CS 211: Computer Architecture, Fall 2012

Programming Assignment 3: mysFunc

Due by Nov. 12 - Midnight

1 Introduction

This assignment is designed to give you additional practice in understanding Assembly Language programs. As discussed in lecture, unless you are working in increasingly rare areas such as low-level OS development, you are unlikely to be reading and/or writing Assembly Language programs in the remainder of your career. However, we are still requiring you to write some here to make sure you understand the computing model underlying your fancy C and Java programs. Being able to read Assembly Language is particularly important because there are times when you need to understand what the compiler is doing to your code. Having some experience of reading assembly codes will help improve your reading skills too.
In this assignment, you will read a mystery function in Assembly and implement a function in C has the same functionality.

2 Implementation

Resource

You will be given four files:
mysFunc.c: A C program file consisting of a main function that calls the *mystery()* function
mystery.h: A header file with the prototype for the *mystery()* function so that it can be called by *mysFunc.c*
mystery.s: An assembly program implementing the mystery function
Makefile: a file used to compile the source code

Objective

Write a function using **C programming language** which implements the same functionality as ***mystery.s***.

Usage

Compile and run your program on an iLab machine. Please name your executable file as "***mysFunc***".
The program should have the following usage interface:

mysFunc <input_data>

where:
mysFunc: is the executable file of this program
input_data: is an integer in the range of [1, 20]
The output of the function is also an integer.
Example:

Input: ***mysFunc*** **2**
Output: **1**

Important: The assembly program is designed to run on machines with an x86 architecture and 64-bit processors. Your personal computers may not have 64-bit processors. Please ensure that you execute the assembly program on a x86-64 bit machine. The iLab machines meet this criterion.

Steps

1. Compile and run the source code by yourself.
Put all the files into one directory.
$ make //compile
$ ./mysFunc 2 //run the program
You can try other integer as input and get the corresponding output.

2. Trace the value of some key registers as *mystery.s* executes and write those values in the file "***regvalue.txt***". Trace the values for an example where **INPUT DATA is 5**. The registers whose contents you should track are "***eax, edx***".
regvalue.txt should be formatted in the following manner:

```
eax: a1          //the first value of eax
     a2          //the changed value of eax
     …           //other value
edx: d1
     d2
     …
```

3. You need to implement a C program with the name "***mystery.c"***, which will be called by "***mysFunc.c"*** as provided to you. You should not change ***mysFunc.c***, only change the ***Makefile*** to compile your "***mystery.c***" with the two source files "***mysFunc.c***" and "***mystery.h***". Test your program with a variety of different values, the output should be identical to that of "***mystery.s***".
IMPORTANT: Make sure your program can be complied and executed in the iLab environment by simply typing make at the command prompt. Your grade for the assignment will be a zero if your code will not compile and execute ie if your Makefile or code have to be changed in order to compile and run.

3 Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named pa3.tar. Follow similar instructions as PA1 for tar file construction.

Your tar files must contain:
readme.pdf: this file should describe your design and implementation. In particular, it should detail your design, any design/implementation challenges that you ran into, and an analysis of the space and time performance of your program.
Makefile: following instruction in PA1, making any modifications as needed.
mystery.c: the file includes your function implementation using C.

4 Grading Guidelines

4.1 Functionality

This is a large class so that necessarily a significant part of your grade will be based on programmatic checking of your program. That is, we will build a binary using the Makefile and source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- You should make sure that we can build your program by just running make.
- You should test your code as thoroughly as you can. In particular, your code should be adept at handling exceptional cases.

Be careful to follow all instructions. If something doesn't seem right, ask.

4.2 Coding Style

Having said the above about functionality, it is also important that you write "good" code. Thus, part of your grade will depend on the quality of your code. Here are some guidelines for what we consider to be good:

- Your code is modularized. That is, your code is split into pieces that make sense, where the pieces are neither too small nor too big.
- Your code is well documented with comments. This does not mean that you should comment every line of code. Common practice is to document each function (the parameters it takes as input, the results produced, any side-effects, and the function's functionality) and add comments in the code where it will help another programmer figure out what is going on.
- You use variable names that have some meaning (rather than cryptic names like i).

Further, you should observe the following protocols to make it easier for us to look at your code:

- Define prototypes for all functions.

4.3 Performance

Finally, part of your grade will depend on your design. That is, we expect you to write reasonably efficient code based on reasonably performing algorithms. You will need to analyze the performance of your code and justify it as part of discussing your design.