Getting Started with `PCSpim`

If you are going to use `PCspim` on Microsoft Windows, this is the first section to read. If you are going to use `xspim` on UNIX, go back to the previous section and start there.

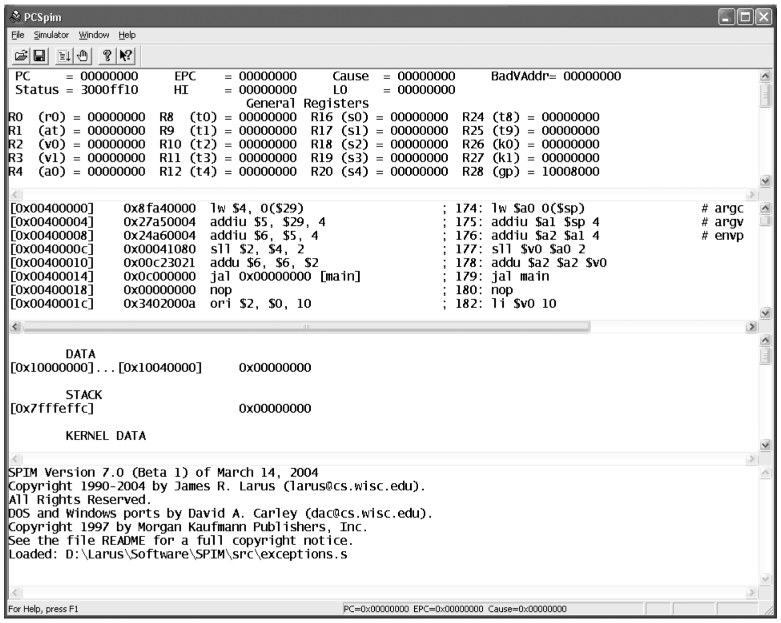
After you read this section, be sure to take a look at the "SPIM Command-Line Options" section to see how to accomplish the same thing with `spim` commands.

You may need to install `PCSpim` on your computer before you first run it. Your instructor should tell you where to get a copy of the program and how to install it.

When `PCSpim` is installed, it adds an entry to your start menu called `PCSpim`. Click on it and `PCSpim` will start running. When `PCSpim` starts, it pops up a large window on your screen (see Figure A.9.2). The window is divided into four panes:

- The top pane shows the values of all registers in the MIPS CPU and FPU. This display is updated whenever your program stops running.
- The next pane displays instructions from both your program and the system code that is loaded automatically when `PCSpim` starts running. Each instruction is displayed on a line that looks like

```
[0x00400000] 0x8fa40000 lw $4, 0($29)  ; 89: lw $a0, 0($sp)
```



```
 PC      = 00000000    EPC     = 00000000    Cause    = 00000000    BadVAddr= 00000000
 Status = 3000ff10     HI      = 00000000    LO       = 00000000
                           General Registers
R0  (r0) = 00000000  R8  (t0) = 00000000  R16 (s0) = 00000000  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000000  R17 (s1) = 00000000  R25 (t9) = 00000000
R2  (v0) = 00000000  R10 (t2) = 00000000  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 00000000  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 00000000  R12 (t4) = 00000000  R20 (s4) = 00000000  R28 (gp) = 10008000

[0x00400000]    0x8fa40000  lw $4, 0($29)              ; 174: lw $a0 0($sp)         # argc
[0x00400004]    0x27a50004  addiu $5, $29, 4           ; 175: addiu $a1 $sp 4       # argv
[0x00400008]    0x24a60004  addiu $6, $5, 4            ; 176: addiu $a2 $a1 4       # envp
[0x0040000c]    0x00041080  sll $2, $4, 2              ; 177: sll $v0 $a0 2
[0x00400010]    0x00c23021  addu $6, $6, $2            ; 178: addu $a2 $a2 $v0
[0x00400014]    0x0c000000  jal 0x00000000 [main]      ; 179: jal main
[0x00400018]    0x00000000  nop                        ; 180: nop
[0x0040001c]    0x3402000a  ori $2, $0, 10             ; 182: li $v0 10

        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7fffeffc]                    0x00000000

        KERNEL DATA

SPIM Version 7.0 (Beta 1) of March 14, 2004
Copyright 1990-2004 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
See the file README for a full copyright notice.
Loaded: D:\Larus\Software\SPIM\src\exceptions.s

For Help, press F1          PC=0x00000000  EPC=0x00000000  Cause=0x00000000
```

FIGURE A.9.2 SPIM's Microsoft Windows interface: `PCSpim`.

