

# ECE 484/584

## Tutorial on Using QtSpim

QtSpim is software that will help you to simulate the execution of MIPS assembly programs. It does a context and syntax check while loading an assembly program. In addition, it adds in necessary overhead instructions as needed, and updates register and memory content as each instruction is executed. Below, is a tutorial on how to use QtSpim.

### Where to Get QtSpim?

Download the source from the SourceForge.org link at:

<http://pages.cs.wisc.edu/~larus/spim.html>

(See "New versions of spim" text in red at the top of the page.)

Alternatively, you can go directly to:

<http://sourceforge.net/projects/spimsimulator/files/>

- Note that versions for Windows machines, Linux machines, and Macs are all available

### Important Documents to Read :

Kindly make a point to read the below documents before starting, from the Appendix A of the third edition of Hennessy & Patterson, *Computer Organization and Design: The Hardware/Software Interface*. This documentation is far more complete and up-to-date than the documentation included in the *spim* distribution. :

*Assemblers, Linkers, and the SPIM Simulator* ([PDF](#)). An overview and reference manual for *spim* and the MIPS32 instruction set.

*Getting Started with spim* ([PDF](#)). Overview of the console version of *spim* (both Unix and Windows).

*Getting Started with xspim* ([PDF](#)). Overview of the X-windows version of *spim*.

*Getting Starting with PCSpim* ([PDF](#)). Overview of the Microsoft Windows version of *spim*.

*SPIM Command-Line Options* ([PDF](#)). Overview of the command line options of *spim* (all versions).

### **SPIM in action**

When you open QtSpim, A window will open as shown in Figure 1. The window is divided into different sections:

1. The *Register* tabs display the content of all registers.
2. Buttons across the top are used to load and run a simulation

3. The *Text* tab displays the MIPS instructions loaded into memory to be executed. (From left-to-right, the memory address of an instruction, the contents of the address in hex, the actual MIPS instructions – where register numbers are used, the MIPS assembly that you wrote, and any comments you made in your code are displayed.)

