Eastern Mediterranean University

Computer Engineering Department

CMSE 222 Introduction to Computer Organization– Lab. 7

1. Overview of QtSpim (review)

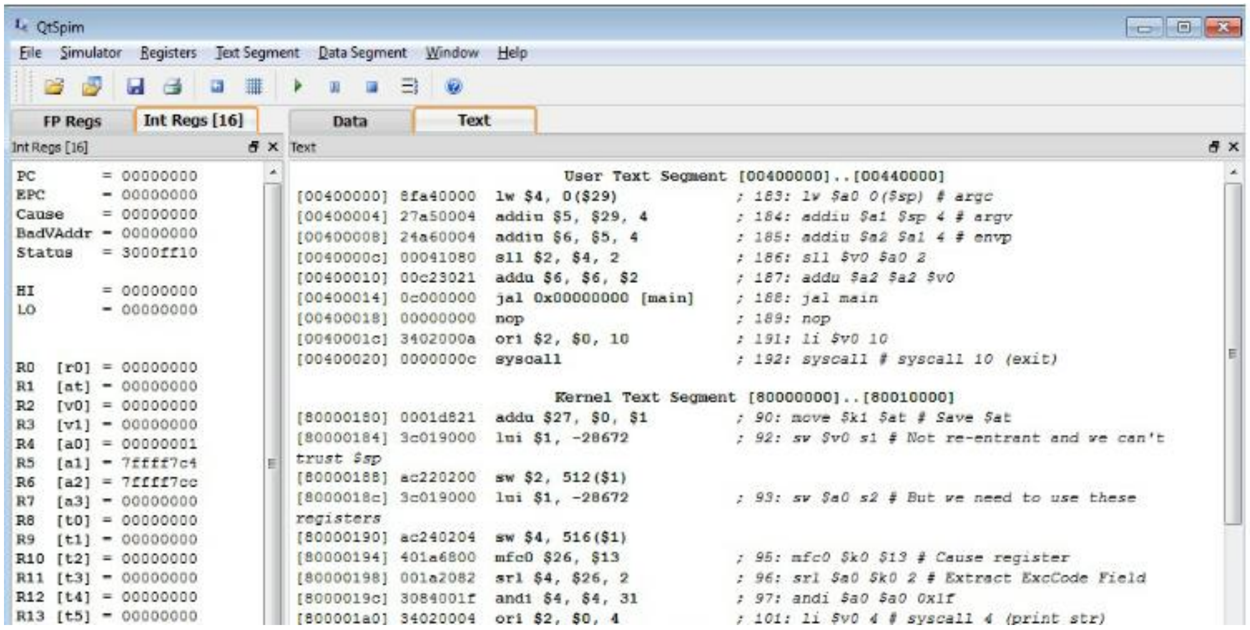
QtSpim is a self-contained simulator that will run a MIPS32 assembly program and display the processor's registers and memory. QtSpim reads and executes programs written in assembly language for a MIPS computer. QtSpim does not execute binary (compiled) programs. To simplify programming, QtSpim provides a simple debugger and small set of operating system services.

QtSpim implements most of the MIPS32 assembler-extended instruction set. (It omits the floating point comparisons and rounding modes and the memory system page tables.) The MIPS architecture has several variants that differ in various ways (e.g., the MIPS64 architecture supports 64-bit integers and addresses), which means that QtSpim will not run programs for all MIPS processors.

In this lab we are going to present a brief overview of QtSpim and implement our program here.

2. Getting Started with QtSpim (quick review)

When QtSpim starts up, it opens a window containing that looks like the one below. (The features in the window look slightly different on Microsoft Windows than on Linux or Mac OSX, but all the menus and buttons are in the same place and work the same way).


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text

```
Int Regs [16]
PC        = 00000000
EPC       = 00000000
Cause     = 00000000
BadVAddr  = 00000000
Status    = 3000ff10

HI        = 00000000
LO        = 00000000


R0  [r0] = 00000000
R1  [at] = 00000000
R2  [v0] = 00000000
R3  [v1] = 00000000
R4  [a0] = 00000001
R5  [a1] = 7ffff7c4
R6  [a2] = 7ffff7cc
R7  [a3] = 00000000
R8  [t0] = 00000000
R9  [t1] = 00000000
R10 [t2] = 00000000
R11 [t3] = 00000000
R12 [t4] = 00000000
R13 [t5] = 00000000
```

```
Text
                         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)
```

QtSpim's main window has three parts:

- The narrow pane on the left can display integer or floating-point registers. Select the set of registers by clicking the tab at the top of the pane.
- The wide pane on the right can display the text segment, which contains instructions, and the data segments. Choose between text and data by clicking the tab at the top of the pane.
- The small pane on the bottom is where QtSpim writes its messages.

