Lab 5: Subroutines

Due Friday, 7 December 2018 11:55 PM

Minimum Submission Requirements

- Create a Lab5 folder (note the capitalization convention, include no extra characters in the directory name) that contains the following files:
  - Lab5.asm
  - Diagram.pdf
  - README.txt
- Commit and Push your repository
- Tag the commit that you would like to be graded
  - The tag must be in the form Lab5_submission_# (note the capitalization convention)

Lab Objective

In this lab, you will learn how to implement subroutines and manage data on the stack.

Lab Preparation

Read chapter 5 from Introduction To MIPS Assembly Language Programming.

Specification

This program will display a string and prompt the user to type the same string in a given time limit. It will check if this string is identical to the given one and whether the user made the time limit. If the user types in the prompt incorrectly or does not finish the prompt in the given time limit, the game is over.

String Prompts

The strings that will be given to the user are contained in the .data segment defined in the Lab5_test.asm file. The address of these strings will be part of the arguments to your subroutines. When testing for string correctness, you will be checking punctuation.

Timing

To monitor timing, you will be using syscall 30. This syscall gets the current system time in milliseconds as a 64 bit value. It stores the upper 32 bits in $a1 and the lower 32 bits in $a0.

The Stack

The stack is a data structure that we will be using in this lab. This will primarily be used the handle the preservation of certain register values at the start of a subroutine so that they can be restored at the end of a subroutine. The most notable use of this will be preserving the jump and link return address, $ra, so that you can navigate out of nested subroutines. In addition, the values in registers $s0 - $s7 must be preserved across subroutine calls.

Subroutines

Lab5.asm will contain the subroutines to display and check the string as well as keep track of the time limit. You must implement all of the subroutines listed, and you may create more of your own subroutines. If you plan on using any of the saved registers, $s0 - $s7, you must save these registers appropriately on the stack (push at the beginning of your subroutine, then pop at the end of the subroutine).

Nested Subroutines

Two of the following subroutines will be called from within another subroutine. Specifically, compare_strings will be called from inside of check_user_input_string, and compare_chars will be called from inside of compare_strings. In order to properly execute these nested subroutines, you must properly use the stack to handle the return address register, $ra.

```
#-----------------------------------------------------------------
# give_type_prompt
#
# input:  $a0 - address of type prompt to be printed to user
#
# output: $v0 - lower 32 bit of time prompt was given in milliseconds
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# check_user_input_string
#
# input:  $a0 - address of type prompt printed to user
#         $a1 - time type prompt was given to user
#         $a2 - contains amount of time allowed for response
#
# output: $v0 - success or loss value (1 or 0)
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# compare_strings
#
# input:  $a0 - address of first string to compare
#         $a1 - address of second string to compare
#
# output: $v0 - comparison result (1 == strings the same, 0 == strings not the same)
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# compare_chars
#
# input:  $a0 - first char to compare (contained in the least significant byte)
#         $a1 - second char to compare (contained in the least significant byte)
#
# output: $v0 - comparison result (1 == chars the same, 0 == chars not the same)
#
#-----------------------------------------------------------------
```

Testing

Your program must work properly with the testing program provided (on Canvas).

Output

Your output format should appear as follows where the prompt is displayed next to "Type Prompt:" and the user inputs their prompt beneath this.

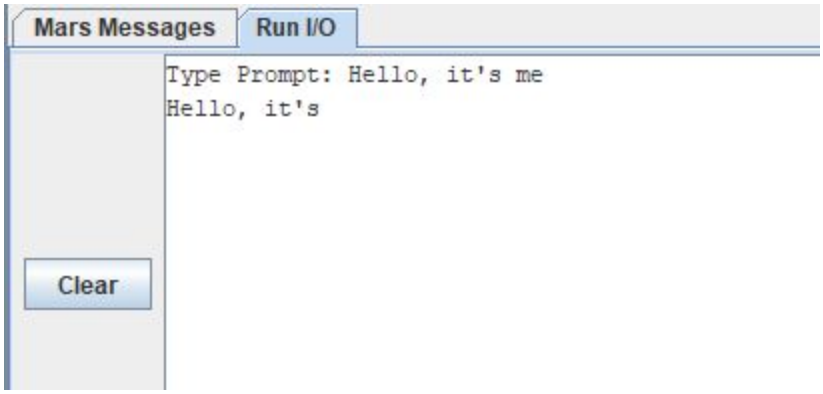

Diagram.pdf

This file will contain a block diagram or flowchart of how the different components of your code work together.

Lab5.asm

This file contains your assembly code. See the *Code Documentation* section from previous labs for instructions on proper code formatting. Pseudocode is not required for this lab, though you may include it after the header comment. You must include a block comment on register usage as indicated in previous labs.

README.txt

This file must be a plain text (.txt) file. It should contain your first and last name (as it appears on Canvas) and your CruzID. Your CruzID is your email address before @ucsc.edu. Your answers to the questions should total at least 8 sentences with complete thoughts.

Your README should adhere to the following template:

```
------------------------
Lab 5: Subroutines
CMPE 012 Fall 2018

Last Name, First Name
CruzID
------------------------

What was your design approach?
Write the answer here.
```

What did you learn in this lab?
Write the answer here.

Did you encounter any issues? Were there parts of this lab you found enjoyable?
Write the answer here.

How would you redesign this lab to make it better?
Write the answer here.

Extra Credit

As extra credit, alter your compare_strings subroutine to not return a 0 if the user input string only differs by punctuation or capitalization. It should still require correct spelling and spacing. Ex. “Hello, it’s me” user input “hello its me” should return a 1 from compare_strings.