4. The *Data* tab displays memory addresses and their values in the data and stack segments of the memory.

5. The *Information Console* lists the actions performed by the simulator.

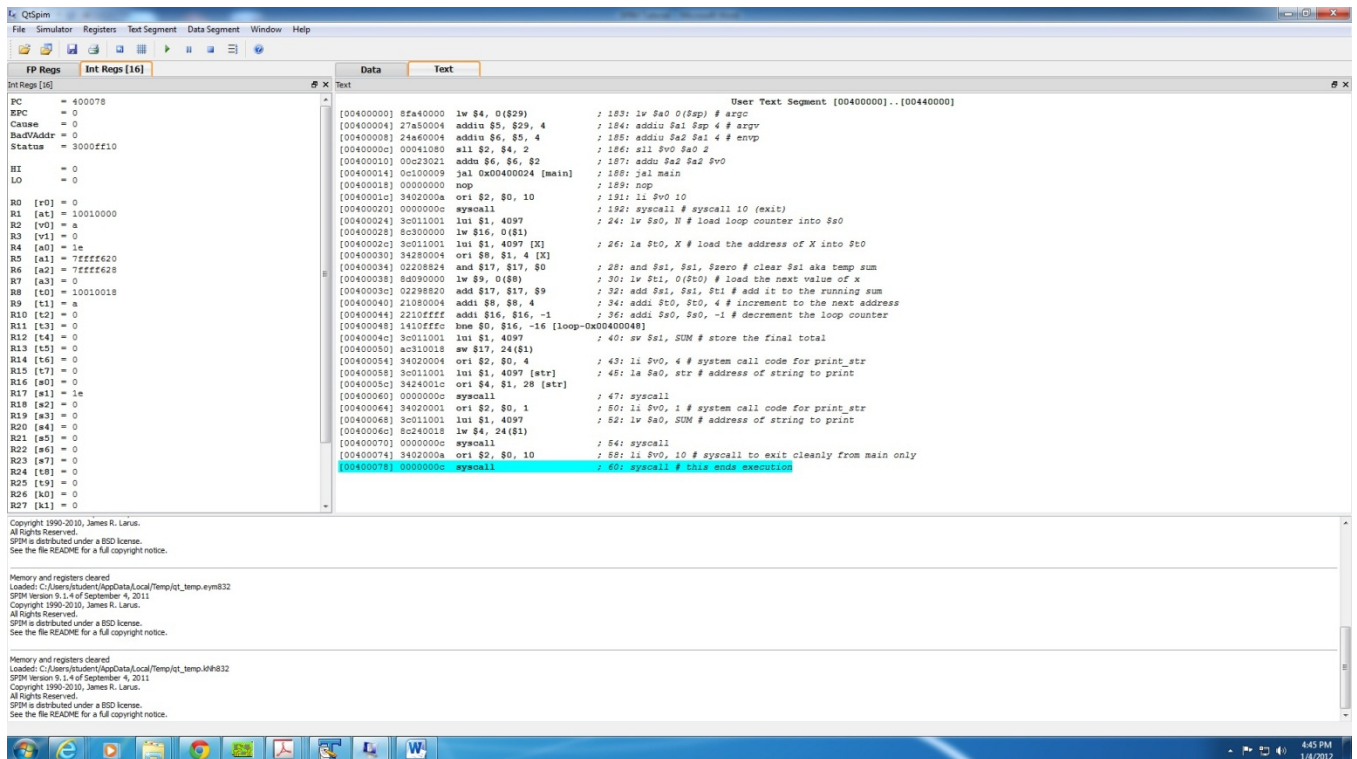


Figure 1 : QtSpim

## To run the program in QtSpim:

1. Use a text editor to create your program yyyyyy.s
2. Click on the “load” button and open yyyyyy.s
3. You can then run the program by simply pressing the “run” (play) button – all instructions will be executed, and the final contents of memory and the register file will be reflected in the QtSpim window.

## Debugging

Suppose your program does not do what you expect. What can you do? QtSpim 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. The single stepping icon can be found in the toolbar. Every time you do single stepping, QtSpim 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. A better alternative is to use a *breakpoint*, which tells QtSpim to stop your program immediately before it executes a particular instruction. When QtSpim is about to execute the instruction where there is a breakpoint, it asks for continue, single stepping or abort.

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. Click on the Rinitialize simulator tab in the toolbar and load the sourcefile again.

## Generally Useful Information

When using QtSpim, you may find the following information to be useful:

You can access all of the commands via the “File” and “Simulator” menus as well.

When examining register or memory data, you can view the data in binary, hex, or decimal format. Just use the “Register” pull down menu to select.

Kernel Text and Kernel Data may not be necessary to be viewed all the times, you can unselect them by unselecting “Kernel Text” in the “Text Segment” pull down menu and unselecting “Kernel Data” in the “Data Segment” pull down menu.

You can set breakpoints in your code simply by right clicking on an instruction in the Text tab.

To view memory data, simply click on the Data tab.

By right clicking on a register file value or memory address value, you can change its contents dynamically.

