Lab 2: C and Assembly
Computer Organization
Spring 2018
30 points

Part A. (8pts)
3x3 Matrix Addition.
The two 3x3 matrix should be a 1D array of signed integers on the stack.

1. Write C code on your lab computer that makes these two arrays and write the for-loops to add them together all in the main function. You can overwrite one of the arrays. Make sure to compile and test it on the lab computers (this means add numbers to the arrays and print them out before and after). Save as *add.c*
2. Translate your code into MIPS using MARS. You do not have to do explicit prints. You can evaluate it using the memory in the simulator. Save as *add.asm*
3. Take the C code you wrote (minus the print debug stuff) and go to godbolt.org. Compile with both GCC MIPS and GCC x86-64 with the –O0 compile flag. Write a couple sentences about the differences between your MIPS code and GCC MIPS, plus a couple sentences about the differences between your GCC MIPS and GCC x86-64. Save as *add.txt*

Part B. (8pts)
Calling leaf procedures with multiple arguments.
Consider the code in *leaf.c* and write MIPS code for it in MARS.
You may not rearrange lines to reduce the number of instructions.
You may find the example on page 98 useful.
Save it as *leaf.asm*

Part C. (8pts)
Nested procedures.
Consider the code in *rec.c* and write MIPS code for it in MARS.
You may not rearrange lines to reduce the number of instructions.
You may find the example on page 101 useful.
Save it as *rec.asm*

Part D. (6pts)

Adding assembly to C code.

It is important to know assembly to help you have an understanding of how your program translates to assembly then to machine code to run. This helps you better write code and allows us to study the hardware the machine code is ran on. However, there is a second part. In many cases, assembly code is often handwritten to optimize certain pieces of code. This is normally done in two ways. The first is writing C code and writing a particular function in assembly. These two pieces can be linked together later during compile time. The second way is to inline assembly right into the C code. You will do the second type in this part of the lab.

1. Please read the following: http://www.delorie.com/djgpp/doc/brennan/brennan_att_inline_djgpp.html
2. Write a C program that in the main takes two Integers (one and two) and adds them together into an Integer (three). After it prints out the value in three. The twist is you have to use inline assembly "__asm__" in order to do the addition. I recommend looking online for more example using asm in GCC. Save this file as *inline.c*