


Getting Started with xspim

If you are going to use xspim on UNIX or Linux, this is the first section to read. If you are going to use PCSpim on Microsoft Windows, skip to the next 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.

To start xspim, type xspim in response to your system's prompt (%):

```
% xspim
```

On your system, xspim may be kept in an unusual place, and you may need to execute a command first to add that place to your search path. Your instructor should tell you how to do this.

When xspim starts up, it pops up a large window on your screen (see Figure A.9.1). The window is divided into five panes:

- The top pane is called the *register display*. It shows the values of all registers in the MIPS CPU and FPU. This display is updated whenever your program stops running.
- The pane below contains the *control buttons* to operate xspim. These buttons are discussed below, so we can skip the details for now.
- The next pane, called the *text segments*, displays instructions from both your program and the system code that is loaded automatically when xspim starts running. Each instruction is displayed on a line that looks like

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

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 next pane, called the *data and stack segments*, displays the data loaded into your program's memory and the data on the program's stack.
- The bottom pane is the *SPIM messages* that xspim uses to write messages. This is where error messages appear.

xspim		
	PC = 00000000 EPC = 00000000 Cause = 00000000 BadVaddr = 00000000 Status = 00000000 HI = 00000000 LO = 00000000	
	General registers	
Register display	R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000 R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (s9) = 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) = 00000000 R5 (a1) = 00000000 R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 00000000 R6 (a2) = 00000000 R14 (t6) = 00000000 R22 (s6) = 00000000 R30 (s8) = 00000000 R7 (a3) = 00000000 R15 (t7) = 00000000 R23 (s7) = 00000000 R31 (ra) = 00000000	
	Double floating-point registers	
	FP0 = 0.000000 FP8 = 0.000000 FP16 = 0.000000 FP24 = 0.000000 FP2 = 0.000000 FP10 = 0.000000 FP18 = 0.000000 FP26 = 0.000000 FP4 = 0.000000 FP12 = 0.000000 FP20 = 0.000000 FP28 = 0.000000 FP6 = 0.000000 FP14 = 0.000000 FP22 = 0.000000 FP30 = 0.000000	
	Single floating-point registers	
	Control buttons	<div style="display: flex; flex-wrap: wrap; justify-content: space-around;"> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">quit</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">load</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">run</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">step</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">clear</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">set value</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">print</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">breakpt</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">help</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">terminal</div> <div style="border: 1px solid black; padding: 2px 5px; margin: 2px;">mode</div> </div>
		Text segments
	Text segments	<pre> [0x00400000] 0x8fa40000 lw \$4, 0(\$29) ; 89: lw \$a0, 0(\$sp) [0x00400004] 0x27a50004 addiu \$5, \$29, 4 ; 90: addiu \$a1, \$sp, 4 [0x00400008] 0x24a60004 addiu \$6, \$5, 4 ; 91: addiu \$a2, \$a1, 4 [0x0040000c] 0x00041080 sll \$2, \$4, 2 ; 92: sll \$v0, \$a0, 2 [0x00400010] 0x00c23021 addu \$6, \$6, \$2 ; 93: addu \$a2, \$a2, \$v0 [0x00400014] 0x0c000000 jal 0x00000000 [main] ; 94: jal main [0x00400018] 0x3402000a ori \$2, \$0, 10 ; 95: li \$v0 10 [0x0040001c] 0x0000000c syscall ; 96: syscall </pre>
		Data segments
Data and stack segments	<pre> [0x10000000] ... [0x10010000] 0x00000000 [0x10010004] 0x74706563 0x206e6f69 0x636f2000 [0x10010010] 0x72727563 0x61206465 0x6920646e 0x726f6e67 [0x10010020] 0x000a6465 0x495b2020 0x7265746e 0x74707572 [0x10010030] 0x0000205d 0x20200000 0x616e555b 0x6e67696c [0x10010040] 0x61206465 0x65726464 0x69207373 0x6e69206e [0x10010050] 0x642f7473 0x20617461 0x63746566 0x00205d68 [0x10010060] 0x555b2020 0x696c616e 0x64656e67 0x64646120 [0x10010070] 0x73736572 0x206e6920 0x726f7473 0x00205d65 </pre>	
SPIM messages	SPIM Version 5.9 of January 17, 1997 Copyright (c) 1990-1997 by James R. Larus (larus@cs.wisc.edu) All Rights Reserved. See the file README for a full copyright notice.	