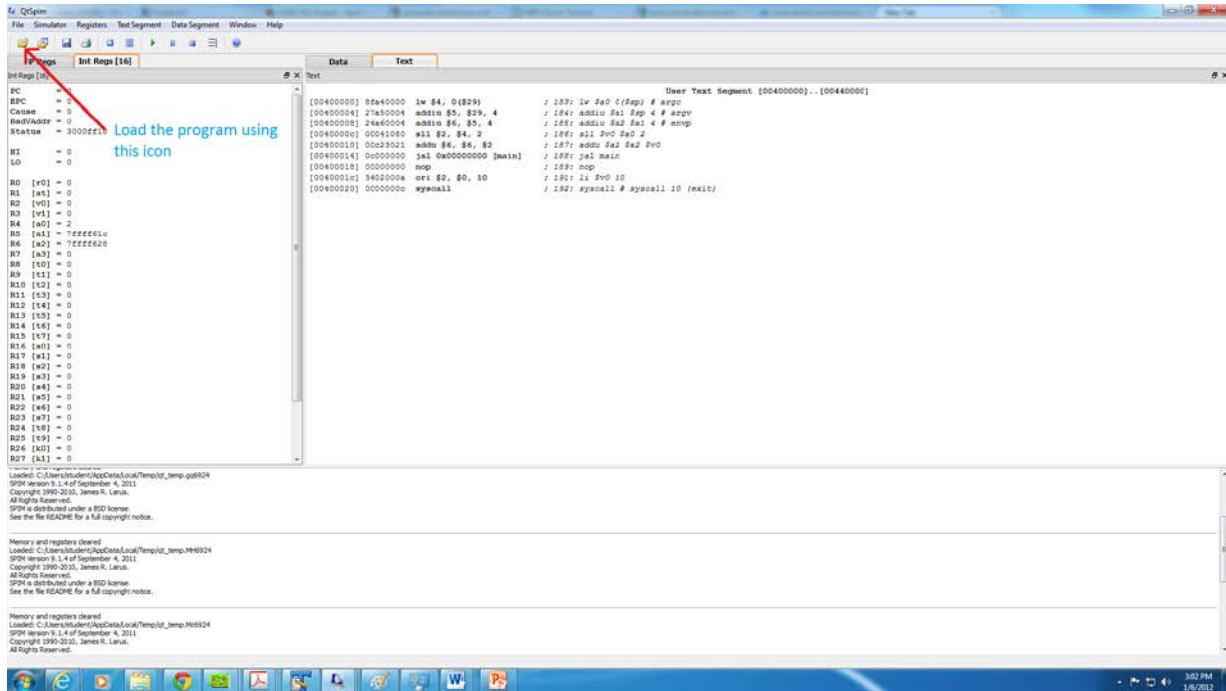
## Example Program

Below is an example program to find the sum of an array. Copy this into a text editor and save it as a .s file and open it in QtSpim by loading the file. You can directly run it or do single stepping and observe the change in the Register file. At the end of the Program you should be able to see the result stored in S1 as “1e” ( $2+4+6+8+10 = 30 = 0x1e$ ) and the console will print this result. The code is well commented which should help you start straight away.,

