

Laboratory 10 MIPS and Spim

Computer Science 240

The goal of this laboratory is experiment with the MIPS assembly language and operating environment, **PCSpim**. You will be using these to write programs for the rest of the semester.

Exercise 1: Log in to FirstClass. Go to the Lab Materials folder, and download the **multiply.asm** and **bugs.asm** programs.

Start PCSpim, and pull down the **Window** menu. Make sure that both **Toolbar** and **Status Bar** are checked.

Open the multiply.asm file from the **File** menu. This is a source file that contains the text of a MIPS assembly language program which is a solution for the lab assignment for today.

Load Program

Opening the source file in PCSpim assembles it and, if there are no errors, loads an executable version into the simulator.

Run Program

To run the program, select **Go** (alternately, select the Go icon or hit the F5 key). When the **Run Parameters** dialog box pops up, click **OK** to select 0x0040000 (by default) as the starting address.

When you are prompted for two inputs in the Console window, enter a value of 2 and 3, respectively, and observe the result.

Save Log File

In the PCSpim window, select **Save Log File...** from the **File** menu. When the **Save As** Dialog box pops up, save the **PCSpim.log** file on your desktop.

Double-click on the PCSpim.log file (it should open in Notepad).

Print the PCSpim.log file, and circle and label the following on the hardcopy. Attach to your lab report at the end of lab.

1. Find the listing of the Registers. Circle and label final values of R8,R9,R10. What do these represent?
2. Circle and label Console output.
3. Find the Text Segment. The default starting address of the program is 0x400000. However, notice that the first instructions are not from your program; they are produced by the kernel. Find the first line of your program, and circle and label the address.
4. Circle and label the instruction produced by the assembler at this address. Is it the same as the instruction from your program? Explain why it does the same thing as your instruction:
5. Circle and label **op1**, **op2**, and **result** in the Data segment.
6. Circle and label the **prompt1** and **prompt2** strings in the Data segment.

Reload Program

In PCSpim, run your program again by selecting **Reload** from the **Simulator** menu.

You normally reload your program if you have made changes to it, and want to try it again. Click on **Yes** when PCSpim asks if you want to clear program and reinitialize the simulator.

You can then select **Go** to run your program again.

Syntax Errors

Exercise 2: Open the bugs.asm in PCSpim file. The file contains some syntax errors, and a dialog box will pop up with error messages. Use the error messages listed (the most useful information is the line number) to go back to the source file and fix the errors (the **Goto...** option in the **Edit** menu of NotePad is useful for this purpose). NOTE that the first instruction in the program is:

```
main: lw $v0,4
```

There is a problem with this instruction, but it is not a syntax error (so don't fix it!). Later, it will generate a run-time error which is useful to observe as a debugging exercise (you will get a chance to fix it at that point).

Debugging

Exercise 3: When you are finally able to load without errors, select **Go** to run your program. You should see the Console pop up, but it will display an error message:

```
Exception 7 [Bad data address] occurred and ignored
```

This exception is generated on a Bus Error on a Load or Store. You cannot be sure which instruction in your program is causing this error (although it is probably a load or store!), so you need to use the simulator commands to pinpoint the source of the error.

In the PCSpim window, restart your program by selecting the **Reload** option from the **Simulator** menu.

Single-Step Execution

Execute one instruction at a time by repeatedly selecting **Single Step** from the **Simulator** menu, or by pressing the **F10** key. Every time you do this, PCSpim will execute one instruction and update its display, so that you can see what the instruction changed in the registers or memory.

The first instructions starting at [0x400000] are not from your program – step past them. However, when you execute the first instruction from your program (`lw $v0,4`), examine the Message pane, and notice the message there:

Exception occurred

Bad address in data/stack read: 0x00000004

Examine the first instruction in your program carefully. What is causing this exception?

Go back to NotePad, fix the error, save the file, and return to PCSpim. **Reload** your program, and select **Go** to run it again. You should see the Console pop up, with the two prompts for inputs. Enter '2' and '3', as before.

Notice that your result is being displayed as '0'. You know this is incorrect, but why?

It appears that your prompts are being output correctly, but how do you know whether your inputs were read in from the console?

Setting Breakpoints

What do you do if your program runs for a long time before the bug arises? You could single-step until you get to the bug, but that can take a long time, and it is easy to get so bored and inattentive that you step past the problem. A better alternative is to use a **breakpoint**, which tells PCSpim to stop your program immediately before it executes a particular instruction.

If you set a breakpoint in this program following the input of the values, you can check whether they are read in and stored in memory as you would expect. Begin by reloading your program.

Examine the Text pane in PCSpim, and note the address of the instruction following the one which stores the second operand into register \$t2 after the value is input:

```
                                move $t2,$v0  
[note this address]         li $t0,0
```

Set a breakpoint at this address by selecting the **Breakpoints...** option from the **Simulator** menu. When the Breakpoints dialog box pops up, enter the address, and click on **Add**. Then, click on **Close**.

If you wish to set a breakpoint at a line with a label, you can simply type the label into the box, rather than determining the actual address associated with the label.

Also, if you wish to remove a breakpoint, you can do so by selecting it in the Breakpoints dialog box, and clicking on **Remove**.

Execute the program by selecting **Go**. When prompted in the Console, enter '2' and '3' as the inputs. At this point, a PCSpim dialog box will pop up, notifying you that a breakpoint was encountered, and asking whether to continue execution. Click on **No**.

If you examine the program, you notice that the first two items declared in the data section are **op1** and **op2**, and these are words, so use 4 bytes of storage. Examine the DATA pane. You see that the Data section starts at address [0x10010000]. The DATA pane shows the following:

```
[0x10010000]  0x00000002 0x00000003 0x00000000 0x61656c50
```

What do each of the 4 words displayed represent?

