Download MARS from the following link:
http://courses.missouristate.edu/KenVollmar/MARS/
Goals
This lab will give you practice running and debugging assembly programs using the MARS simulator. You also need Java J2SE 1.5.0 (or later) SDK installed on your computer, which can be obtained from Oracle.

Running Assembly programs
Assembly programs go in text files with a .s extention. The program must contain a label "main:" (similar to the main function in C programs) and end with a "addi $v0,$0,10" followed by a "syscall". Unlike normal functions which use "jr $ra" to return main is special and must transfer control back to the operating system when it is done rather than just returning.

For this lab we will be running our code in MARS, a MIPS simulator which provides a rich debugging GUI, rather than trying to run on a bare processor. In general, assembly programmers prefer this mode of development when possible as it is far easier to debug and work with the code.

You can run MARS in the lab simply by typing the mars command.

To run the program remotely, you may either run via an instructional server (but not one of the macs), or through a local installation.
To run a local installation, you can download the file and install it.

If you are coding your .s file on the (non-mac) instructional computers with no intent of running or debugging it, please use another (non-java) editor such as emacs, vi or pico.

After starting mars, you can load your .s file using File->Open. You can edit your code by clicking the "Edit" tab and run or debug your program by clicking the "Execute" tab. In order to be able to execute your program, you must assemble the code first using Run->Assemble (F3).

To debug your assembly code in MARS you can set breakpoints, step through instructions one by one, view what are in your registers or in memory, as well as initialize values in registers before your program starts. You should spend some time before the lab getting familiar with MARS if possible.

Exercises
Setup

Exercise 1: Familiarizing yourself with MARS
Load lab1_ex1.s into MARS and assemble the code. Assume that fib[0] = 0; fib[1] = 1; fib[n] = fib[n-1] + fib[n-2]
Use Help (icon with the question mark) in order to answer the following questions about how

to use MARS.

What do the .data, .word, .text directives mean (i.e., what do you put in each section)?
How do you set a breakpoint in MARS? Set breakpoint on line 15.
After your program stops because of a breakpoint, how do you continue to execute your code? How do you step through your code?
How can you find out the contents of a register? How do you modify the value of a register.
At what address is n stored in memory? Calculate the 13th fib number by modifying this memory location.
line 18 and 20 use syscall instruction. What is it and how do you use it? (hint: syscall inside Help!))
Exercise 2: A short MIPS program
Write a piece of MIPS code that, given values in $s0 and $s1, put into the $t? registers 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, it stores the sum of the previous two $t? register values. The $s0 and $s1 registers contain the initial values.

Don't set the value of $s0 and $s1 in your code. Instead, learn how to set it manually with MARS.

Save your code in a file called lab1_ex2.s.
Exercise 3: Debugging a MIPS program
Debug the loop in the program in lab1_ex3.s. It is meant to copy integers from memory address $a0 to memory address $a1, until it reads a zero value. The number of integers copied (up to, but NOT including the zero value), should be stored in $v0.

Describe in a file lab1_ex3.txt the bug(s) of the code. Create a file lab1_ex3_ok.s that contains a bug fixed version of lab1_ex3.s.
Exercise 4: Compiling from C to MIPS
The file lab1_ex4.c is a C version of the program in the previous part of the lab.
Compile this program into MIPS code using the command:
$ mips-gcc -S -O2 -fno-delayed-branch lab1_ex4.c -o lab1_ex4.s
On the Macs, type exit to log out of the remote machine.
The -O2 option (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 lab1_ex4.c are originally stored in the assembly file. Finally, explain how these pointers are manipulated through the loop.

Find the section of code in lab1_ex4.s that corresponds to the copying loop and explain how each line is used in manipulating the pointer.