# Machine Organization and Assembly Language Programming

## Problem Set #2

Due: Wednesday 19 October 2005

1. Chapter 2. Ex 2.30. How many times will the line of code at label "outer" be executed? How many times will the line of code at label "inner" be executed? Give the result as a function of *n*, the number of elements in the arrays (in the exercise *n* = 2500).

2. Chapter 2 Ex.2.34. You are encouraged to use SPIM to find the errors (both "logical" and "syntactic"). In fact, SPIM might "self-correct" some of the errors.

3. Programming in SPIM. Chapter 2, Ex. 2.21 slightly modified as follows. The integer can be positive, negative or null. However, you can be sure that the ASCII string contains only the ASCII equivalents (cf. green card) of the characters "+", "-", and "0" through "9", and of course the null termination character (byte of value 0). The resulting value should be displayed on the SPIM console. (Note: In this problem we are not asking you to do any error checking on the input, that is you don't have to worry about the one-character strings "+" and "-" as well as the empty string or strings that have non-digit characters—except of course for the possible leading "+" or "-").

4. Programming in SPIM. Write a program in MIPS assembly language that finds the number of elements strictly greater than the last element, the minimum element, the maximum element, and the (integer) average of all elements in an array of integers (each element is a 32-bit positive, negative, or null integer). Your input, the array and its size, should be in ".data" section of your program. As output, your program should display the 4 values asked for above on the console with appropriate messages.

5. After reading the "Compact Code and Stack Architectures" on the CD under Chapter 2, Historical Perspectives (2.19) and the "In More Depth—Instruction Set Styles" on the CD, repeat "on paper" the last exercise if it had to be programmed on a stack machine. You do not have to write the part relative to the output on the console.

You can assume that there is a local area of storage that contains the array, its size, and locations to store the minimum value, the maximum value, etc. as well as any temporaries/constants that you might need. You can also assume that if you need constants, they have been initialized in that local storage.

You can be extremely vague in your addressing, i.e., use instructions like:

```
Push array[last]   #put last element of array on top of stack
Pop minimum        #i.e, store top of stack in "minimum" and pop
Pop                #remove top of stack
Ifeq target        #if top = 0 branch target. Pop (in either case)
```

Useful instructions will be Push, Pop, all arithmetic-logical operations that you need, all relational operators associated with transfer of control (ifeq, ifne etc.). Another operation that might be useful is "dup" (duplicate the top of stack).

A future programming assignment will be based on the stack architecture concept so this exercise is important.

You should design your own test files for the programming assignments but you won't have to turn them in. We'll have our own!

A good template for SPIM programs can be found under the "Software" section in the CS378 homepage or at:

http://www.cs.washington.edu/education/courses/378/05au/handouts/template.spim

Instructions for Turnin for the SPIM programming will be given to you shortly.