Introduction to Computer Engineering (E114)

Lab 6: MIPS Assembly Language

Introduction

In this lab, you will be writing an assembly language program that computes **Fibonacci** numbers. The basic MIPS assembly language instructions should be familiar to you after reading Chapter 3 of your book. Refer to section A.10 for a complete list of all the available instructions along with descriptions of their functions.

Recall that each number in the Fibonacci series is the sum of the previous two numbers. Table 1 lists the first few numbers in the series.

n	1	2	3	4	5	6	7	8	9	10	11	…
fib(n)	0	1	1	2	3	5	8	13	21	34	55	…

Table 1: Fibonacci Series

We can also define the fib function for negative values of n. To be consistent with the definition of the Fibonacci series, what would the following values be?

fib(0) = ____

fib(-1) = ____

These values are useful when writing a loop to compute fib(n) for all non-negative values of n.

To execute the fib program that you write, you will be using the **PCSpim** application. PCSpim allows you to step through one line of code at a time while displaying the values of all registers and memory, so that you can watch what the program is doing and catch any bugs that appear. PCSpim is available in the E114 directory on Engserve1. Therefore, you can do this lab either from a home PC or from any PC cluster that can connect to Engserve1 on the Network Neighborhood.

1. Writing the Code

To get you started, here is a template for the assembly program that you should write:

```
main:   addi $a0,$0,8          # n = 8
        addi $t0,$0,-1         # jump-start loop with
        addi $t1,$0,1          #   fib(-1) and fib(0)
loop:   <add your code here>   # <your comments here>
        . . .
done:   j done                 # infinite loop
```

Use Notepad (or your favorite text editor) to enter this code into a text file named lab6_xx.asm (where 'xx' are your initials).

Fill in the missing lines of code to complete the loop that computes fib(n). You should be able to do this in fewer than 10 lines of code. Use only the instructions that your Lab 2 controller handles (ADD, SUB, AND, OR, SLT, ADDI, J, BEQ, LW, SW) because you will use your program to test your processor when you build it in a later lab. Remember that SUB does not take a constant operand. However, you may subtract a constant by using ADDI with a negative value. You may find the summary of registers on page A-22 of your book to be a handy reference.

Note that the first line of code places the value 8 into register A0. This is the 'n' argument to the fib function. You should test your program on several different values (not just 8).

The third and fourth lines of code load the values of fib(-1) and fib(0) into registers T0 and T1, respectively. As mentioned above, these can be used to "bootstrap" your loop.

After looping the number of times specified by the value placed in register A0, your code should leave the resulting value of fib(n) in register $v0 and branch to `done`. The last line of code is simply a way of ensuring that your program terminates in a predictable manner.

Comments begin with the '#' character and continue to the end of the line. You should use comments to include your name and the date at the top of the file. Please add comments after **every** line of code also, explaining clearly what the code does.

2. Running the Code in PCSpim

Once you have a complete assembly file, you are ready to test your code using the MIPS simulator. E114\PCSpim directory on Engserve1 and start PCSpim now.

Since this is the first time you are running Spim, you will need to choose the correct simulator settings. Choose Simulator•Settings from the menu and make sure that your settings match those shown in Figure 1. simple.trap is a "Trap Handler" that loads your program starting at memory address 0x00400000.

