CS341, Computer Architecture Lab, Lab 02

Goals

1. Learn xspim system calls
2. Learn global data declaration in MIPS assembly language
3. Understand arrays in the MIPS data segment
4. Writing a function call

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.

Learning xspim system calls for input/output

- Look at: http://www.cs.uic.edu/~troy/spring04/cs366/mp1.html

- The above webpage shows you how to use the "syscall" instruction to do simple input/output in xspim
- Examples are given in: http://www.cs.uic.edu/~troy/spring04/cs366/ex2.s
- Copies of the above pages are also available on BodhiTree.
- **Demo to TA [1 mark]:** As an exercise, write a program called "add.s", to input two integers, add them, and print the sum; show how your above program works, to your TA. There is no need to have any prompts for the input (no need to print any string).

Global data declarations, arrays

- Search the web for "MIPS assembler directives", and learn how to statically allocate global data using the ".data" and related assembler directives.
- **Demo to TA [1 mark]:** Now statically declare some prompt strings, and redo the above program to input two integers. Now you must use appropriate strings to prompt the user, and to print the result (sum). *Note: you can demo this jointly with the previous one, since you will anyway likely be over-writing the same file.*
- **Demo to TA [2 marks]:** Now in the same program, include an array of 8 integers declared as static data. Be sure to use the ".align" assembler directive, after understanding what it does (use google). Your main routine should include a loop to read 8 values from the console (using the appropriate syscall) and store it in the 8 array locations. You should use appropriate prompt strings. Show your TA the appropriate portion of the data segment in xspim, after inputting the 8 values.
- **Demo to TA [1 mark]:** Examine what happens without the ".align" directive. Create a situation where there is mis-aligned access and show/explain the simulator behaviour to your TA.

Using arrays, writing functions

- Open a new file called "merge.s"
- Write a function called "merge", which takes three integer arrays as arguments; there will actually be five arguments (each 32-bit): array pointer 1, array length 1, array pointer 2, array length 2, and result array pointer. The function can assume that the input arrays are sorted, and should merge the input arrays into the result array. The function can assume that the result array has enough space.
    - ***Hint-1:*** *first write a C-program, debug the logic, and then translate the same into assembly language.*
    - ***Hint-2:*** *you will be better off blindly translating the C-code to assembly code; else you may find debugging non-trivial!*
    - ***Warning:*** *stepping through the program or continuing after a breakpoint may change any unpreserved register (why? we will answer this later in the course).*
- Your main function should input an array of 8 integers onto an array (you can copy code from the previous exercise). Call merge on the two halves of the 8-element array. Don't forget to declare another 8-element array in your data segment, for the output. The main routine should also print the output array.

- **Demo to TA [4 marks]:** show the working of the merge routine to your TA.

- **Question [1 mark]:** What were the aspects which were most difficult in the above assembly language programming? Which aspects can be improved with better practice, and which aspects cannot be?