Programming with QtSpim: A User's Manual

John McCranie

November 15, 2013

1 Introduction to SPIM

Spim is a self-contained simulator that runs MIPS32 programs. It utilizes the instruction set originally created for the MIPS R2000/3000 series of microprocessors. The instruction set is representative of Instruction Set Architectures (ISAs) of processors designed since 1984 and of the core of modern microprocessors including the X86.

The current version of Spim is called QtSpim. Unlike previous versions, QtSpim is available for Windows, Mac, and Linux, utilizing the same source code and the same user interface on all three platforms. Older versions of Spim are available, but QtSpim is the only version currently being updated and maintained. Thus, it is highly recommended that you use QtSpim.

The following is a basic user's manual for utilizing QtSpim, and is intended specifically for Georgia Tech students entering ECE 3056. Please note that a basic understanding of the MIPS assembly language is assumed, and that this manual is not intended to teach programming in MIPS.

2 Getting Started

The first thing that will need to be done is to download QtSpim onto your computer. QtSpim can be easily and quickly downloaded using the following link:

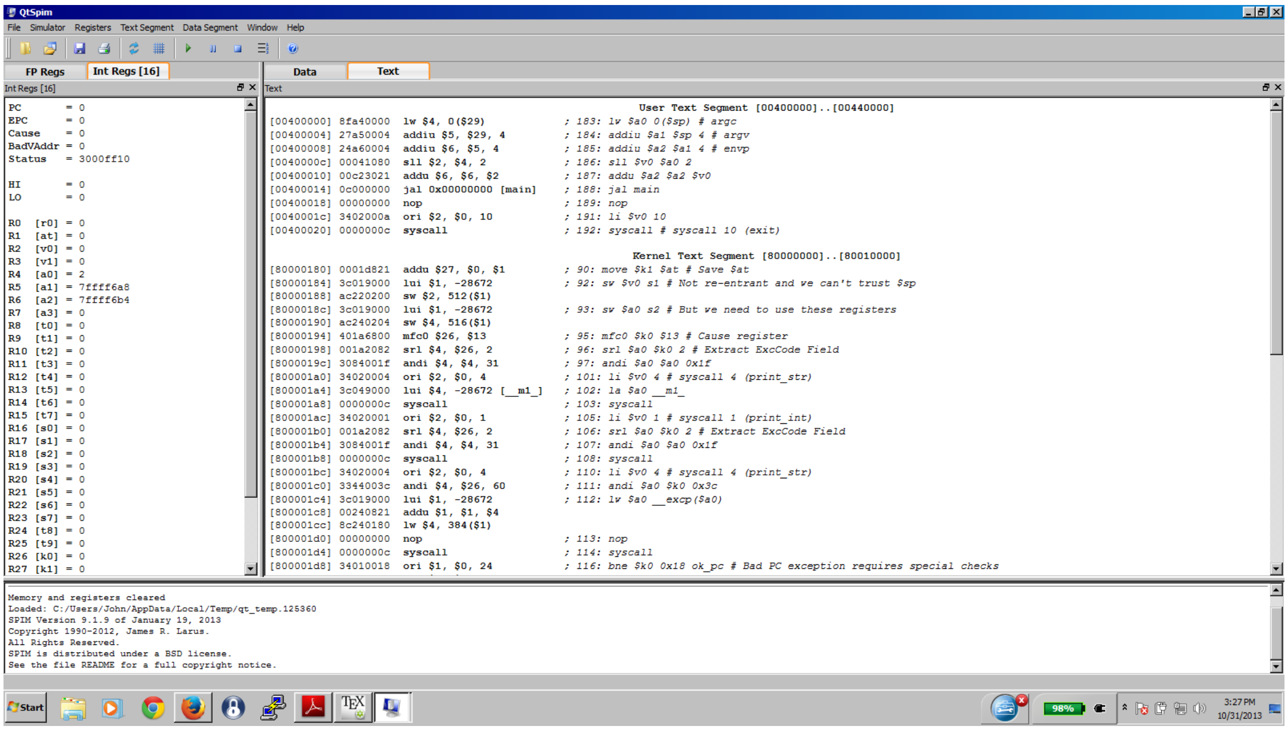
http://sourceforge.net/projects/spimsimulator/files/

Multiple versions currently exist, but downloading the most recent version, QtSpim_9.1.9, is recommended. Make sure to choose the version formatted for download on your specific platform, i.e. Windows, Mac, etc.