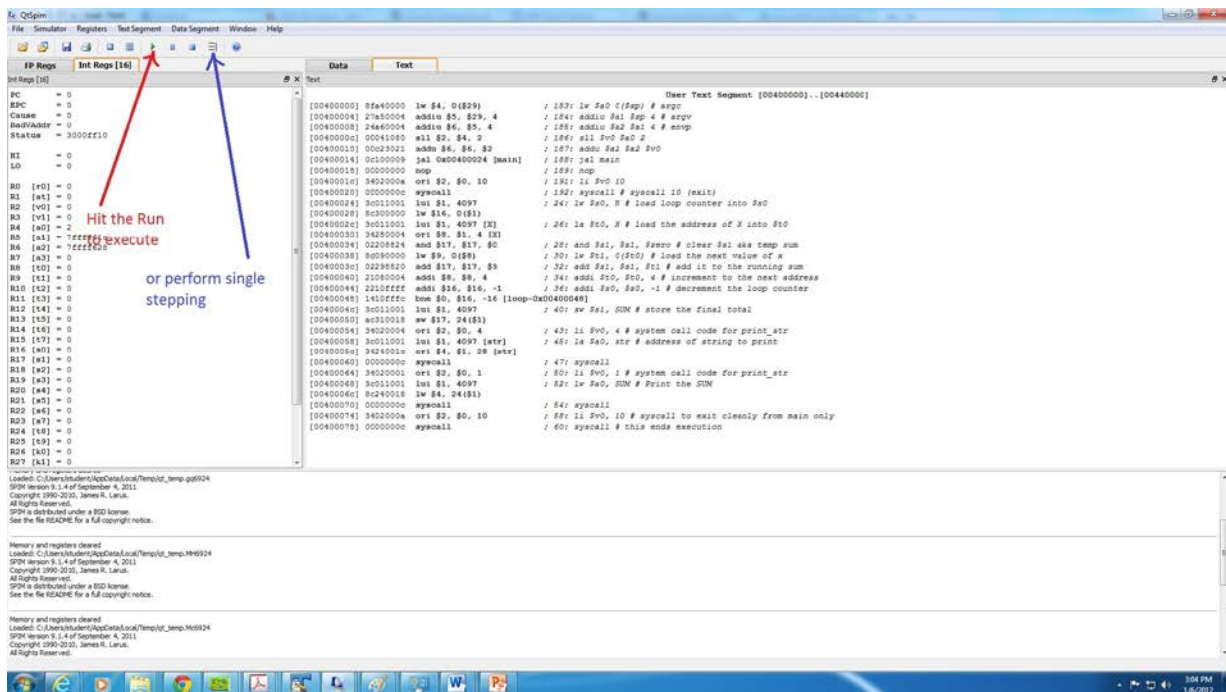


# Steps:

## 1. Load the program



## 2. Execute



### 3. Observe the change in Register contents

Observe the change in contents of \$S1

```
FP Regs  Int Regs [16]  Data  Text
Int Regs [16]
R0 [x0] = 0
R1 [a1] = 10010000
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 2
R5 [a1] = 1ffffe10
R6 [a2] = 1ffffe20
R7 [a3] = 0
R8 [x0] = 10010008
R9 [t1] = 2
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [a0] = 1
R17 [a1] = 1ffffe10
R18 [a2] = 0
R19 [a3] = 0
R20 [a4] = 0
R21 [a5] = 0
R22 [a6] = 0
R23 [a7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [a0] = 0
R27 [a1] = 0

[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $a0 4 # argv
[00400008] 28a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # nargv
[0040000c] 00a10000 wli $2, $4, 2 ; 186: wli $v0 $a2 2
[00400010] 00a23021 addiu $4, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 00100009 jal 0x00400024 (main) ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 00000000 syscall ; 192: syscall # syscall 10 (exit)
[00400024] 30c11001 lui $1, 4097 ; 24: lw $a0, $0 # load loop counter into $a0
[00400028] 8c300000 lw $16, 0($1) ; 26: la $t0, $0 # load the address of X into $t0
[0040002c] 30c11001 lui $1, 4097 [X] ; 28: and $a1, $a1, $zero # clear $a1 aka temp sum
[00400030] 34200004 ori $9, $1, 4 [X] ; 30: lw $t1, 0($t0) # load the next value of X
[00400034] 02208224 and $17, $17, $0 ; 32: add $a1, $a1, $t1 # add it to the running sum
[00400038] 80900000 lw $9, 0($0) ; 34: addi $t0, $t0, 4 # increment to the next address
[0040003c] 02208224 and $17, $17, $0 ; 36: addi $a0, $a0, -1 # decrement the loop counter
[00400040] 21080004 addi $8, $9, 4 ; 40: sv $a1, $t0 # store the final total
[00400044] 2210ffff addi $16, $16, -1 ; 42: li $v0, 4 # system call code for print_str
[00400048] 1410ffff hmv $0, $16, -16 [loop-0x00400048] ; 44: la $a0, $t0 # address of string to print
[0040004c] 30c11001 lui $1, 4097 ; 47: syscall
[00400050] a0310018 sw $17, 24($1) ; 49: li $v0, 1 # system call code for print_str
[00400054] 34020004 ori $2, $0, 4 ; 51: lw $a0, $t0 # Print the SUM
[00400058] 30c11001 lui $1, 4097 [str] ; 54: syscall
[0040005c] 34240010 ori $4, $1, 20 [str] ; 56: li $v0, 10 # syscall to exit cleanly from main only
[00400060] 00000000 syscall ; 60: syscall # this ends execution
[00400064] 34020001 ori $2, $0, 1
[00400068] 30c11001 lui $1, 4097
[0040006c] 8c240018 lw $4, 24($1)
[00400070] 00000000 syscall ; 64: syscall
[00400074] 3402000a ori $2, $0, 10 ; 66: li $v0, 10 # syscall to exit cleanly from main only
[00400078] 00000000 syscall ; 68: syscall # this ends execution
[0040007c] 00000000 syscall
```