FIGURE A.9.1 SPIM's X-window interface: xspim.

Let's see how to load and run a program. The first thing to do is to click on the `load` button (the second one in the first row of buttons) with the left mouse button. Your click tells `xspim` to pop up a small prompt window that contains a box and two or three buttons. Move your mouse so the cursor is over the box, and type the name of your file of assembly code. Then click on the button labeled `assembly file` within that prompt window. If you change your mind, click on the button labeled `abort command`, and `xspim` gets rid of the prompt window. When you click on `assembly file`, `xspim` 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 left of the text segments, and click the left mouse button on the white part of this scrollbar. A click scrolls the text pane down so you can find all the instructions in your program.

To run your program, click on the `run` button in `xspim`'s control button pane. It pops up a prompt window with two boxes and two buttons. Most of the time, these boxes contain the correct values to run your program, so you can ignore them and just click on `ok`. This button tells `xspim` to run your program. Notice that when your program is running, `xspim` blanks out the register display pane because the registers are continually changing. You can always tell whether `xspim` is running by looking at this pane. If you want to stop your program, make sure the mouse cursor is somewhere over `xspim`'s window and type `control-C`. This causes `xspim` to pop up a prompt window with two buttons. Before doing anything with this prompt window, you can look at registers and memory to find out what your program was doing. When you understand what happened, you can either continue the program by clicking on `continue` or stop your program by clicking on `abort command`.

If your program reads or writes from the terminal, `xspim` 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. Click on the button labeled `step` and another prompt window pops up. This prompt window contains two boxes and three buttons. The first box asks for the number of instructions to step every time you click the mouse. Most of the time, the default value of 1 is a good choice. The other box asks for arguments to pass to the program when it starts running. Again, most of the time you can ignore this box because it contains an appropriate value. The button labeled `step` runs your program for the number of instructions in the top box. If that number is 1, `xspim` executes the next instruction in your program, updates the display, and returns control to you. The button labeled `continue` stops single-stepping

and continues running your program. Finally, `abort` command stops single-stepping and leaves your program stopped.

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 `xspim` to stop your program immediately before it executes a particular instruction. Click on the button in the second row of buttons marked `breakpoints`. The `xspim` program pops up a prompt window with one box and many buttons. Type in this box the address of the instruction at which you want to 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 on `add`. You can then run your program.

When SPIM is about to execute the breakpointed instruction, `xspim` pops up a prompt with the instruction's address and two buttons. The `continue` button continues running your program, and `abort` command stops your program. If you want to delete a breakpoint, type in its address and click on `delete`. Finally, `list` tells `xspim` to print (in the bottom pane) a list of all breakpoints that are set.

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 SPIM, which you get in two ways. Either you can exit from `xspim` by clicking on the `quit` button, or you can clear `xspim` and reload your program. If you reload your program, you *must* clear the memory, so remnants of your previous program do not interfere with your new program. To do this, click on the button labeled `clear`. Hold the left mouse button down and a two-item menu will pop up. Move the mouse so the cursor is over the item labeled `memory & registers` and release the button. This causes `xspim` to clear its memory and registers and return the processor to the state it was in when `xspim` first started. You can now load and run your new program.

The other buttons in `xspim` perform functions that are occasionally useful. When you are more comfortable with `xspim`, you should look at the description below to see what they do and how they can save you time and effort.