PCSPIM TUTORIAL

VE370 Intro to Computer Organization

DOWNLOAD AND INSTALL

1. Download PCSpim (file PCSpim_9.1.4.zip)

   This tutorial assumes you are using PCSpim version 9.1.4 on Windows 7, which is the same version installed in the laacb m hines.

2. Unzip PCSpim_9.1.4.zip and run setup.exe to install it.

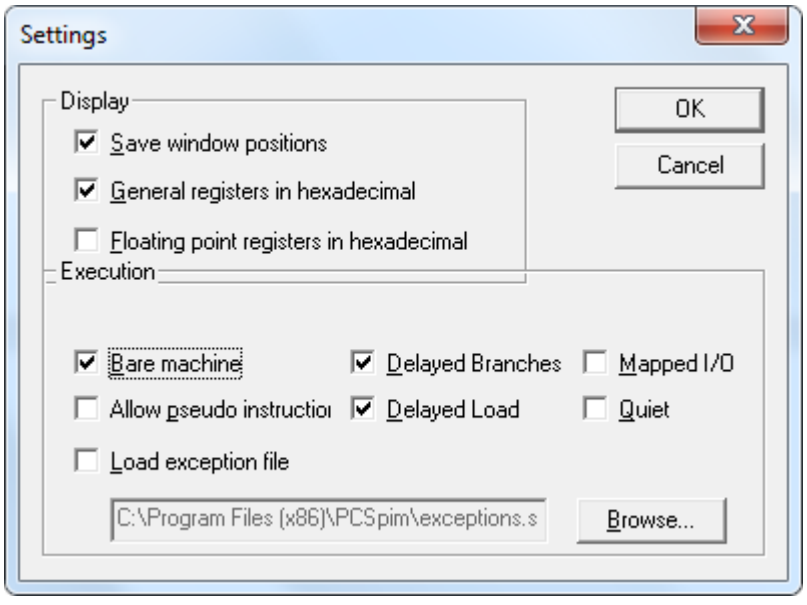
CONFIGURATION

1. Start PCSpim (hereafter shortened to "Spim")

   Note for 64-bit users: The first time you run Spim it may complain about a missing exception handler (exceptions.s). If you see this message, open Settings, look under "Load exception file", and change the path to the following (or something similar):

   C:\Program Files (x86)\PCSpim\exceptions.s

2. In this class we will be emulating a real processor ("bare machine"), which includes delayed branches and delayed loads. Open Settings from the main menu (Simulator->Settings…) and configure it as shown below:



Note: It is critical that you set the machine exactly as above. If you don't, you will see weird behavior, or at the very least we may not be able to run your code.

Now we're ready to write our first program!

