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.

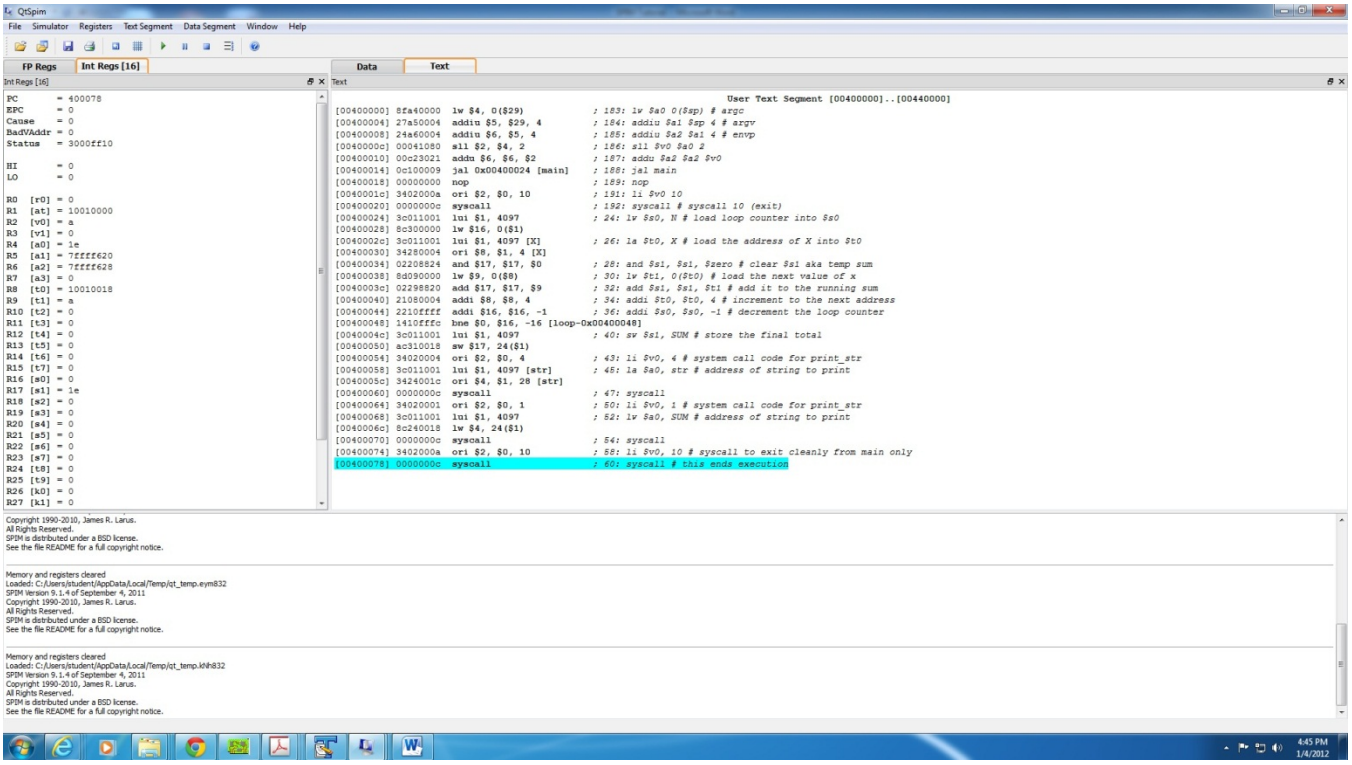


Figure 1 : QtSpim

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 Riinitialize simulator tab in the toolsbar 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.