Once downloading is complete, simply use the installer provided in the download. Follow the instructions of the install-wizard to complete installation. This should be a fairly quick and easy process. Once installed, simply click on QtSpim.exe to begin using the simulator.

3 The Interface

When QtSpim starts, two windows should open. The first window will be the user interface seen in Figure 1 below. The other window opened is called the console, the usage of which will be explained later. In the user interface, as seen in Figure 1, you will see several different panes which are used for debugging. The largest pane is the data/text segment pane. The two tabs located at the top of the pane are used to switch between viewing the data or the text segment. The text segment displays the currently loaded program, and can be used to monitor a program as it executes. The data segment, of course, displays whatever is inserted into the data segment of your program. The pane on the left is the register pane; it lists all 32 standard registers, as well as the HI/LO registers, and displays what is contained within the registers at any given time. The final pane at the bottom is the message pane. This pane will list any errors detected by QtSpim while attempting to run your program.



```
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 2
R5  [a1] = 7ffff6a8
R6  [a2] = 7ffff6b4
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

```
                                                          User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)             ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4          ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4           ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2             ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2           ; 187: addu $a2 $a2 $v0
[00400014] 0c000000  jal 0x00000000 [main]     ; 188: jal main
[00400018] 00000000  nop                       ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10            ; 191: li $v0 10
[00400020] 0000000c  syscall                   ; 192: syscall # syscall 10 (exit)

                                                         Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1          ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672            ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672            ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13             ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2            ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31           ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4             ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]    ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                   ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1             ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2            ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31           ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c  syscall                   ; 108: syscall
[800001bc] 34020004  ori $2, $0, 4             ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c  andi $4, $26, 60          ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000  lui $1, -28672            ; 112: lw $a0 __excp($a0)
[800001c8] 00240821  addu $1, $1, $4
[800001cc] 8c240180  lw $4, 384($1)
[800001d0] 00000000  nop                       ; 113: nop
[800001d4] 0000000c  syscall                   ; 114: syscall
[800001d8] 34010018  ori $1, $0, 24            ; 116: bne $k0 0x18 ok_pc # Bad PC exception requires special checks
```