Do you think that the inputs from the Console were read in correctly?

Record the value of the next 4 words in memory, starting at [0x10010010]:

What are these?

It seems that there must be a problem with the calculation of the result, which occurs in the loop of the program. Continue execution of your program by single-stepping each instruction, and examine the values of registers \$t0 - \$t3 in the Registers pane after each step, to see if you can determine why the result is incorrect.

What instruction is incorrect? How can you fix it?

Select **Continue** from the **Simulator** menu to continue execution of your program to the end. Fix the faulty instruction in NotePad, save the file, reload it to PCSpim,

Multiple-Step Execution

Sometimes, you simply want to execute a certain number of instructions before examining registers or memory for debugging purposes. You can do this by selecting **Multiple-Step...** from the **Simulator** menu. Do so now, and when the Multiple Step dialog box pops up, enter **24** as the number of steps, and click on **OK**.

The Console window will prompt you for the two inputs (again, enter '2' and '3'), and will then pause execution, in the same place you did previously, by setting a breakpoint. Examine the Text pane to verify that this is the case.

Set Value

Sometimes in the process of debugging, you wish to set the value of a register or memory location during simulation of a program. You can do this with the **Set Value** option from the **Simulator** menu. Select it now, and the Set Value dialog box pops up, asking you for an **Address** or **Register Name** and a **Value**.

Modify register **\$t1** to a value of **4**, and click on **OK**.

Select **Continue** from the **Simulator** menu. What result is displayed in the Console window? Why?

Reload the program. Now, set a breakpoint which will cause the program to pause right before the result is displayed – to determine the breakpoint address, examine the text pane, and find the address of the code directly before the output of the result.

When the program is paused, select **Set Value** from the **Simulator** menu, and modify address **0x10010008** to value **100**, and click on **OK**.

Perform one **Single-Step**, and then **Continue**. What is the result, and why?

Stopping your program

If you want to stop your program (for example, if it seems to be stuck in an infinite loop!), go to the **Simulator** menu and click on **Break**. This causes PCSpim to pop up a dialog box with two buttons.

Before clicking on either button, you can look at registers and memory to find out what your program was doing when you stopped it. When you understand what happened, you can either continue the program by clicking on **Yes** or stop your program by clicking on **No**.

Modify the current program to make the loop infinite by going to NotePad and changing the instruction at the loop label to:

```
loop: beq $t1,$t2,end
```

Why does this make the loop infinite?

Save the program, reload it to PCSpim, and run it with the input values of '2' and '3' again. What happens?

Select **Break** from the **Simulator** menu (the Stop Sign icon will also accomplish this).

When the dialog box asks if you wish to continue execution, click on **No**. Notice what instruction the break occurs on. Does this make sense?

You have now seen what commands are available in PCSpim to assist you in simulating and debugging your programs. You can use the notes from lab 10 as a handy reference to these commands.

Exercise 4: Consider the following program:

```
.text
.globl main

main: la $s0,foo
      la $s1,boo
      la $s2,goo
      la $s3,moo
      lb $t0,3($s1)
      add $s1,$s1,$t0
      lb $t1,0($s1)
      addi $s1,$s1,2
      add $s2,$s1,$t1
      sb $t0,0($s2)
      addi $s1,$s1,7
      add $a0,$s1,$t0
      li $v0,4
      syscall
      lw $t0,0($s3)
      sh $t0,-5($s2)
      li $v0,10
      syscall

.data
foo: .byte -1
     .align 2
boo: .word 0x02030405

goo: .word 02,03

moo: .asciiz "abc"

zoo: .half 10,11,12,13,14,15
```

How many bytes are allocated in the data section, from the starting address foo to the end of the data stored at zoo (count any extra bytes allocated by the data directives)?

Assume that **foo** represents address 0x10010000, and that all unspecified bytes in memory contain 00.

Note that the hexadecimal value for the ascii character 'a' is 0x61, 'b' is 0x62, and 'c' is 0x63.

List the values of the bytes stored in the data section, in *hexadecimal*, 4 word on each line (similar to PCSpim):

[10010000]

[10010010]

Fill in the following table to show the contents of registers s0-s3 and t0-t1 after each instruction is executed. Only fill in the entries that change for each instruction. If a value stored in memory is modified by an instruction, also record that address and its new value on that line.

| | <u>\$s0</u> | <u>\$s1</u> | <u>\$s2</u> | <u>\$s3</u> | <u>\$t0</u> | <u>\$t1</u> | <u>address:</u> | <u>value</u> |
|--------------|-------------|-------------|-------------|-------------|-------------|-------------|-----------------|--------------|
| init values= | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | | |

la \$s0,foo

la \$s1,boo

la \$s2,goo

la \$s3,moo

lb \$t0,3(\$s1)

add \$s1,\$s1,\$t0

lb \$t1,0(\$s1)

addi \$s1,\$s1,2

add \$s2,\$s1,\$t1

sb \$t0,0(\$s2)

addi \$s1,\$s1,7

add \$a0,\$s1,\$t0

li \$v0,4

syscall

lw \$t0,0(\$s3)

sh \$t0,-5(\$s2)

What should be displayed in the Console window at the end of the program?

Show the value of the bytes stored in the data section after the program is done executing (in hexadecimal):

[10010000]

[10010010]

Now execute the program **lab10.asm** line by line (single-step). Verify that you correctly predicted the register values, contents of memory, and program output.

Demonstrate to the instructor.

Exercise 5: Modify the **multiply.asm** program so that it is a loop. After each calculation, prompt the user for an answer of 'y' or 'n' if they want to multiply again. If 'y', repeat the loop. If 'n', exit the program.

Demonstrate your program to the instructor, and attach a hardcopy of the source file to your lab report.