CSE31: Lab #7 – Procedures in MIPS

Overview

In this lab, we will continue to familiarize ourselves with programming MIPS using MARS. We will focus on writing **procedures** in MIPS

Getting started

Before we begin any activities, create a directory (Lab_7) inside the CSE31 directory you created during Lab #1. You will save all your works from this lab here.

(Exercise) How procedures work in MIPS?

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

1. Perform a search on the Internet on the difference(s) between the terms ***procedures***, ***functions***, and ***methods***.
2. As we have learned in lectures, our compiled program is stored in the static part of the memory when it is being executed. When the CPU runs a program, it executes the statements according to the statement order (line numbers). Unless there is a branch (*beq*/*bne*) or jump (*j*/*jal*) statement, it will just execute the next statement.
3. Load ***proc1.s*** in MARS and study the MIPS program without assembling and running it. **Write an equivalent program in C and name it *proc1.c*.** You can treat ***m*** and ***n*** as variables declared in main.
4. In ***pro1.s***, we use **j SUM** to ask the CPU to jump to the line with the label ***SUM*** and continue running the program from there. What line number is this? What does this line do?
5. After function ***SUM*** is over, the program is supposed to return to the line after **j SUM**. In the code, **jr $ra** is used. Can we use **j** instead (we can create a label for that line)?

6. Since the return address keeps changing depending on where ***SUM*** is called, we need to save the return address before ***SUM*** is called. At what line in ***proc1.s*** is the return address supposed to be saved? In what register is the address saved to? What is the value of address being saved here? Does this address value make sense?
7. Assemble the code and open the **Execute** tab. Here the program is listed in the *Text Segment* (as we have seen this in last lab). What happens when you try to run the program? This error is due to the invalid return address (program counter tells the CPU where to look for a statement).
8. Now, let's correct the return address value. From the *Text Segment* window, what is the address of the statement that the program should return to from ***SUM***?
9. Modify the code so the correct return address is saved. Assemble it and take a look at the *Text Segment* again. (DO NOT execute it yet!) Double check the return address. Is it correct? What happened? You will know more about what happened here in later lectures.
10. What is the new return address? Modify your code, assemble, and run the program. What is the output of the program?
11. As you can see, saving the correct return address before each procedure call is tedious. It would be nice if the assembler can do it for us! Instead of using **j** to call a procedure, what operator should we use?
12. Modify the code so you don't use **j** to call SUM. Make sure to comment out the line where the return address is saved.

(Exercise) Register convention

Now we've understood how procedure works in MIPS programs, let's dig deeper into how to manage registers in procedures.

Even though there are 32 registers in a MIPS CPU, we technically can only use about 24 of them. As a result, the same register (e.g. *$s0*) may be used in different procedures to store local variables. Following a register convention when writing MIPS programs will save us a lot of time in managing the use of registers. Refer to Lecture #11 for the exercise below.

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

1. Study ***proc2.c*** and trace the program. What will be the output if you run the program? Compile and run ***proc2.c*** in a terminal (or any IDE) and verify your answer.
2. Load ***proc2.s*** in MARS. This is the MIPS version of ***proc2.c***. Do not assemble and run this program, as there are errors due to the misuse of registers. Study the ***MAIN*** function and discuss with your partner about what it does (compare it with the C version).
3. When ***MAIN*** calls ***SUM***, ***SUM*** knows where to return to. Why? After ***SUM*** called ***SUB***, what happens to the address returning to ***MAIN***? Discuss with your partner about how you would resolve this problem. Do not attempt to fix it yet, as we have more problems to come.
4. The input argument (***n***) in function ***SUM*** is used to call the next function, SUB, as well as being added to the return value. According to the register convention, the first argument of all function calls must be stored in ***$a0***. From line 28 of ***proc2.s***, the value in ***$a0*** is no longer the same as the input argument of ***SUM*** (it has been changed to store input argument of ***SUB***). We can resolve this problem by saving the original ***$a0*** into a temporary register, but we may eventually run out of registers if our program is large. Discuss with your partner about how you would resolve this problem. Do not attempt to fix it yet.
5. Let's take a look at line 25, what happens to the original value in ***$s0*** from ***MAIN*** after this statement is executed? Is this a problem? Why? Discuss with your partner about how you would fix this problem. Do not attempt to fix it yet.
6. Now we know that SUM needs to backup 3 values before calling SUB. Insert prologue and epilogue into the code so the program will run correctly. (Hint: study function SUB, as it contains not error)

(Assignment 1, individual) Create proc3.s

Study the ***proc3.c*** and re-write the same program in MIPS with the following requirements:

1. Local variables mapping:
    a. ***main(): x -> $s0, y -> $s1***
    b. ***sum(): p -> $s0, q -> $s1***
2. Input arguments mappings:
    a. ***sum(): m -> $a0, n-> $a1***
    b. ***sub(): a -> $a0, b-> $a1***
3. All return values from a function must be stored in V registers in ascending order (i.e. $v0, $v1).
4. Use of stack memory according to register convention.

Save your code as ***proc3.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 ***proc1.c, proc1.s, proc2.s***, ***proc3.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.)