CS231, DLDCA Lab, Lab 05

Goals

1. Learn qtspim 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 "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.

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 qtspim
- 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, calling it **add-prompt.s**. Now you must use appropriate strings to prompt the user, and to print the result (sum).

- Now consider the following C structure and global variable definitions:

  ```
  typedef struct {
  int re; int im;
  } complex;
  complex A[4] = { {0,0}, {-1,2}, {0,2}, {-1,-1} };
  int len=4;
  ```

- **Demo to TA [1 mark]:** In a program called **aDeclare.s**, declare the global variables corresponding to the above C declarations, as static data. Be sure to use the .align assembler directive, after understanding what it does (use google). Now add the real and imaginary parts of A[2]. In this process, you would have use the pseudo-instruction 'la' after learning what it does.

- **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

- Now open a new file called **numLessThan.s**

- Write functions isLessThan and numLessThan, with the following C prototypes.

  ```
  // e1 is considered less than e2 if e1.re<e2.re or

  // e1.re==e2.re and e1.im<e2.im

  // Return 1 if e1<e2, 0 otherwise

  int isLessThan(complex e1, complex e2);



  // Given argument 'elt', return the number of elements of A less than

  // 'elt'; The sub-array of A to considered here is [start, end)

  int numLessThan(complex elt, complex* A, int start, int end);
  ```

- The function numLessThan should call isLessThan internally. Subsequently, write the main function, which will take as STDIN input the real and imaginary parts of a complex 'x', and call numLessThan(x, A, 0, len). It should then print the return value on STDOUT.

- The arguments of numLessThan cannot all be accommodated in the 4 argument registers. This is intentional, so that you learn to use the stack for passing the additional argument.

- ***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! By translating blindly, I was able to do the assembly coding in about half an hour, and got the code working on the first run.*

- ***Hint-3****: When numLessThan calls isLessThan, you would have to carefully prepare the arguments; some of its own arguments are passed to isLessThan, but in a different order.*

- ***Warning****: Stepping through the program or continuing after a breakpoint may change any unpreserved register (this is a possible qtspim bug).*

- **Demo to TA [5 marks]:** Show the working of the numLessThan routine to your TA. If only the isLessThan part works, you will get 1 mark in this part. If you do not follow the standard MIPS conventions of caller and callee saved registers, and argument/return registers, you will get a maximum of 2 marks in this part.

- **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?