CS231, DLDCA Lab, Lab 06

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 "answers.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.
4. Before leaving the lab, ensure the following:
    a. You have marked attendance on SAFE
    b. You have uploaded your submission on BodhiTree, and downloaded and checked if the submissions is right
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
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 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 binary search

- Write a recursive binary search function in MIPS assembly code in file binarySearch.s. The function must take 5 arguments, as per the C prototype below:

```
// Take an array A with 'len' elements as input; A is assumed to be sorted in increasing order

// Search for value 'val' in the index range [start, end)

// return index where 'val' is found, if it is found; return -1 if 'val' not found

int binarySearch(int *A, int len, int start, int end, int val);
```

- While it may be possible to write the function with a different set of arguments, you MUST write as per the above specification. Also, you MUST use recursion, a non-recursive solution will fetch you ZERO marks.
- You must follow the MIPS coding conventions in terms of caller and callee saved registers and other register usage. You can define and use your own stack frame format though. Specify your stack frame format in the answers text file.
- In the same file, declare a static array A of size 8 (or more); ensure to initialize it in a sorted manner.
- From the main routine, call the recursive binarySearch, to search for a given value (taken via STDIN). The main function should call:

  binarySearch(A, <len–of–A>, 0, <len–of–A>, <some–val>)
- Print the result of the search onto the console (STDOUT).
- ***Hint****: As earlier, it would be useful to first debug the C-code version of the program, and then translate it (blindly) into assembly code.*
- **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****: What input of 'value' will cause the maximum recursion depth? Think about this.*