ECE 2504 Fall 2003 Lab 3: 16-bit Arithmetic

Project Objective
Design, code, debug, and test an assembly language subroutine that sums an array of 16-bit two’s complement format binary numbers. The intent of this project is to reinforce concepts related to the operation of the assembler, the use of subroutines, program debugging and simulation, two’s complement arithmetic, and performing 16-bit operations using an 8-bit microprocessor.

Project Description
Create a simple main program and the subroutine that performs addition. The subroutine accepts the address of the operands in the FSR register and the address to place the result in the W register. Name the subroutine F_Add.

The two’s complement format allows negative and positive numbers to be represented in binary. The values \((-2^{15}) \rightarrow 2^{15-1}(-32,768 \rightarrow 32,767)\) can be represented with 16 bits. Refer to the Mano text for more information about the two’s complement format and associated arithmetic operations. Because the PIC16F84 is an 8-bit microcontroller it does not have the ability to manipulate 16-bit numbers natively. The project subroutine will use the 8-bit arithmetic operations of the PIC to implement 16-bit operations.

After calling F_Add to sum the array the main program computes the average value of the elements in the array.

Getting Started
Program Overview
Your program will consist of a main program and the subroutine F_Add. The requirements follow.

Main Program
Use the following cblock to define the memory locations needed for argument passing and receiving the result:

```assembly
     cblock 0x10
     add_arg:20 ; argument for F_Add (16 16-bit numbers = 0x20 bytes)
     add_rv:2  ; space for the return value (MSB is in lower reg file location)
     average:2 ; main computes the average and places it here
                 ; LSB in memory location following MSB

e ndc
```

The purpose of the main program is to call the subroutine with the proper parameters and then compute the average value of the array. The parameters to the subroutine will be in the memory locations defined by the preceding cblock. Before calling the subroutine the main program must place the parameters in FSR and W. This code sets up the registers for a call to F_Add:

```assembly
     movlw 0x10
     movwf FSR     ; set FSR to 0x10, the address of the operands
     movlw 0x30    ; set W to 0x30, the address to place the result
```

In your code, instead of hard coding the values 0x10 and 0x30, use the symbolic names from the cblock.
To calculate the average of an array of numbers, divide the total sum of the numbers by the length of the array. In this project the array always has length 16. Recall that shifting a number to the right is the same as dividing it by two. Therefore, to divide a number by 16 shift it to the right four times. Because we are dealing with signed number the arithmetic shift right operation must be used in order to preserve the sign bit.

F_Add Subroutine
The subroutine may not modify the operand memory. Therefore it may be useful to use this cblock:

cblock 0x40
addResult:2 ; space for the result before copying to final desitination
add_rv_addr ; address to place result when finished
addCount
tempByte ; temporary space for a byte
endc

You may use more or different file register locations if you desire, this is just a good starting point.
The FSR register contains the address of the operands. The operands are stored in 32 consecutive register file locations. The first 16 locations contain the least significant bytes (LSB) of the array elements. The remaining 16 locations contain the most significant bytes (MSB) of the array elements. So the LSB of element 0 is in location 0x10 with the corresponding MSB in location 0x20.
Overflow must be detected. If overflow occurs set the return value to 0xDEAD and set W = 1 and return. If all sixteen numbers are summed successfully then set W=0 before returning.

Summing an array of 16-bit numbers with 8-bit instructions is straightforward.
1. Clear the total by setting both the MSB and LSB to 0.
2. Sum the least significant byte (LSB) of the current element and the LSB of the total.
   If this causes overflow increment the MSB of the total. Do this for every element in the arrays.
3. Sum the MSB of the current element and the MSB of the total. Summing numbers
   with the same sign can cause overflow. If overflow occurs then the carry out will
   be different than the most significant bit (MSB). Do this for every element in the
   array.

You must use loops to sum the array of 16-bit numbers. You may not write 32 addwf
statements. There may be no more than 10 addwf/subwf operations in your program.

