# EASTERN MEDITERRANEAN UNIVERSITY
# DEPARTMENT OF COMPUTER ENGINEERING
# CMSE222

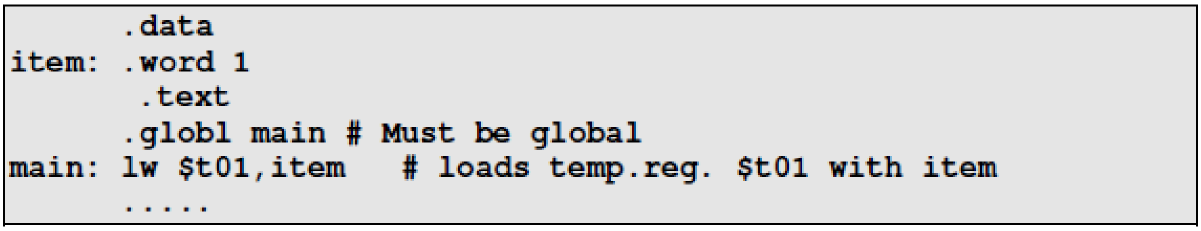
# LAB # 6
# INTRODUCTION TO MIPS

Aim of the Lab Experiment: In this lab experiment, we will examine the MIPS programs, implement programs using QtSpim and develop assembly programs with MIPS.

## 1. MIPS Assembler Syntax

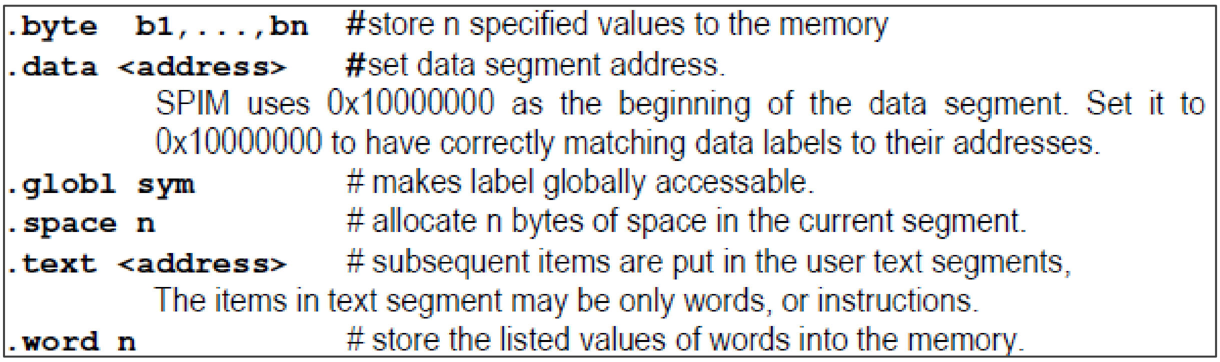
**Comments** in assembler files begin with a sharp sign " # ". Everything from the sharp sign to the end of the line is ignored.

**Identifiers** are a sequence of alphanumeric characters, underbars "_", and dots " . " that do not begin with a number Instruction opcodes are reserved words that cannot be used as identifiers. Labels are declared by putting them at the beginning of a line followed by a colon, for example:



```
      .data
item: .word 1
       .text
      .globl main # Must be global
main: lw $t01,item   # loads temp.reg. $t01 with item
      .....
```

Numbers are base 10 by default. If they are preceded by 0x, they are interpreted as hexadecimal. Hence, 256 and 0x100 denote the same value. Strings are enclosed in double quote "...". Special characters in strings follow the C convention: i.e., newline is \n; tab \t, and quote \" Some important SPIM (and also MIPS) assembler directives:



```
.byte  b1,...,bn   #store n specified values to the memory
.data <address>    #set data segment address.
        SPIM uses 0x10000000 as the beginning of the data segment. Set it to
        0x10000000 to have correctly matching data labels to their addresses.
.globl sym         # makes label globally accessable.
.space n           # allocate n bytes of space in the current segment.
.text <address>    # subsequent items are put in the user text segments,
        The items in text segment may be only words, or instructions.
.word n            # store the listed values of words into the memory.
```