RUNNING YOUR FIRST PROGRAM

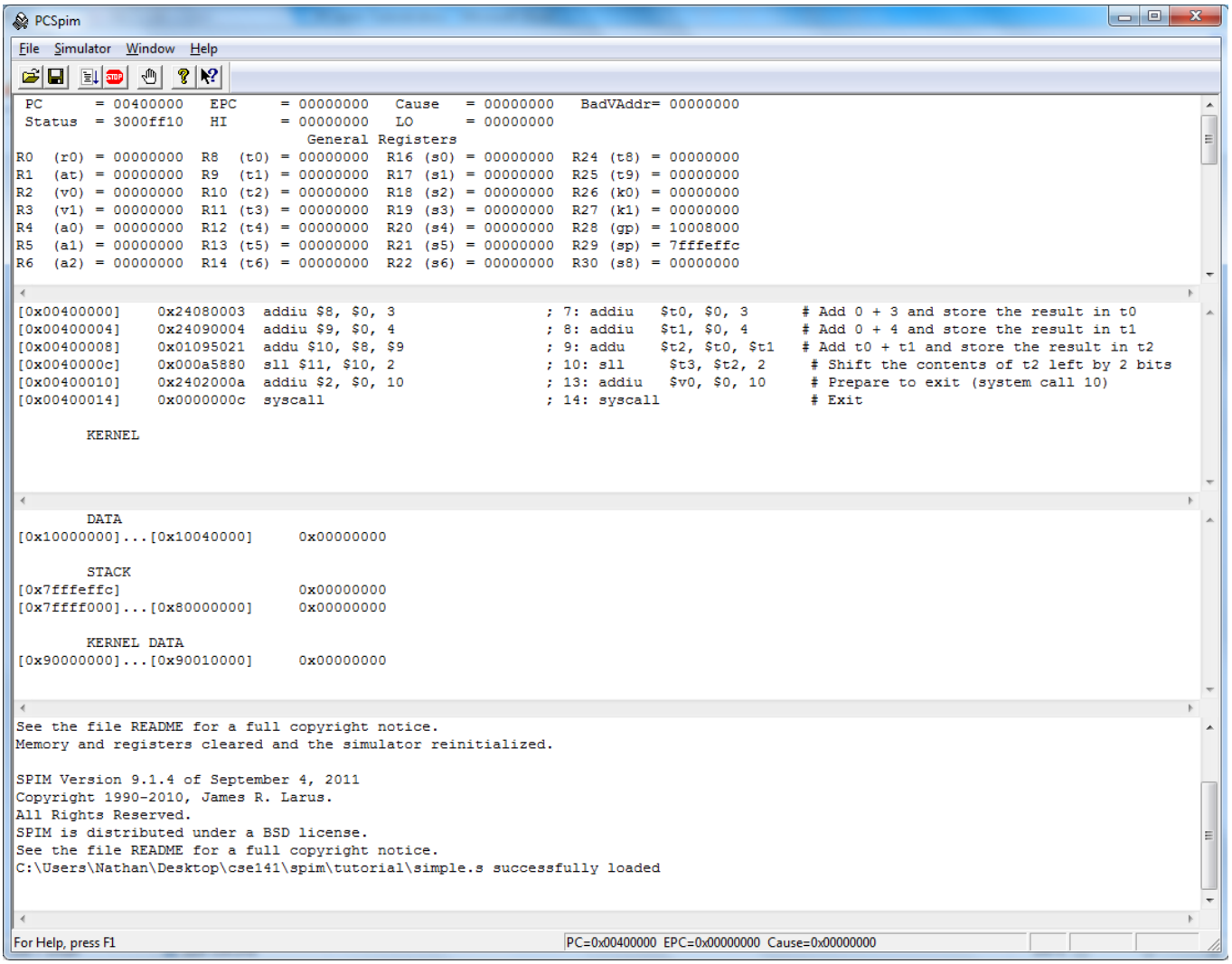
1. Open your favorite* text editor (*as a computer scientist, this should be Vim or Emacs)

2. Enter the following MIPS assembly code, and save it as simple.s:

```
#simple.s
 .text
 .globl __start
__start:
# Some simple arithmetic
addiu $t0, $0, 3 # Add 0 + 3 and store the result in t0
addiu $t1, $0, 4 # Add 0 + 4 and store the result in t1
addu $t2, $t0, $t1 # Add t0 + t1 and store the result in t2
sll $t3, $t2, 2 # Shift the contents of t2 left by 2 bits
                # (same as multiply by 4)
# Exit
addiu $v0, $0, 10 # Prepare to exit (system call 10)
syscall # Exit
```

This program does some simple arithmetic and then quits. We'll use it to get more familiar with Spim.

3. Now open your program in Spim using File->Open. (If Spim asks you to clear the program and reinitialize the simulator, choose Yes.) Your screen should look like this:



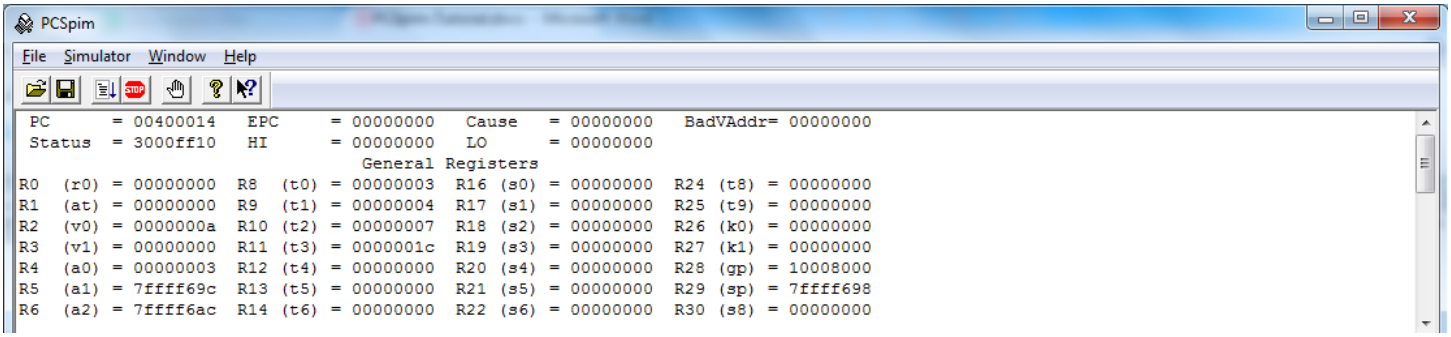
The main Spim window is split into four panes. From top to bottom:

- **Registers** – Shows the contents of every register. Scroll down to see floating point registers.
- **Text Segment** – Shows the code loaded into the simulator. Each line is a separate instruction, and from left to right the columns show: a) the address of the instruction, b) a hex representation of the machine code, c) the assembly representation, and d) your original source assembly code with line numbers.
- **Data Segment** – Shows the simulator's memory contents. Addresses are shown on the left, and the corresponding data are shown on the right.
- **Messages** – Useful information about what Spim is doing.

Spim also has a separate Console window for I/O; we'll talk more about it later.

4. Before we run our simple program, take a look in the Registers pane. Notice how the contents of most of the registers are set to zero. In particular, registers t0 through t3 are zero too.
5. Now run your program. You can press the Go button from the toolbar, or press the F5 key. Spim will ask you for a starting address. You should see 0x00400000 as the default, and notice in the Text pane that this corresponds to your first instruction (addiu). Press OK.

6. Your program will run and then stop when it reaches the exit syscall. Take a look at the contents of the registers again:



Note that the register contents are displayed in hex, so t3 holds our expected value of 0x1c = 28. Congratulations on running your first program! Next we'll take a look at using more advanced features of Spim.

DEBUGGING: STEPPING THROUGH A PROGRAM AND MODIFYING REGISTERS

In our first example we ran a program all the way from start to finish. In practice, you'll need to use the simulator to debug your programs, and an easy way to do this is to "step" through your program one instruction at a time.

1. Reload your program by choosing Simulator->"Reload C:\...\simple.s", and select Yes to reinitialize the simulator. Notice that General Registers are zero again, and that the PC register is back at 0x00400000. This tells you the next instruction that Spim will execute.

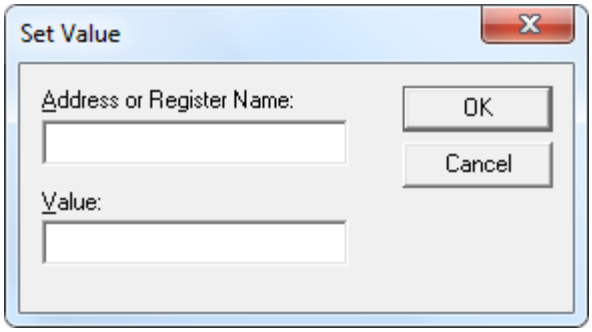
2. Instead of running your whole program all at once, you can step through it one instruction at a time. In the menu, select Simulator->"Single Step" (or press the F10 shortcut key on your keyboard), and look at the contents of the registers. The simulator has executed one instruction, register t0 now has the value 3, and the PC has been incremented to the next instruction.

3. You can keep pressing F10 to step through your program slowly.

4. You can also choose Simulator->"Multiple Step..." (or press F11) and specify a number of instructions to execute before pausing again.

