CSE2312-002/003, Fall 2014, Programming Assignment 1
Basic ARM Assembly Programming
Due date via Blackboard: 11/04/14 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. You should branch to `my_exit` at the end of your program.
- The partial solution that you are provided for each task starts by waiting for the user to enter one or two integers (depending on the task). 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 the ranges of legal integer values that you should worry about. Do not worry about what your program will do for values outside those ranges.
- 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 pa01.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 pa01 directory to a .zip file. To do this, right click on the pa01 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)

Complete the task1.s code to produce an assembly program that:

- Waits for the user to enter an integer (this part is already coded).
- Stores the integer in `r5` (this part is already coded). This integer will correspond to an ASCII code.
- (This is what you need to code) Prints the character for each ASCII code from n -5 to n+5 where n is the integer entered by the user.
- Assume the integer entered by the user will be no less than 70 and no more than 85.

For example, if the user enters 76 your output should look exactly like this:

```
76
G
H
I
J
K
L
M
N
O
P
Q

END
```

Task 2 (30 points)

Consider the following code in C:

```
int i;
for (i = r5; i <= r6; i++)
{
  if (i == 24) printf("24\n");
  else if (i == 27) printf("27\n");
  else printf("z\n");
}
```

Complete the task2.s code to produce an assembly program that:

- Waits for the user to enter two integers (this part is already coded in).
- Places the first integer at r5, and the second integer at r6 (this part is already coded in).
- (This is what you need to code in) Produces the exact same output as the C code would produce, assuming that:
    - The integer at r5 is not smaller than 15 and not larger than 32.
    - The integer at r6 is not smaller than the integer at r5, and not larger than 32.
    - Variables r5 and r6 in the C code have the same values as the registers r5 and r6 have received.

For example, if the first integer is 20 and the second integer is 30, your output should look exactly like this:

```
20
30
z
z
z
z
24
z
z
27
z
z
z

END
```

Task 3 (20 points)

Complete the task3.s code to produce an assembly program that:

- Waits for the user to enter two integers (this part is already coded in).
- Places the first integer at `r5`, and the second integer at `r6` (this part is already coded in).
- (This is what you need to code in) Prints out the character for each ASCII code from the first integer up to (and including) the second integer, assuming that:
    - The integer at `r5` is not smaller than 32 and not larger than 127.
    - The integer at `r6` is not smaller than the integer at `r5`, and not larger than 127.

Each character should be on its own line.
For example, if the first integer is 80 and the second integer is 86, your output should look exactly like this:

```
80
86
P
Q
R
S
T
U
V

END
```

Task 4 (30 points)

Complete the task4.s code to produce an assembly program that:

- Waits for the user to enter two integers (this part is already coded in).
- Places the first integer at `r5`, and the second integer at `r6` (this part is already coded in).
- (This is what you need to code in) Adds the two integers together. If the result is between 0 and 9 (including 0 and 9) the program should print the result. If not, is should print a question mark (?).
- Assume both integers entered will be no less than 0 and no greater than 50

For example, if the first integer is 4 and the second integer is 5, your output should look exactly like this:

```
4
5
9

END
```

If the first integer is 10 and the second is 3, your output should look exactly like this:

```
10
3
?

END
```