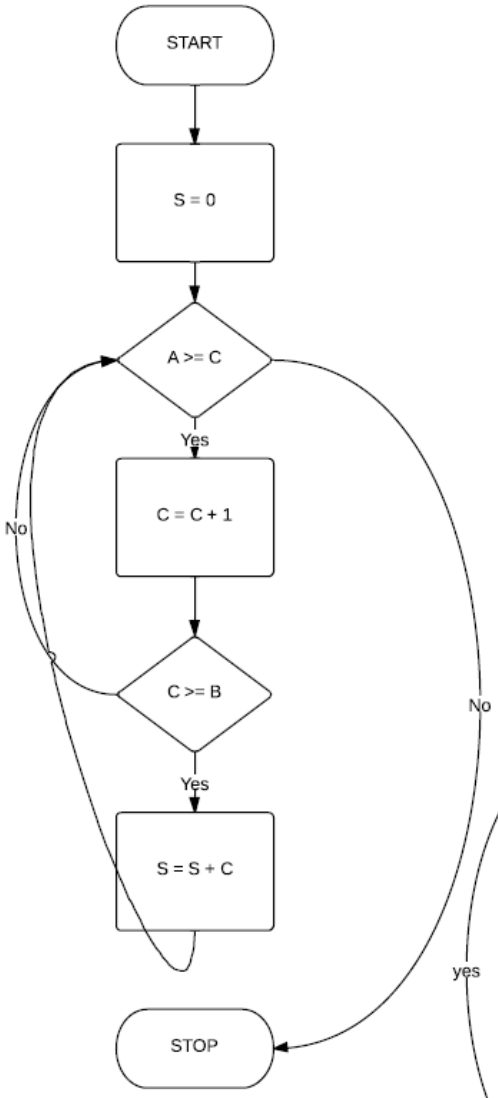
```
;we put the command goto Main in address 0
        ORG 0
        goto Main    ;point the reset vector to the start of our program

;We assemble the following program in the memory location 0x0200
        org 0x0200

; Since this program is written in assembly it is coded for efficency and not
; size.  Extra instructions are taken out.  For example to check if C >= B, we
; can only check that C == B since we know that initially C < B and C increases
; in value.  Another example is since C is two bytes, we do not always have to
; check both bytes of B since if the LSB of both aren't equal then we know that
; we do not need to check the MSB.
;
; This program performs the following C program
;
; void main()
; {
;   unsinged int myA = 0x125;
;   unsinged int myB = 0x115;
;   unsinged int myC = 0xFE;
;   unsinged int myS = 0;
;
;   while(True)
;   {
;      if(myC == B)
;         break;
;      else
;         myC++;
;   }
;
;   while(True)
;   {
;       myS = myS + myC;
;       if(myC == myA)
;          break;
;       else
;          myC++;
;   }
; Note: This is with the knowledge that the initial conditions are myB < myA and
;       myC < myB.
Main
;We initialize our unsingned ints myA, myB, myC
```

```
    movlw VAL_OF_A          ; move value of A to WREG (working rigister)
    movwf myA, A            ; move from WREG to myA
    movlw VAL_OF_A1         ; move the MSB byte block to the WREG
    movwf myA + 1, A        ; move the contents of the WREG to the MSB of myA
    movlw VAL_OF_B
    movwf myB, A
    movlw VAL_OF_B1
    movwf myB + 1, A
    movlw VAL_OF_C
    movwf myC, A
; We could probably use clear to initialize these register to zero
    movlw VAL_OF_S
    movwf myC + 1, A
    movwf myS, A
    movwf myS + 1, A

; Increment myC and test if it is equal to myB if this condition is true then we
; can start adding C to the running sum to be accumulated.
WHILE_C_NOT_EQ_B
    infsnz myC, F       ; C = C + 1
    incf myC+1, F

    ;bc ADD_HI_C     ; If there is a carry bit add this to MSB byte of myC


;DONE_ADD_C          ; A lable to return to from ADD_HI_C
    movf myC, W     ; move C into the  working register Test if(C == B)
    subwf myB,W     ; B - C -> WREG
    bz SUB_HI_B     ; if myC == myB we need to check myC+1 == myB+1
    goto WHILE_C_NOT_EQ_B  ; Continue to increment C

; The block below checks if myC(MSB) == myB(MSB)
SUB_HI_B
    movf myC+1,W    ; move myC(MSB) to the write register
    subwf myB+1,W   ; subtract myC+1 from myB+1 or myB(MSB) - myC(MSB)
    bz ADD_C_TO_SUM    ; Branch if C == B, since we know what C < B initialy

    bra WHILE_C_NOT_EQ_B  ; we know that C will eventually equal B
                    ; else C != B return to the loop


; In this block of code we know that C => B so we can start summing C
ADD_C_TO_SUM
```

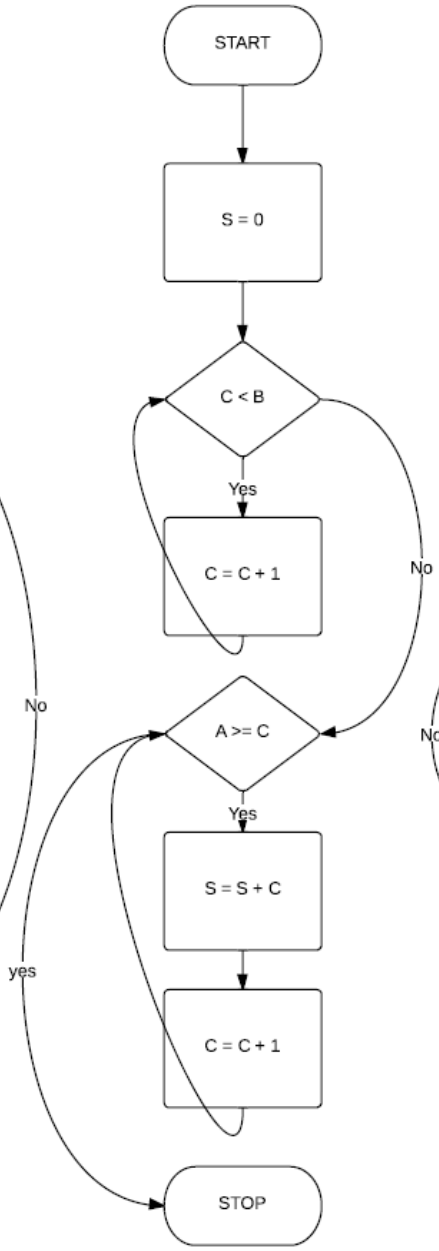
```
    movf myS, W    ;
    addwf myC,W    ;
    movwf myS, A      ; myS = myS + myC(LSB)

    movf myS+1,W
    addwfc myC+1,W
    movwf myS+1     ;myS = myS + C + carry bit (MSB)

;Now we need to test if C == A
    movf myC, W     ; move C into the  working register Test if(C== B)
    subwf myA,W     ; myA - myC (LSB)
    bz SUB_HI_A     ; If the answer is zero we need to subtract the MSB
A_GREATER_C         ; If we get to this point we know that A > C
    infsnz myC, F       ; C = C + 1
    incf myC+1, F
    goto ADD_C_TO_SUM  ; Loop until C == A

;This block substracts the MSB of myA and myC
SUB_HI_A
    movf myA+1,W
    subfwb myC+1,W
    bz FINISH
    bra A_GREATER_C

FINISH
halt
          end
```

Appendix B : Flow Charts

Slow



Faster



Fastest