Additional Hints
You may define additional subroutines as you please. If the same operation must be performed in
different places or more than once it is generally beneficial to place the operation in a subroutine
so the code only has to be written and debugged once.

You may use statements of the form:

    movwf address+1

to access the memory location after addresult.
Using MPLAB
Refer the to CEL 2504 website for help with MPLab 6.30
http://www.ece.vt.edu/cel/Site-Map.html#2504
The link MPLab v6.30 Full Version (26 MB zip) (requires VT PID/password) contains the installer for the assembler and simulator. Follow the Quick Start Guide for help with setting up your workspace. There is also documentation about assembly language, the PIC16F84 instruction set, and the MPLab IDE available at the website.

1. Follow the procedures and methodology mentioned in this specification to develop your program’s algorithms.

2. Create a new assembly language source file that implements your algorithm. Fully document/comment your source code so that the GTAs and instructor can review your code and easily understand what you are doing and how you are doing it. Name your source file project4_yourPID.asm (example: project4_jdoe.asm). At the top of the file place in the comments your name, PID, and the name of your instructor.

3. Test and debug your code until neither logical errors nor assembler errors exist.

Submission Requirements
Turn in the following documentation of your design process.

- Lab 3 Cover Sheet provided in this document
- Program Verification Plan – Part of any engineering project is verifying the results are correct. This applies to both digital logic circuits and assembly language programs. Document how you verified your program worked correctly. It is not feasible to test the function for all possible input values. There are over 2^256 possible input combinations of 16 16-bit values. Your verification plan should ensure that all possible paths of execution are tested. For example, verifying F_Add can successfully sum 16 positive numbers does not verify that F_Add detects overflow. The Program Verification Plan should be less than two pages.

- The listing file for your program. Because the listing has more than 80 columns it should be printed in landscape mode to improve readability.

Additionally, submit a 3.5” floppy disk or CD-R containing just your assembly file. Disks must be labeled with your name, student number, and the name of your instructor. Unlabeled disks will not be checked resulting in a loss of the 25 validation points.

Honor Code
You must comply completely with Virginia Tech’s Honor System. For more information, see http://filebox.vt.edu/studentinfo/ugradhonor/index.html. All aspects of the program and report must be your own work. You may ask other students general questions about the PIC instruction set and how MPLAB works. You are not allowed to ask anyone except your instructor or an ECE 2504 GTA any questions about the design of the program. It is an Honor Code violation to share your design with another person or to copy another person’s design either as a paper design or as a computer file.
Bradley Department of Electrical and Computer Engineering
ECE 2504: Introduction to Computer Engineering ■ Fall 2003
Design Project 3: Software Implementation of 16-bit Arithmetic

Student Name: ____________________________________________

Student ID: _____________________________________________

Section (circle):  Davis   Hendry   Kachroo   Thweatt

__________________________

Pledge: I have neither given nor received unauthorized assistance on this assignment.

Signed: ________________________________________________

Project Grading to be completed by GTA or Instructor.

Grading: The design project will be graded on a 100 point basis, as shown below:

- Manner of Presentation (25 points)
  o Completed cover sheet with name, student ID, instructor, and signed pledge (3 points)
  o Report organization: clear concise presentation of information, use of sections
    (introduction, body, summary, etc.) (15 points)
  o Mechanics: spelling, grammar, etc. (7 points)

- Technical Merit (75 points)
  o Design procedure discussion (10 points)
  o Implementation: discussion of the operation of your program, what extra subroutines you
    wrote and why, what file register locations were used, how, and why (15 points)
  o Program Verification Plan: consists of more than “this testcase worked.” Explain the
testcases chosen and document how the testcases completely test your code (25 points)
  o Validation (25 points)

The primary basis for grading these elements will be the clarity of the comments in your program
and the thoroughness of your verification plan.

The basis for grading the program will be whether or not the submitted program works and is
consistent with the description given in the specification.

Grade: ____________ (of 100)