The first number on the line, in square brackets, is the hexadecimal memory address of the instruction. The second number is the instruction's numerical encoding, again displayed as a hexadecimal number. The third item is the instruction's mnemonic description. Everything following the semicolon is the actual line from your assembly file that produced the instruction. The number 89 is the line number in that file. Sometimes nothing is on the line after the semicolon. This means that the instruction was produced by SPIM as part of translating a pseudoinstruction into more than one actual MIPS instruction.

- The third pane displays the data loaded into your program's memory and the data on the program's stack.
- The bottom pane is where `PCSpim` writes messages. This is where error messages appear.

Let's see how to load and run a program. You can either go to the `File` menu at the top left of `PCSpim`'s window and click on `Open`, or you can click on the open icon right below. Either way, `PCSpim` opens up a familar dialog box, which lets you find the file that you want to load. After you've selected the file, click on the `Open` button and `PCSpim` will load the file. If you change your mind, click on the `Cancel` button and nothing will happen. When you click on `open`, `PCSpim` gets rid of the prompt window, then loads your program and redraws the screen to display its instructions and data. Now move the mouse to put the cursor over the scrollbar to the right of the second pane and move the bar down so you can find the instructions from your program.

To run your program, you can either go to the `Simulator` menu and click on `Go` or just click on the `Go` icon. Either way, `PCSpim` pops up a dialog box with two entries and two buttons. Most of the time, these entries contain the correct values to run your program, so you can ignore them and just click on `OK`. This button tells `PCSpim` to run your program. Notice that when your program is running, `PCSpim` does not update the register display pane because the registers are continually changing. You can always tell whether `PCSpim` is running by looking at this pane or the title bar at the top of `PCSpim`'s window.

If you want to stop your program, 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`.

If your program reads or writes from the terminal, `PCSpim` pops up another window called the *console*. All characters that your program writes appear on the console, and everything that you type as input to your program should be typed in this window.

Suppose your program does not do what you expect. What can you do? SPIM has two features that help debug your program. The first, and perhaps the most useful, is single-stepping, which allows you to run your program an instruction at a time. Open the `Simulator` menu and click on `Single Step`. Everytime 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.

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. In the `Simulator` menu, there is an entry called `Breakpoints`. Click on it, and the `PCSpim` program pops up a dialog box with an entry box and a list. Type into this box the address of the instruction at which `PCSpim` should stop. Or, if the instruction has a global label, you can just type the name of the label. Labeled breakpoints are a particularly convenient way to stop at the first instruction of a procedure. To actually set the breakpoint, click the `Add` button. The breakpoint will then appear in the list of active breakpoints. You can set as many breakpoints as you want. To remove a breakpoint, select it in the list, and click on `Remove`. After you have set your breakpoints, `Close` the dialog and run your program.

When SPIM is about to execute the breakpointed instruction, `PCSpim` pops up a dialog with the instruction's address and two buttons. The `Yes` button continues running your program, and `No` stops your program. Before you click on either button, you can examine the display to see the values in registers or memory or you can add or remove breakpoints.

Single-stepping and setting breakpoints will probably help you find a bug in your program quickly. How do you fix it? Go back to the editor that you used to create your program and change it. To run the program again, you need a fresh copy of `PCSpim`, which you get in one of two ways. Either you can exit from `PCSpim` by clicking on the `Exit` entry in the `File` menu, or you can just reload your program. When you reload your program, `PCSpim` will ask if you want to reinitialize the simulator. The answer is yes if you have changed your program and want to try it again.