All of the panes are dockable, which means that you can grab a pane by its top bar and drag it out of QtSpim's main window, to put on some other part of your screen. QtSpim also opens another window called Console that displays output from your program.

Loading a Program

Your program should be stored in a file. Assembly code files usually have the extension ".s", as in file1.s. To load a file, go to the File menu and select Load File. The screen will change as the file is loaded, to show the instructions and data in your program.

Another very useful command on the File men is Reinitialize and Load File. It first clears all changes made by a program, including deleting all of its instructions, and then reloads the last file. This command works well when debugging a program, as you can change your program and quickly test it in a fresh computer without closing and restarting QtSpim.

<u>Running a Program</u>

To start a program running after you have loaded it, go to the Simulator menu and click Run/Continue. Your program will run until it finishes or until an error occurs. Either way, you will see the changes that your program made to the MIPS registers and memory, and the output your program writes will appear in the Console window.

If your program does not work correctly, there are several things you can do. The easiest is to single step between instructions, which lets you see the changes each instructions makes, one at a time. This command is also on the Simulator menu and is named Single Step.

Sometimes, however, you need to run your program for a while before something goes wrong, and single stepping would be too slow. QtSpim lets you set a breakpoint at a specific instruction, which stops QtSpim before the instruction executes. So, if you think your problem is in a specific function in your program, set a breakpoint at the first instruction in the function, and QtSpim will stop everytime the function is invoked. You set a breakpoint by right-clicking on the instruction where you want to stop, and selecting Set Breakpoint. When you are done with the breakpoint, you can remove it by selecting Clear Breakpoint instead.

If you want to stop your program while it is running, go to the Simulator menu and click Pause. This command stops your program, let you look around, and continue execution if you want. If you do not want to continue running, click Stop instead.

When QtSpim stops, either because of an error in your program, a breakpoint, after clicking Pause, or after single stepping, you can continue the program running by clicking on Run/Continue (or you can continue single stepping by clicking Single Step). If you click Stop, instead of Pause, then clicking Run/Continue will restart your program from the beginning, instead of continuing from where it stopped. (This is roughly the same way that a music player operates; you can pause and restart a song, but if you stop the music, you need to start playing at the beginning.)

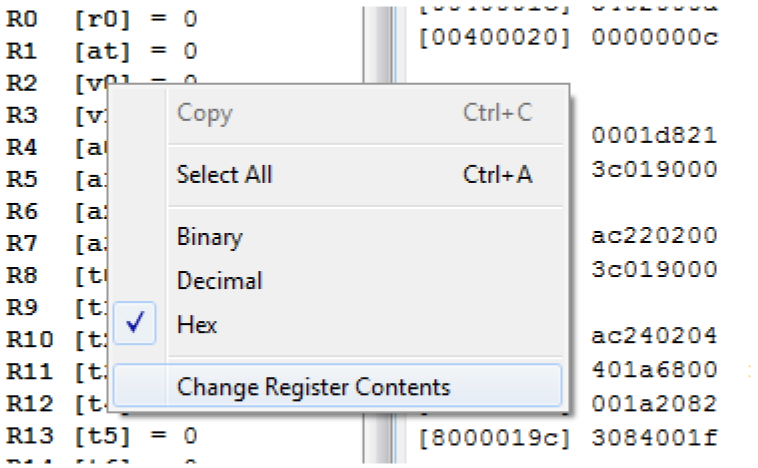
Display Options

The three other menus -- Registers, Text Segment, and Data Segment -- control QtSpim's displays. For example, the Register menu controls the way QtSpim displays the contents of registers, either in binary, base 8 (octal), base 10 (decimal), or base 16 (hexadecimal). It is often quite convenient to flip between these representations to understand your data.

These menus also let you turn off the display of various parts of the machine, which can help reduce clutter on the screen and let you concentrate on the parts of the program or data that really matter.