2+4+6+8+10 = 30 = 0x1e

```
FP Regs  Int Regs [16]  Data  Text
Int Regs [16]
R0 [x0] = 0
R1 [a1] = 10010000
R2 [v0] = a
R3 [v1] = a
R4 [a0] = 2
R5 [a1] = 1ffffe10
R6 [a2] = 1ffffe20
R7 [a3] = 0
R8 [x0] = 10010018
R9 [t1] = a
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [a0] = 1
R17 [a1] = 1ffffe10
R18 [a2] = 0
R19 [a3] = 0
R20 [a4] = 0
R21 [a5] = 0
R22 [a6] = 0
R23 [a7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [a0] = 0
R27 [a1] = 0

[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $a0 4 # argv
[00400008] 28a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # nargv
[0040000c] 00a10000 wli $2, $4, 2 ; 186: wli $v0 $a2 2
[00400010] 00a23021 addiu $4, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 00100009 jal 0x00400024 (main) ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 00000000 syscall ; 192: syscall # syscall 10 (exit)
[00400024] 30c11001 lui $1, 4097 ; 24: lw $a0, $0 # load loop counter into $a0
[00400028] 8c300000 lw $16, 0($1) ; 26: la $t0, $0 # load the address of X into $t0
[0040002c] 30c11001 lui $1, 4097 [X] ; 28: and $a1, $a1, $zero # clear $a1 aka temp sum
[00400030] 34200004 ori $9, $1, 4 [X] ; 30: lw $t1, 0($t0) # load the next value of X
[00400034] 02208224 and $17, $17, $0 ; 32: add $a1, $a1, $t1 # add it to the running sum
[00400038] 80900000 lw $9, 0($0) ; 34: addi $t0, $t0, 4 # increment to the next address
[0040003c] 02208224 and $17, $17, $0 ; 36: addi $a0, $a0, -1 # decrement the loop counter
[00400040] 21080004 addi $8, $9, 4 ; 40: sv $a1, $t0 # store the final total
[00400044] 2210ffff addi $16, $16, -1 ; 42: li $v0, 4 # system call code for print_str
[00400048] 1410ffff hmv $0, $16, -16 [loop-0x00400048] ; 44: la $a0, $t0 # address of string to print
[0040004c] 30c11001 lui $1, 4097 ; 47: syscall
[00400050] a0310018 sw $17, 24($1) ; 49: li $v0, 1 # system call code for print_str
[00400054] 34020004 ori $2, $0, 4 ; 51: lw $a0, $t0 # Print the SUM
[00400058] 30c11001 lui $1, 4097 [str] ; 54: syscall
[0040005c] 34240010 ori $4, $1, 20 [str] ; 56: li $v0, 10 # syscall to exit cleanly from main only
[00400060] 00000000 syscall ; 60: syscall # this ends execution
[00400064] 34020001 ori $2, $0, 1
[00400068] 30c11001 lui $1, 4097
[0040006c] 8c240018 lw $4, 24($1)
[00400070] 00000000 syscall ; 64: syscall
[00400074] 3402000a ori $2, $0, 10 ; 66: li $v0, 10 # syscall to exit cleanly from main only
[00400078] 00000000 syscall ; 68: syscall # this ends execution
[0040007c] 00000000 syscall
```