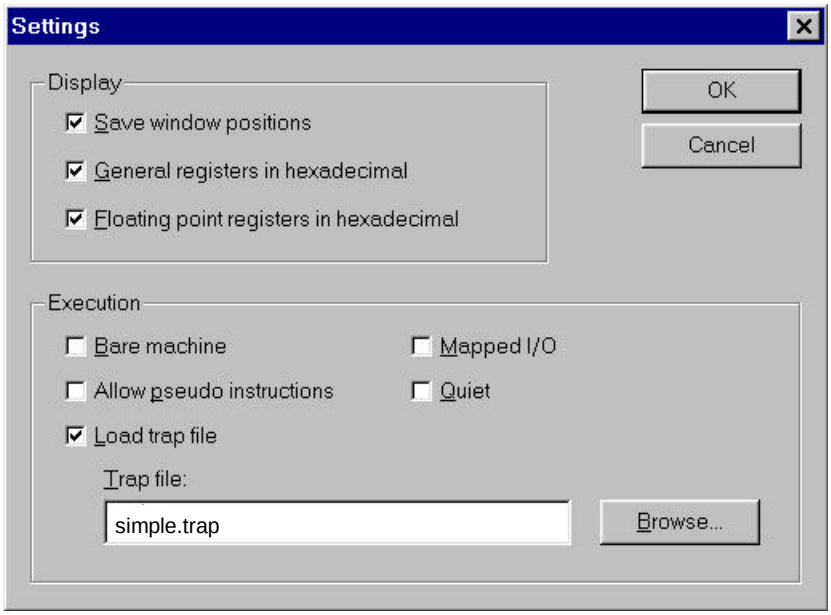

Figure 1: PCSpim Settings Window

You are now ready to load your program into PCSpim. You can do so by selecting File•Open from the menu, or using the leftmost toolbar button. If you can't find the file, you may have named your file with an extension other than .asm; use the view All Files option in the Open dialog to see files with other extensions. Your window should now look similar to the one in Figure 2, after you select Window•Tile from the menu.