Changing Registers and Memory

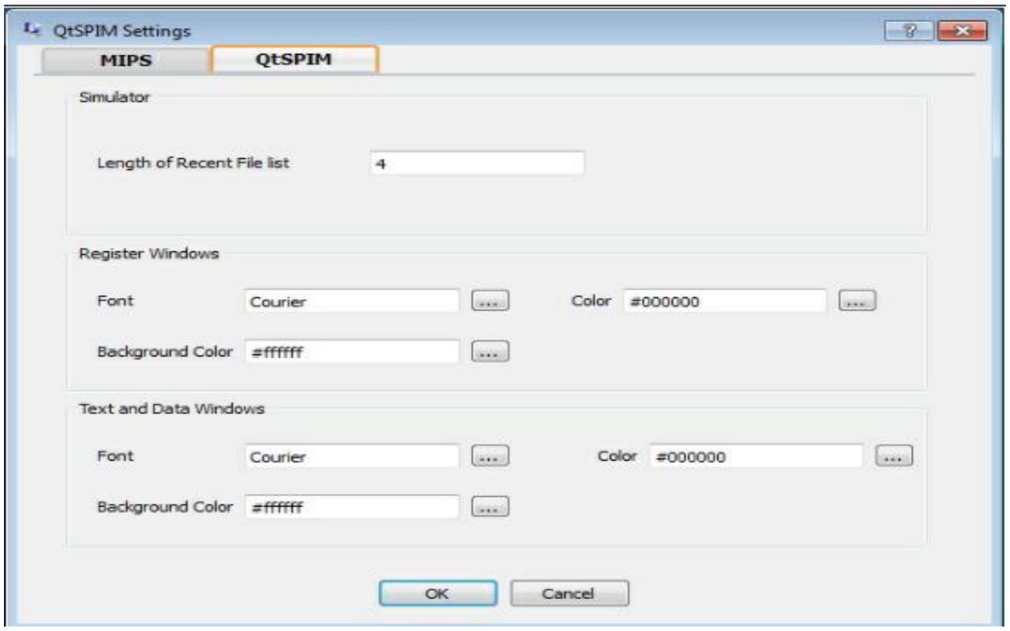
You can change the contents of either a register or memory location by right-clicking on it and selecting Change Register Contents or Change Memory Contents, respectively.



```
R0   [r0] = 0                 [00400020] 0000000c
R1   [at] = 0
R2   [v0] = 0
R3   [v                                   0001d821
R4   [a                                   3c019000
R5   [a
R6   [a                                   ac220200
R7   [a                                   3c019000
R8   [t
R9   [t                                   ac240204
R10  [t                                   401a6800
R11  [t                                   001a2082
R12  [t
R13  [t5] = 0                 [8000019c] 3084001f
```

Settings

The Simulator menu contains the Settings command, which brings up a dialog like this:



Simulator

Length of Recent File list: 4

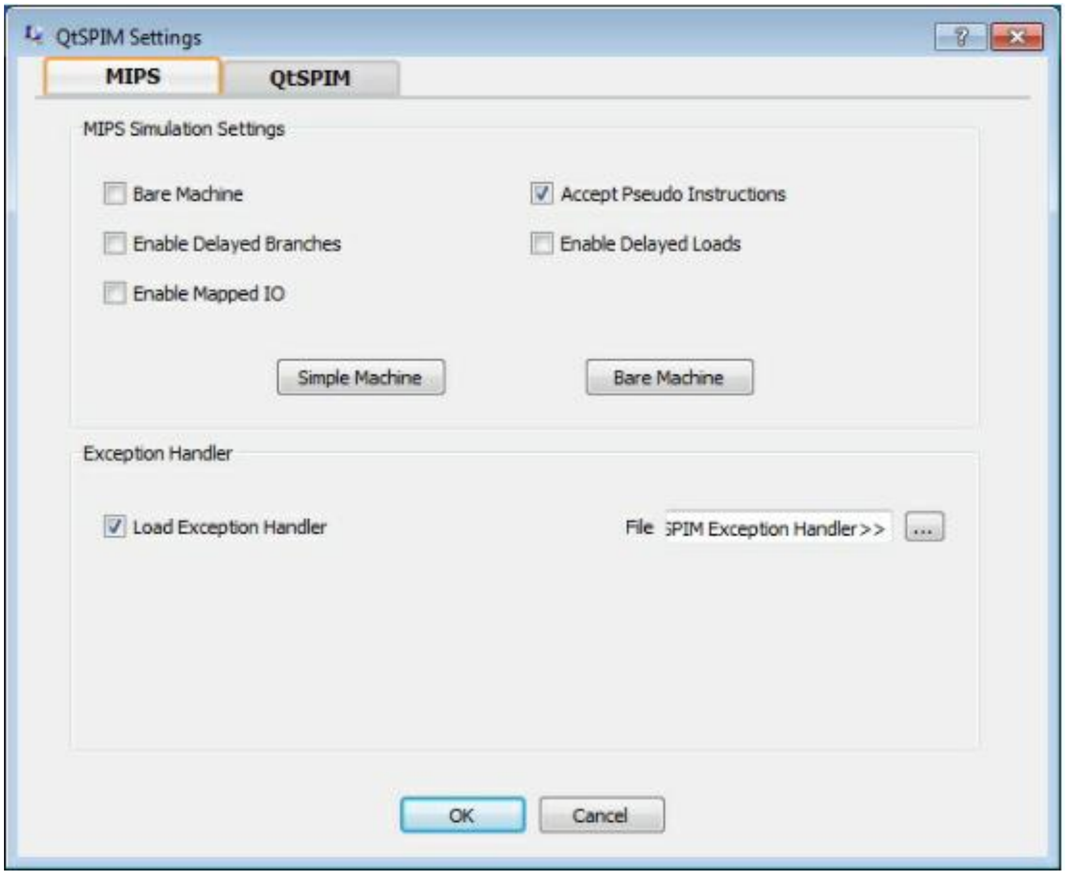
Register Windows

Font: Courier
Color: #000000
Background Color: #ffffff

Text and Data Windows

Font: Courier
Color: #000000
Background Color: #ffffff

The dialog has two tabs. The first, shown above, changes the visual aspects of QtSpim, such as the fonts. The second looks like this:



MIPS Simulation Settings

Bare Machine
[✓ Accept Pseudo Instructions]
Enable Delayed Branches
Enable Delayed Loads
Enable Mapped IO

Exception Handler

[✓ Load Exception Handler]
File: SPIM Exception Handler>>

It changes the way that QtSpim operates:

- ✓ Bare machine make QtSpim simulate a bare MIPS processor.
- ✓ Accept pseudo instructions enables QtSpim to accept assembly instructions that MIPS does not actually execute, to make programming easier.
- ✓ Enable delayed branches causes QtSpim to execute the instruction immediately after a branch instruction before transferring control and to calculate the new PC from the address of this next instruction.
- ✓ Enable delayed loads causes QtSpim to delay the value loaded from memory for one instruction after the load instructions.
- ✓ Enable mapped IO turns on memory-mapped IO.

The button marked Simple Machine enables the most common options (Accept Pseudo Instructions) that are what most people use. The button marked Bare Machine turns on the instructions corresponding to a real MIPS processor (Bare Machine, Delayed Branches, and Delayed Loads).

Note: Now let's begin our first program for LAB2

3. Experimental Work

Part 1:
In this part, you will use the SPIM in pseudo-code allowing mode.

- ✓ Clear the bare machine setting, and check only the allow-pseudocode option in the settings dialog-box (key-sequence alt-S,L).
- ✓ Write the following text to a file named "exp2.asm"

```
.data
.text
.globl main
main:
li $8,0x3210
li $9,0x76543210
sge $11,$8,$9
mul $12,$11,$10
infloop:
bge $11,$0,infloop
syscall
```

- ✓ Load the file to SPIM, and watch the corresponding machine codes of each line. Use the log file to fill in the following binary-machine-code table to understand the fields of each instruction in a better manner.

Note that the given text is not a program, it is not traceable. It contains a sample of some commonly used MIPS pseudo-instructions.

Part 2: Trace the MIPS example

This a simple MIPS code, trace the code and think about its aim. You can find many MIPS instruction here. Pay attention to .word in this code. How many times we use it and why?

```
# A demonstration of some simple MIPS instructions
# used to test QtSPIM
        # Declare main as a global function
        .globl main
        # All program code is placed after the
        # .text assembler directive
        .text
# The label 'main' represents the starting point
main:
        li $t2, 25                  # Load immediate value (25)
        lw $t3, value               # Load the word stored in value (see bottom)
        add $t4, $t2, $t3           # Add
        sub $t5, $t2, $t3           # Subtract
        sw $t5, Z                   #Store the answer in Z (declared at the bottom)
        # Exit the program by means of a syscall.
        # There are many syscalls - pick the desired one
        # by placing its code in $v0. The code for exit is "10"
        li $v0, 10 # Sets $v0 to "10" to select exit syscall
        syscall # Exit
        # All memory structures are placed after the
        # .data assembler directive
        .data
        # The .word assembler directive reserves space
        # in memory for a single 4-byte word (or multiple 4-byte words)
        # and assigns that memory location an initial value
        # (or a comma separated list of initial values)
value:  .word 12
Z:      .word 0
```

9. Reporting

Before the Lab-time is over, fill in the following report page as soon as you complete the laboratory work, and submit it to your assistant. Your report is important for your grading.

Name: ____________________ Student Number:______________

Submitted to (Asst.): ____________________ Date:dd/mm/yy ___/___/___

EASTERN MEDITERRANEAN UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

2019 Fall

CMPE 324 -Computer Architecture and Organization

EXPERIMENT 1 - Reporting Sheet

Part 1: The observed binary machine codes of the instructions are:

Instruction	opc	rs	rt	rd	sa	fn
`li $8,0x3210`						
`li $9,0x76543210`						
`sge $11,$8,$9`						
`mul $12,$11,$10`						
`bge $11,$0,infloop`						

Part 2: The observed binary machine codes of the instructions are:

Instruction	**opc**	**rs**	**rt**	**rd**	**sa**	**fn**
li $t2, 25						
lw $t3, value						
add $t4, $t2, $t3						
sub $t5, $t2, $t3						
sw $t5, Z						
li $v0, 10						

Grading:

Lab Performance:

Asst. Observations: