CS341, Computer Architecture Lab, Lab 03

Goals

1. Understand stack operations, by writing recursive functions

Instructions

1. These exercises are to be done individually.
2. While you are encouraged to discuss with your colleagues, do not cross the fine line between discussion *to understand* versus discussion as a *short-cut* to complete your lab without really understanding.
3. Create a directory called <rollno>-<labno>. Store all relevant files to this lab in that directory.
   a. In the exercises, you will be asked various questions. Note down the answers to these in a file called “<rollno>-<labno>.txt”.
   b. In some parts of the exercises, you will have to show a demo to a TA; these are marked as such. The evaluation for each lab will be in the subsequent lab, or during a time-slot agreed upon with the TAs. For this evaluation, you need to upload your code as well.
   c. While submitting (on BodhiTree), you have to create a tar.gz or zip of the entire <rollno>-<labno> directory in which all your relevant files reside.
4. Before leaving the lab, ensure the following:
   a. You have signed the attendance sheet
   b. You have uploaded your submission
5. Things to ensure during TA evaluation of a particular lab submission:
   a. The TA has looked at your text file with the answers to various questions
   b. The TA has given you marks out of 10, and has entered it in the marks sheet, with his/her signature
   c. You have counter-signed the above-mentioned marks
6. You have to use the MIPS conventions, unless mentioned otherwise.

Getting warmed up with 'factorial'

- You will first code up and test the assembly version of the recursive factorial computation, in a file called “factorial.s”. *Hint: Like always, you will find it useful to write the C-code version and translate it (blindly). Note: the code for the factorial is given in the slides, you are not allowed to look at this, but only **before** coding, **not while** coding.*
- Your main program should call the factorial routine on an integer (this can be hardcoded).
- **Demo to TA [2 marks]:** show the working of the above program.

Recursive merge-sort

- Take the merge procedure written in the 2nd lab. Make sure you are familiar with it (if you had forgotten), by running it again.

- You are now going to use the above merge procedure, and write a recursive merge-sort procedure, in file “merge-sort.s”. The recursive procedure should take two arguments: the starting address of the sub-array to be sorted, and the length of this sub-array. The procedure, before it returns, should guarantee that the (sub-)array given as argument is sorted. *Hint-1: You can use a global temporary array for the merging, after the recursive calls, and copy over from the temporary array to the relevant portion of the main array. Hint-2: As earlier, it would be useful to first debug the C-code version of the program, and then translate it (blindly) into assembly code.*

- Your main routine should input an 8-element integer array from the user, and call the merge-sort procedure to sort it.

- **Demo to TA [4 marks]:** show the working of the above program.

Using break-point to watch the stack

- **Question [4 marks]:** At what point in your tree of call activations will the stack depth be the maximum (just with respect to the recursive calls)? Insert a breakpoint at a place in the code where the stack is at its maximum size. Examine the stack and explain its contents with reference to your code. Point out the various call activation records. *Hint: you have to insert a breakpoint at an appropriate place in the mergesort procedure, not the merge procedure.*