CSE2312-002/003, Fall 2014, Programming Assignment 2
Basic ARM Assembly Programming
Due date via Blackboard: 11/13/14 11:59 PM

This assignment consists of 4 separate tasks. General guidelines for all tasks are below. **READ ALL THE INSTRUCTIONS BELOW. IF YOU DO NOT FOLLOW THESE INSTRUCTIONS YOU WILL NOT RECEIVE CREDIT!**

- For each task, you are provided with a partial solution. In this partial solution there is a line that says "Your code starts here" and a line that says "Your code ends here". You are only allowed to place your code between those lines. You are not allowed to modify any other part of the partial solution.
- **THIS IS DIFFERENT FROM ASSIGNMENT 1. READ IT!** For all tasks, you will convert a C function to assembly. You will notice at the top of the provided partial solution a branch with link to foo. Inside the area where you place your code is a foo label. You should implement your code here, put the return value into r0, and branch back when you function is finished (ex. bx lr). The partial solution will then branch to print10, which will print the value in r0, and then branch to my_exit. The input values a and b will be in registers r0 and r1, respectively.
- The partial solution that you are provided for each task starts by waiting for the user to enter two integers. To enter an integer, you should press ctrl-alt-3, type the integer, and press <ENTER>. There should be no spaces or other characters. Do not worry about what the program does if spaces or other characters are placed here.
- Each task specifies exactly what each character of the output should look like. Your solution needs to match the output specifications character by character (no differences in capitalization, more or fewer spaces, more or fewer empty lines). We will be using a grading script that will diff the output of your program with output of our (correct) program. If your output does not match ours exactly you will not receive credit.
- You will be given a pa02.zip file. When you unzip this file, you will see 4 folders inside called task1, task2 and so on. Each folder contains a task1.s (or task2.s and so on) file that provides the partial solution mentioned previously in addition to some other files. You should only modify the .s file. You can run your program using the same procedure as in homework 5.
- To turn in your programs, compress the entire pa02 directory to a .zip file. To do this, right click on the pa02 folder, select compress, choose .zip, and click create (and we assume you know how to use zip software at this point). Submit this .zip file on Blackboard as the solution for this programming assignment. Things you should not do include uploading the task.s files separately and compressing each task folder.
- IMPORTANT NOTES
    - Make sure your code ***ASSEMBLES*** and you follow the ***DIRECTORY STRUCTURE*** instructions specified above. If you code does not assemble or has a different directory structure, you will probably not receive any credit.
    - Make sure the output of your program matches the example output ***EXACTLY***. We will grade your assignment by comparing your program's output to the output of a correct solution automatically (using a variant of `diff`), so if your output differs slightly, you will receive little or no credit. Our comparison method is

reasonably smart and can distinguish minor spacing differences, but you are responsible for following the instructions to ensure your output is correct.

Task 1 (20 Points)

Consider the following function in C:

```
int foo(int a, int b)
{
  if (a == b) return 0;
  return b - a + 10;
}
```

Implement this function in assembly. After branching back from your function, the output should be as follows.

If the user enters 5 then 7:

5
7
12

END

If the user enters 5 then 5:

5
5
0

END

Task 2 (30 Points)

Consider the following code in C:

```
int foo(int a, int b)
{
  int result = 0;
  int i;
  for (i = 20; i <= 30; i++)
  {
    if (i == 24) result += 20;
    else if (i == 27) result = result * 2;
    else result = result + a + 3*b;
  }

  return result;
}
```

Implement this function in assembly. After branching back from your function, the output should be as follows.

If the user enters 4 then 3:

4
3
235

END

Task 3 (30 points)

Consider the following code in C:

```
int foo(int a, int b)
{
  if (a > b) return 0;
  if (a == b) return b;
  return a + foo(a+1, b);
}
```

Implement this function in assembly. After branching back from your function, the output should be as follows.

If the user enters 4 then 10:

4
10
49

END

If the user enters 10 then 4:

10
4
0

END

Task 4 (20 points)

Consider the following code in C:

```
int foo(int a, int b)
{
  int c = a+3;
  int d = c*5;
  int e = c+d;
  int f = e*d;
  int g = f - c;
  return a+b+c+d+e+f+g;
}
```

Implement this function in assembly. After branching back from your function, the output should be as follows.

If the user enters 2 then 3:

2
3
1560

END