When you are debugging a program, you may want to manually change the contents of certain registers or memory locations to see their effect on the program.

5. To do this in Spim, from the menu choose Simulator->"Set Value...":



6. You can specify a memory address (such as 0x10000000) or a register name (such as t0 or $8), along with the value to store. You can specify the value in decimal (default) or in hex by using 0x.

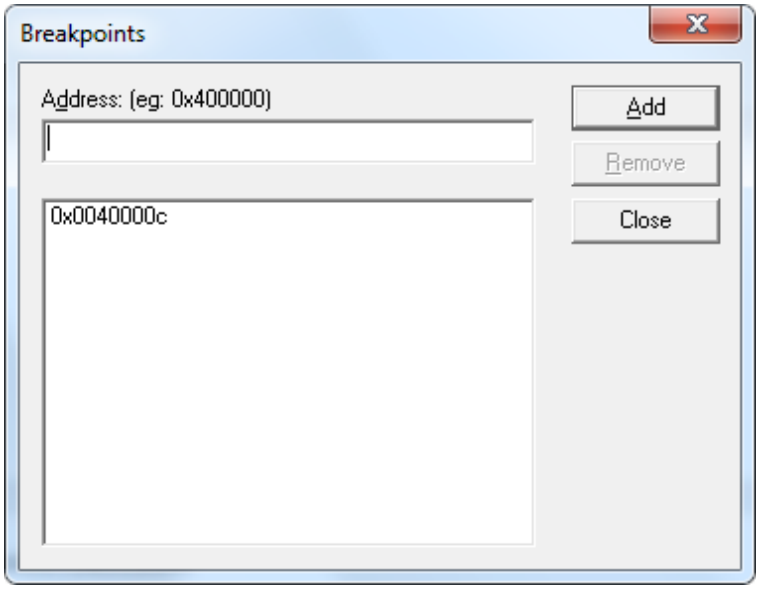
7. Extra credit: You can even use Set Value to rewrite the instructions in your text segment! But this is **very** tricky to get right and not recommended ☺

DEBUGGING: SETTING BREAKPOINTS

Manually stepping through a program can be tedious for long-running (i.e., "interesting") programs. To make things easier, you can specify a stopping point called a "breakpoint" in your code. The simulator will run until it hits a breakpoint, and then you can manually step forward or resume execution.

1. Reload our program again.

2. Let's set a breakpoint on our "sll" instruction. From the menu, choose Simulator->Breakpoints (or press Ctrl+B, or click the hand icon). Add the address of the sll instruction, which you can see in the Text Segment. Your Breakpoints should look like this:



After closing the Breakpoints box, you'll see an asterisk next to the sll instruction's address in the Text Segment window, indicating the breakpoint.

3. Now run your program using F5. When the simulator reaches your breakpoint, it will ask you if you wish to continue execution. From here you can choose Yes to resume execution, or No to pause execution where

you can step through instructions or examine registers as before. You can resume execution by pressing F5 once again.

Note that the breakpoint here wasn't very interesting, but breakpoints are very handy when debugging larger programs. You can set a breakpoint on important branches, at the start of a function or label, or anywhere you would like to pause execution and take a closer look around.

USING THE CONSOLE

The Spim Console lets you print things to the screen and read input values into your programs. This program demonstrates how to print and read integers using syscalls:

```
# console.s

    .text

    .globl __start

__start:

    # Print "7" to the console

    addiu   $a0, $zero, 7   # Load a0 with int to be printed

    addiu   $v0, $zero, 1   # Load v0 with syscall 1 = print_int
    syscall                 # Actually do the syscall

    # Read an int from the console

    addiu   $v0, $zero, 5   # Load v0 with syscall 5 = read_int

    syscall                 # read_int

    addu    $s0, $zero, $v0 # Move int to s0 to save it before exiting

    # Exit

    addiu   $v0, $zero, 10  # Prepare to exit (system call 10)

    syscall                 # exit
```

You can find many more system calls in Appendix B of your textbook.

That's it! You now have the basics to start writing and debugging your own programs.