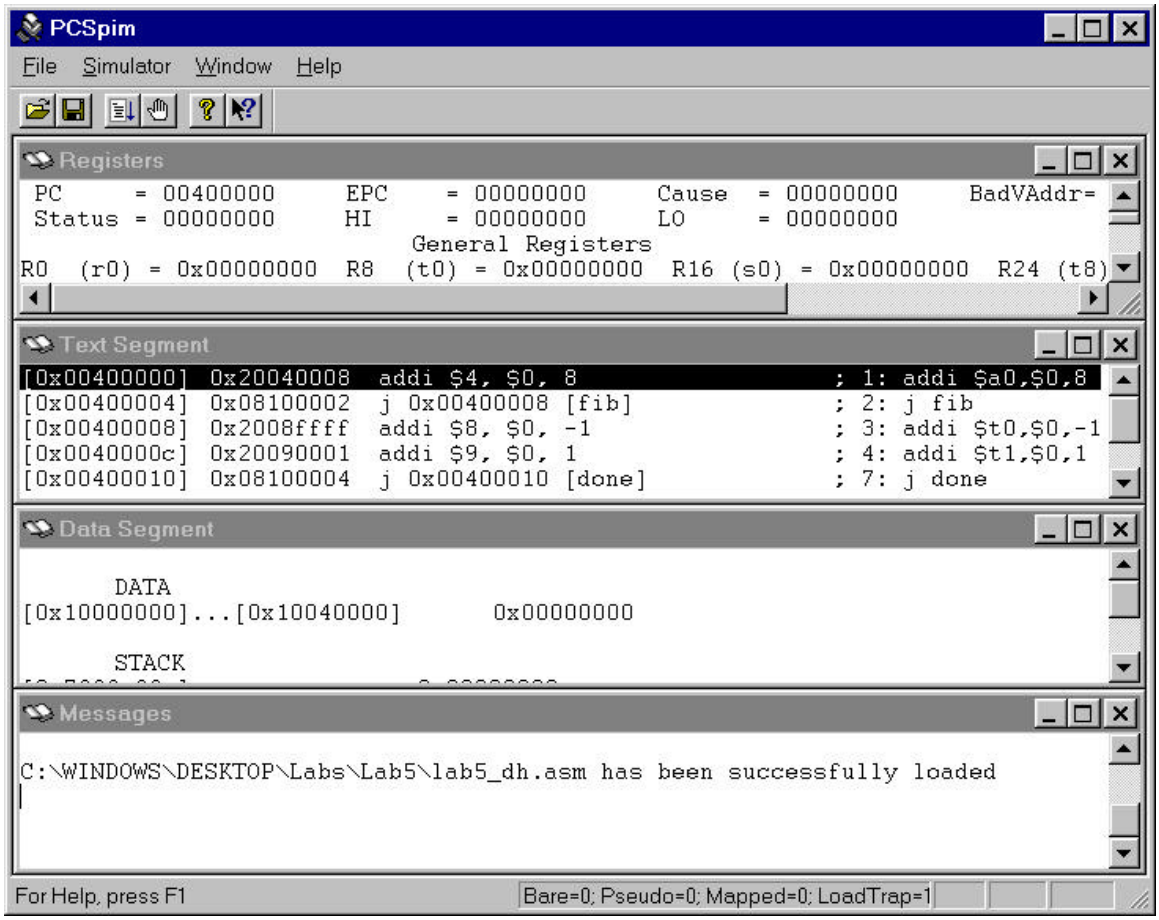

Figure 2: PCSpim

Take a moment to familiarize yourself with the PCSpim interface. There are four inner windows that provide information about a program while it is running:

- The "Registers" window displays the current contents of all registers. **Note that values are shown in hexadecimal**, as you specified in the settings. This information is very useful in debugging programs. Also, note that numbers are represented in 2's complement. Therefore, -1 is FFFFFFFF.
- The "Text Segment" window displays the contents of the instruction memory. You can see each line of your code here, along with the hexadecimal machine code that it assembled to, and the address is resides in. This hex code is the actual content of the instruction memory that is executed directly by the processor. The line of code that is highlighted is the next instruction to be executed (specified by the value in the PC register).
- The "Data Segment" window shows the contents of data memory. Since your code for this lab shouldn't need any `lw` or `sw` instructions, the contents of this window are not relevant.
- The "Messages" window displays status and error messages, and records each line of code as it executes (this provides a complete history of program execution, even when there are loops).

For more information, consult Section A.9 of Patterson & Hennessy or refer to PCSpim's online help by pressing F1.

You are now ready to begin executing your program. **Since your program ends with an infinite loop, be careful not to use the "Go" or "Continue" commands** (found in the Simulator menu, and on the toolbar). Instead, you should use the Simulator•Single Step command to step through your program one line at a time. This command is also mapped to the F10 key for your convenience.

If for some reason you want to run your entire program in one step, you can also use the "Multiple Step" command and specify a large number of steps.

Go ahead and run your program now, until it reaches the infinite loop at the end. Does it leave the correct value in register $v0? Try changing and re-running the program to compute fib(n) for several different values of n. Try this until you are convinced that your program can compute fib(n) for any positive value of n.

Note that some assembly language instructions can generate exceptions (errors) if written incorrectly. If you notice that the simulation seems to get stuck on the same line of code no matter how many times you step, check the Messages window for exception messages. If your program generates exceptions, you should try to change your code to fix the reported exception.

The File•Save Log File feature allows you to save the contents of all four inner windows into a single text file. Change your program again so that it computes fib(8), and reload it into PCSpim. Run the program through once, and then save a log file to turn in. The log file should show the correct value of fib(8) in register $v0, and the Messages area at the top of the file should show the instructions that executed when the program ran. You can open, view, and print the log file from your favorite text editor, such as notepad.

What to Turn In

Provide a hard copy of each of the following:

1. Please indicate how many hours you spent on this lab. This will not affect your grade, but will be helpful for calibrating the workload for next semester's labs.
2. Your completed, thoroughly commented fib.asm file.
3. Your PCSpim log file, which shows how your program leaves the correct value in register $v0.
4. Extra Credit: Write an assembly language program to compute the factorial function: fact(n) = n!. Use only the instructions listed on page 2; notice, in particular, that you may not use mul. Turn in a copy of the code and the PCSpim log showing how your program computes fact(4) = 24. This extra credit is worth one problem set problem.