```
Memory and registers cleared
Loaded: C:/Users/John/AppData/Local/Temp/qt_temp.125360
SPIM Version 9.1.9 of January 19, 2013
Copyright 1990-2012, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
```

Figure 1: QtSpim User Interface

4 Using QtSpim

QtSpim is a simulator that runs MIPS programs loaded from your computer. Programs can be written using any text editor; however, it is recommended that you use Notepad or some other basic editor, as the more complicated editors like Microsoft Word use specialized characters that can cause problems in QtSpim. It is also possible to use editors made specifically for MIPS such as Mipster (link in section 5).

To load a program into QtSpim, simply click File and then Reinitialize and Load File. It will be necessary to reinitialize and load every time you wish to run the program, i.e. after running the program once, you must reinitialize and load to run it again. To simulate your program use the options under Simulator. These options include Run, Pause, Stop, and, most crucially, Single Step, which is used to step through your program one instruction at a time. Use these options to run and debug your programs.

4.1 Register Naming Convention

QtSpim is capable of reading registers in two different formats. The typical numbered format ($1, $2, $3, etc.) and the named register format ($t0, $s0, $v0) which assigns a letter and a number to each register based on its intended function. For example, the "t" registers, $t0 - $t9, are "temporary" registers used for temporarily holding certain values while the "s" registers, $s0 - $s7, are "saved" registers which hold their value over function calls. In addition, certain special registers serve a specific purpose, such as the stack pointer, $sp, and the return address, $ra. While either convention can be used, and any register can be of the type the program writer specifies, it is convenient to use the named format to distinguish between the various types of registers. Table 1 below lists the various names of registers, their types, and the numbered registers to which they correspond.

Name	**Number**	**Type**
$zero	$0	Constant value 0
$at	$1	Assembler Temporary
$v0 - $v1	$2 - $3	Value
$a0 - $a3	$4 - $7	Arguments
$t0 - $t7	$8 - $15	Temporaries
$s0 - $s7	$16 - $23	Saved
$t8 - $t9	$24 - $25	Temporaries
$k0 - $k1	$26 - $27	Reserved for OS Kernel
$gp	$28	Global Pointer
$sp	$29	Stack Pointer
$fp	$30	Frame Pointer
$ra	$31	Return Address

4.2 Pseudo-Instructions

QtSpim can also interpret what are called pseudo-instructions. These are instructions that are written by the program writer to perform a specific purpose not necessarily supported by the MIPS ISA. These instructions are translated by QtSpim into native instructions. For example ”li”, or ”load immediate” loads a value directly into a given register. QtSpim simply translates this instruction into an add immediate (addi) instruction that adds the given immediate to the zero register i.e. li $1, 4 = addi $1, $0, 4. Certain other pseudo-instructions will be translated into two native instructions; for example ”bgt”, or ”Branch if Greater Than” which combines the functionality of a ”Set if Less Than” and a ”branch” instruction. Keep in mind that when running a program with pseudo-instructions the text segment pane will display the native instruction version of the pseudo-instructions and pseudo-instructions such as bgt will still count as two instructions. Pseudo-instructions can not be used to make your program more efficient; they are simply for the convenience of the programmer. Table 2 below contains a brief list of pseudo-instructions, their meanings, and the native instructions they translate to.

Instruction	Meaning	Translation
move $rt,$rs	Move	add $rt,$rs,$zero
clear $rt	Clear	add $rt,$zero,$zero
la $rd, LabelAddr	Load Address	lui $rd, LabelAddr[31:16]; ori $rd,$rd, LabelAddr[15:0]
li $rd, IMMED[31:0]	Load Immediate	lui $rd, IMMED[31:16]; ori $rd,$rd, IMMED[15:0]
bgt $rs,$rt,Label	Branch if Greater Than	slt $at,$rt,$rs; bne $at,$zero,Label
blt $rs,$rt,Label	Branch if Less Than	slt $at,$rs,$rt; bne $at,$zero,Label
bge $rs,$rt,Label	Branch if Greater than or Equal	slt $at,$rs,$rt; beq $at,$zero,Label
ble $rs,$rt,Label	Branch if Less than or Equal	slt $at,$rt,$rs; beq $at,$zero,Label

4.3 Syscalls

Syscalls are very important in QtSpim. Syscalls are used to perform I/O operations as well as end every program. When a syscall is made, QtSpim will check the value of register $v0 and perform a specific function based on the given value. For example, the value 1 will print a string from register $a0, and the value 5 will ask for input from the user. All input and output will be written to the console, the other window that opened when QtSpim was opened. When the value in $v0 is 10 and a syscall is made, the program will end. Every program run in QtSpim should finish in this way. Table 3 below lists some of the services provided by syscalls, the integer in $v0 required to call the specific service, the arguments required, and the results of the call.

Service	Integer	arguments	Result
Print Integer	1	$a0 = integer	
Print Float	2	$f12 = float	
Print Double	3	$f12 = double	
Print String	4	$a0 = string	
Read Integer	5		integer (in $v0)
Read Float	6		float (in $f0)
Read Double	7		double (in $f0)
Read String	8	$a0 = buffer, $a1 = length	
Exit Program	10		

5 Additional Documentation

Complete List of Registers and Syscall functions: https://www.cs.tcd.ie/John.Waldron /itral/spim_ref.html

Spim Homepage: http://spimsimulator.sourceforge.net/

Mipster Text Editor: http://www.downcastsystems.com/mipster/