CS61C Summer 2015 Lab 3 - MIPS Assembly

Goals

This lab will give you practice running and debugging assembly programs using the MARS simulator.

Setup

Copy the lab files with

```
$ cp -r ~cs61c/labs/03/ ~/labs/03
```

Intro to Assembly and MARS

The following exercises use a MIPS simulator called MARS, which provides a rich debugging GUI. You can run MARS on your home computer by downloading the jar file from the Internet or by copying it from `~cs61c/bin/Mars4_3.jar` on the instructional machines. You will need Java J2SE 1.5.0 (or later) SDK installed on your computer, which can be obtained from Sun. If your home computer is a Mac, you can also follow the instructions here to install MARS as an app in one step.

You can run MARS in lab by typing '`mars &`' at the command line. The ampersand is optional but will allow you to continue using that terminal window (on the Macs however, you'll need to run it in it's own terminal tab by pressing command-t first). To run the program remotely, you may either run via an instructional server (but NOT one of the Orchard machines), or through a local installation (recommended). When on an instructional server, you will need to be running an X-Server (like XMing), and enabling X11 tunneling.

Tip: Although it is possible, you should avoid running MARS remotely at all costs - it will be painfully slow to use and will overwhelm the servers if many students attempt to do so. It is in your best interest to setup/run a local copy of MARS.

Assembly Basics:

- Assembly programs go in text files with a `.s` extention.
- Programs must contain a label "`main:`" (similar to the `main()` function in C programs).
- Programs must end with a "`addi $v0,$0,10`" followed by a "`syscall`". `main()` is special and must transfer control back to the operating system when it is done rather than just returning.
- Labels end with a colon (`:`).
- Comments start with a pound sign (#).
- You CANNOT put more than one instruction per line.

Exercises

Exercise 1: Familiarizing yourself with MARS

Getting started:

1. Run MARS.
2. Load `lab3_ex1.s` using File-->Open.
3. View and edit your code in the "Edit" tab. Notice the code highlighting and 'completion suggestion' features.
4. When ready, assemble your code using Run-->Assemble (or press F3).
5. This will take you automatically to the "Execute" tab, which is where you can run and debug your program.
6. Step through the program using Run-->Step (or press F7).
7. You should take the time to familiarize yourself with everything in the Run menu (and the keyboard shortcuts).

For this exercise, we calculate the Fibonacci numbers using `fib[0] = 0; fib[1] = 1; fib[n] = fib[n-1] + fib[n-2]`.
Follow the steps below and record your answers to the questions. The Help menu (F1) may come in handy.

1. What do the `.data`, `.word`, `.text` directives mean (i.e. what do you use them for)?
2. How do you set a breakpoint in MARS? Set a breakpoint on line 14 and run to it. What is the instruction address? Has line 14 executed yet?
3. Once at a breakpoint, how do you continue to execute your code? How do you step through your code? Run the code to completion.
4. Find the "Run I/O" window. What number did the program output? If 0 is the 0th fib number, which fib number is this?
5. At what address is `n` stored in memory? Try finding this by (1) looking at the Data Segment and (2) looking at the machine code (Code column in the Text Segment).
6. Without using the "Edit" tab, have the program calculate the 13th fib number by *manually* modifying this memory location before execution. You may find it helpful to uncheck the "Hexadecimal Values" box at the bottom of the Data Segment.
7. How do you view and modify the contents of a register? Reset the simulation (Run-->Reset or F12) and now calculate the 13th fib number by (1) breaking at a well-chosen spot, (2) modifying a single register, and then (3) unsetting the breakpoint.
8. Lines 19 and 21 use the `syscall` instruction. What is it and how do you use it? (Hint: look in Help)

Checkoff

- Show your TA that you are able to run through the above steps and provide answers to the questions.

Exercise 2: A short MIPS program

Write a piece of MIPS code from scratch that, given values in `$s0` and `$s1`, accomplishes the following:

```
$t0 = $s0
$t1 = $s1
$t2 = $t0 + $t1
$t3 = $t1 + $t2
...
$t7 = $t5 + $t6
```

In other words, for each register from `$t2` to `$t7`, store the sum of the previous two `$t#` register values. The `$s0` and `$s1` registers contain the initial values. Set the values of `$s0` and `$s1` manually with MARS instead of in your code. Finally, have your code print out the final value of `$t7` as an integer (Hint: `syscall`).

Save your code in a file called `lab3_ex2.s`. Don't forget the "`main:`" label and to end your program with a "`syscall 10`"!

Checkoff

- Show `lab3_ex2.s` to your TA, who will give you numbers to enter into `$s0` and `$s1`. Your code should print `$t7` and exit normally.

Exercise 3: Compiling from C to MIPS

For this exercise we need to use a program called `mips-gcc` (a cross-compiler for MIPS) that allows us to compile programs for the MIPS architecture on our x86 machines. If you are doing this lab remotely, you should SSH into one of the hive machines.

Compile The file `lab3_ex3.c` into MIPS code using the command:

```
$ mips-gcc -S -O2 -fno-delayed-branch -I/usr/include lab3_ex3.c -o lab3_ex3.s
```

The `-O2` option (capital letter "O" and 2) turns on a level of optimization. The `-S` option generates assembly code. Don't worry about the delayed branch option for now; we will revisit this topic again when we talk about pipelining. The above command should generate assembly language output for the C code. Please note that you will NOT be able to run this code through MARS.

Find the assembly code for the loop that copies sources values to destination values. Then find where the `source` and `dest` pointers you see in `lab3_ex3.c` are originally stored in the assembly file. Finally, explain how these pointers are manipulated through the loop.

Checkoff

- Find the section of code in `lab3_ex3.s` that corresponds to the copying loop and explain how each line is used in manipulating the pointer.