**Lab 08: More Procedures in MIPS**
**CSE-031-01** **Points: 20**

## Overview

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

## Getting Started

Before we begin any activities, create a directory (`Lab_8`) inside the `CSE31` directory we created in the first lab. You will save all your work from this lab here. **Note that all the files shown in green below are the ones you will be submitting for this assignment.**

**You must have a clear idea of how to answer the TPS questions before leaving lab to receive participation score.**

## User Inputs in MIPS

**TPS (Think-Pair-Share) activity 1:** Discuss questions 1 – 6 (20 minutes) while paired with your classmates assigned by your TA (you will be assigned to groups of 3-4 students) and record your answers in a text file named **`tpsAnswers.txt`** under a section labelled "TPS 1" (***you will continue to use this file to record your answers to all the TPS questions that follow in the lab handout***):

1. Load **`fib.s`** in MARS and study the code. This is the same program we worked on during Lab 06.
2. Recall that **`fib.s`** calculates the 13th Fibonacci number (`n = 13`). Now let us make this program more generic so it will calculate the `n`th Fibonacci number, where `n` can be any number from a user input.
3. From Lab 06, 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, `$t3` was used to store the value of `n`. Now, let us read in a user input and save that value into `$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 will not mess up your program.
6. Assemble the program and test it with different numbers to see if it runs correctly (you may use the original **`fib.s`** to verify your results.).

Your TA will "invite" one of you randomly after the activity to share what you have discussed.

## Recursive Functions

In Lab 07, we understood how register conventions can help us manage registers in procedures. Let us find out how we can follow register conventions in recursive functions.

**TPS activity 2:** Discuss questions 1 – 6 (30 minutes) with your TPS partners in your assigned group and record your answers in **`tpsAnswers.txt`** under a section labelled "TPS 2":

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 – **the program is incomplete**. Study the `main` function and discuss with your partner(s) 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 your `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 step 5, insert the correct instructions to print out the returned value before jumping to the end of program.
7. Now, let us 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

Your TA will "invite" one of you randomly after the activity to share what you have discussed.

## Individual Assignment 1: 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`. **Note: You MUST follow the MIPS register and calling conventions discussed during lectures and shared in CatCourses (see the Announcement) .**

## Collaboration

You must credit anyone you worked with in any of the following three different ways:

1. Given help to
2. Gotten help from
3. Collaborated with and worked together

## What to hand in

When you are done with this lab assignment, submit all your work through CatCourses.

***Before*** you submit, make sure you have done the following:

- Attached `fib.s`, `recursion.s`, `recursion1.s` and `tpsAnswers.txt`.

- Filled in your collaborator's name (if any) in the "Comments…" textbox at the submission page.

Also, remember to demonstrate your code to the TA or instructor before the end of the grace period.

**Scoring:**

- TPS activity 1: 3pts (total) – you may be asked to demo your code.
- TPS activity 2: 7.5pts (total) – you may be asked to demo your code.
- Individual Assignment 1: 9.5pts (with demo, otherwise 0)