CSE31: Lab #8 – More Procedures

Overview

In this lab, we will continue to work with **procedures (or functions)** in MIPS. We will learn about how to apply register convention into recursive functions.

Getting started

Before we begin any activities, create a directory (Lab_8) inside the CSE31 directory you created during Lab #1. You will save all your works from this lab here.

(Exercise) User inputs in MIPS?

TPS (Think-Pair-Share) activity 1 Paired with the classmate sitting next to you and do the following tasks (20 minutes):

1. Load ***fib.s*** in MARS and study the code. The is the same program we worked on in *Lab #6*.
2. Recall that ***fib.s*** calculates the 13th Fibonacci number (n = 13). Now let's make this program more generic so it will calculate the nth Fibonacci number, where ***n*** can be any number from a user input.
3. From *Lab #6*, we have learned how to print out statements in MIPS. Insert instructions in ***fib.s*** so that the program will print out ***"Please enter a number:"*** at the beginning of the program to prompt user for input.
4. In the program, ***$t3*** was used to store the value of ***n***. Now, let's read in a user input and save that value into ***$t3***. Do a search in the MARS documentations to find out how to use ***syscall*** to read an ***INTEGER*** from a user. **Again, you must store the value into $t3**.
5. Since the program now reads a number from a user, do we need to declare ***n*** in the *.data* segment of the program? How about the ***la*** and ***lw*** instructions regarding ***n***? Comment out those instructions so they won't mess up your program.
6. Assemble the program and test it with different numbers to see if it runs correctly. (You can use the original ***fib.s*** to verify your results.)

(Exercise) Recursive functions

In Lab #7, we've understood how register convention can help us manage registers in procedures. Let's find out how we can follow register convention in recursive functions.

TPS (Think-Pair-Share) activity 2 Paired with the same classmate and answer the following questions (30 minutes):

1. Study ***recursion.c*** and trace the program. Without running the program, what will be the output if ***5*** is entered? Compile and run ***recursion.c*** in a terminal (or any IDE) and verify your answer.

2. Load ***recursion.s*** in MARS. This is the MIPS version of ***recursion.c***. Do not assemble and run this program, as the program is incomplete. Study the ***MAIN*** function and discuss with your partner about what it does (compare it with the C version). A lot of instructions are missing, and we will fill them out in the following steps.

3. Since the ***recursion.c*** prompts to a user for input, insert instructions in ***recursion.s*** so the program will prompt the same statement to a user.

4. Insert statements for the program to read in a value from a user. What register should we use to store that value? (Hint: you will use it as the *argument* for ***recursion*** function call.)

5. Next, the ***main*** function calls ***recursion*** with the correct input argument. After returning from ***recursion***, we need to print out the returned value. What register do we expect the returned value to be stored in? However, the ***syscall*** for printing out a value is also using the same register. What can we do?

6. Based on your answer from #5, insert the correction instructions to print out the returned value before jumping to the end of program.

7. Now, let's complete the ***recursion*** function. The stack pointer was moved to create extra storage for the function. How many integer values are reserved in this storage? What is the first thing to be stored in this stack frame? Insert a statement to accomplish this.

8. Based on the branch statement under label **recursion**, update the returning value. Again, you must use the correct register to store the returning value.

9. Based on the branch statement under label **not_minus_one**, update the returning value. Again, you must use the correct register to store the returning value.

10. When the input argument is not 0 or -1, the program will call ***recursion*** 2

times. This happens in the code under label **not_zero**. Why do we need to save $a0 into the stack?

11. Insert a statement to update the input argument for the next ***recursion*** call.

12. After returning from the last ***recursion***, the program is about to call the next ***recursion***. However, the last ***recursion*** came back with a returned value. What will happen to if we call ***recursion*** right away? Insert statements to prevent this from happening.

13. Now the program is ready to call ***recursion*** again. Insert statements to update the next input argument.

14. After returning from the second ***recursion*** call, insert statements to update the final value to be returned to ***main***.

15. Before returning to main, a value needs to be retrieved so the program can return to the correct location of the code. What is this value? Insert a statement under the label **end_recur** to retrieve this value.

(Assignment 1, individual) Create recursion1.s

Study ***recursion1.c*** and translate the same program in MIPS following register convention. You can compare the output of your MIPS program with that of ***recursion1.c***.

Save your program as ***recursion1.s***.

What to submit

When you are done with this lab assignments, you are ready to submit your work. Make sure you have included the following ***before*** you press Submit:

- Your ***fib.s, recursion.s***, ***recursion1.s***, answers to the TPS activities in a text file, and a list of Collaborators.
- Your assignment is closed **7 days after this lab is posted** (at 11:59pm).
- You must demo your submission to your TA **within 14 days** (preferably during next lab so you will have a chance to make correction and re-submit/re-demo.)