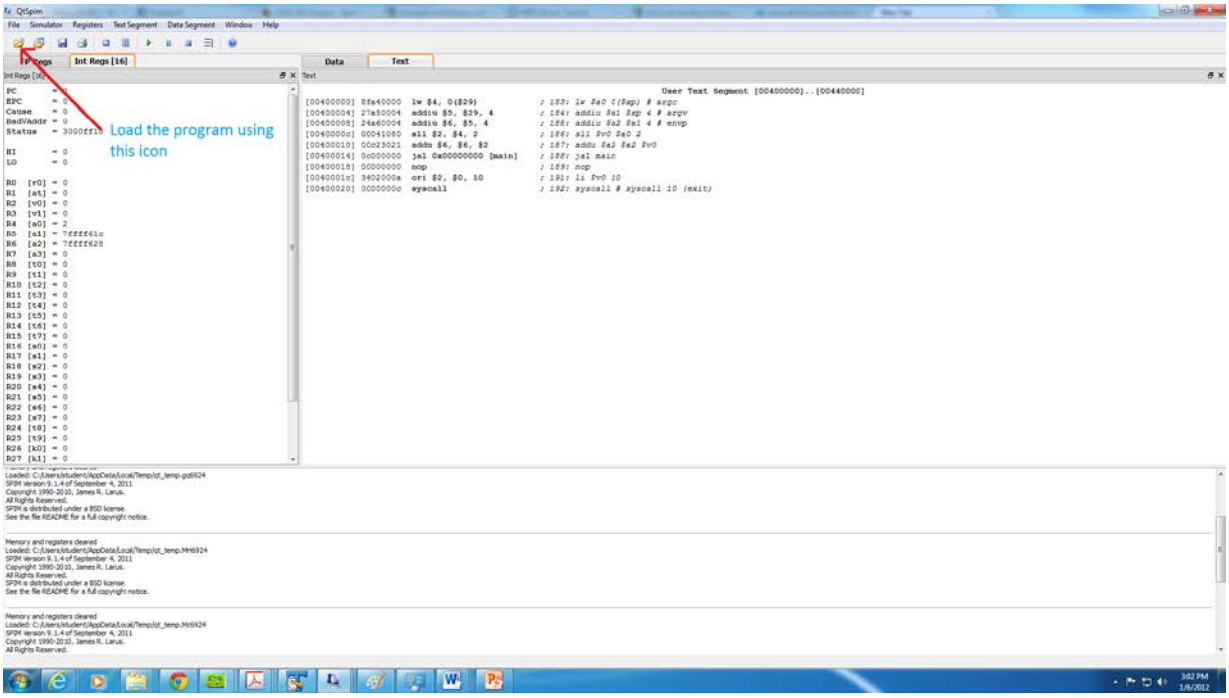
Example Program

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.,

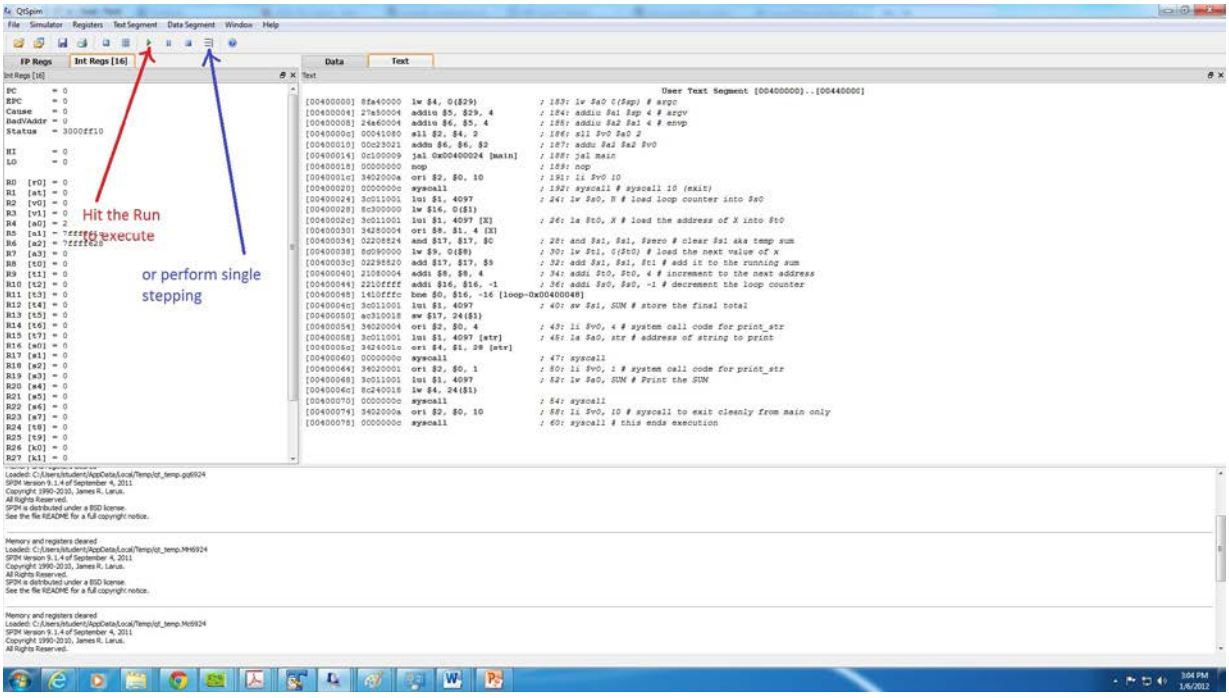
```
# first SPIM program

# ECE 484/584
#

.data                                   # Put Global Data here

N: .word 5                              # loop count

X: .word 2,4,6,8,10                     # array of numbers to be added'

SUM: .word 0                            # location of the final sum


str:

.asciiz "The sum of the array is = "


.text                                   # Put program here

.globl main                             # globally define 'main'

main: lw $s0, N                         # load loop counter into $s0

la $t0, X                               # load the address of X into $t0

and $s1, $s1, $zero                     # clear $s1 aka temp sum

loop: lw $t1, 0($t0)                    # load the next value of x

add $s1, $s1, $t1                       # add it to the running sum

addi $t0, $t0, 4                        # increment to the next address

addi $s0, $s0, -1                       # decrement the loop counter

bne $0, $s0, loop                       # loop back until complete

sw $s1, SUM                             # store the final total

li $v0, 10                              # syscall to exit cleanly from main only, refer to Figure
                                        # A.9.1 in the Assemblers, Linkers, and the SPIM
                                        #Simulator document (PDF).


syscall                                 # this ends execution

.end
```

Steps:

1. Load the program



2. Execute



3. Observe the change in Register contents