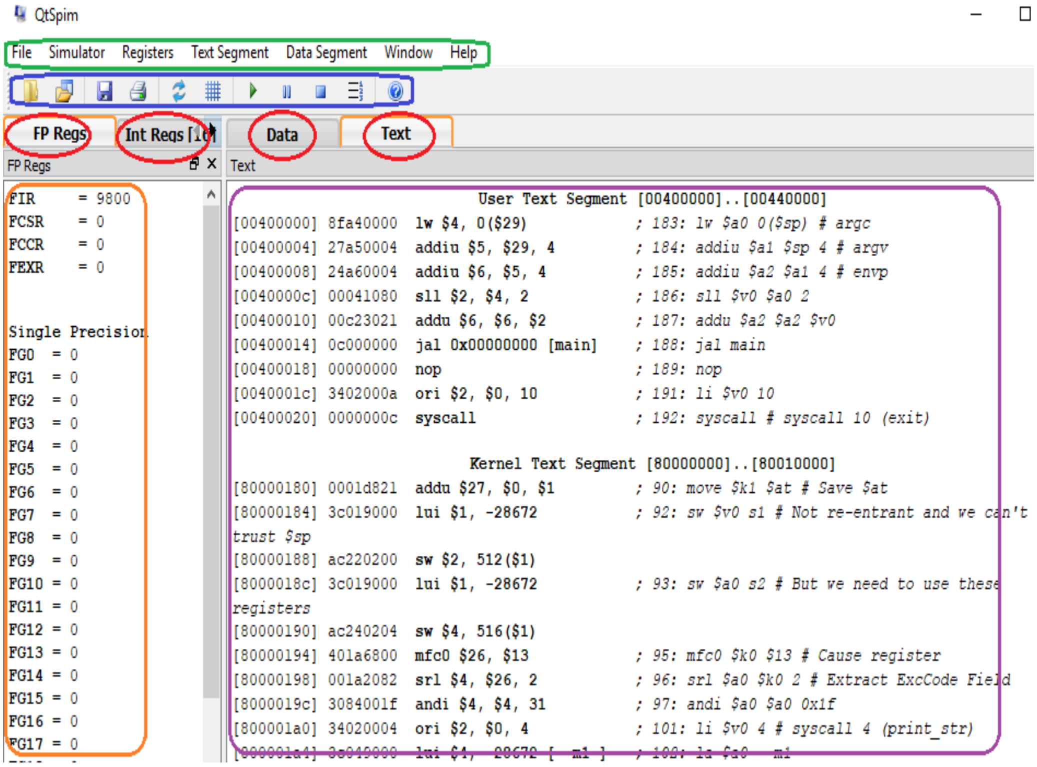
## 2. Introduction to QtSpim

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

Note: you can find this IDE in https://sourceforge.net/projects/spimsimulator/files/ or on google as well.

## 3. Getting Started with QtSpim

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).

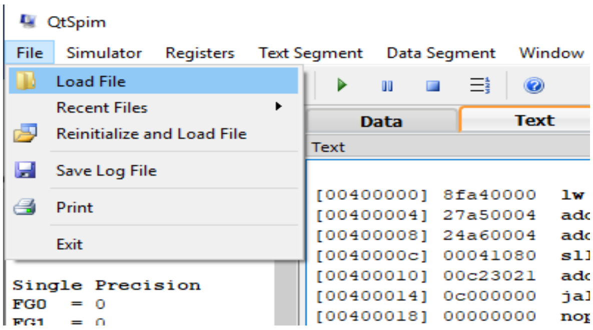


```
FIR     = 9800
FCSR    = 0
FCCR    = 0
FEXR    = 0

Single Precision
FG0  = 0
FG1  = 0
FG2  = 0
FG3  = 0
FG4  = 0
FG5  = 0
FG6  = 0
FG7  = 0
FG8  = 0
FG9  = 0
FG10 = 0
FG11 = 0
FG12 = 0
FG13 = 0
FG14 = 0
FG15 = 0
FG16 = 0
FG17 = 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_
```

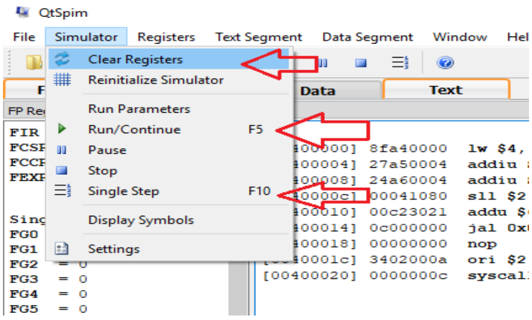
QtSpim's main window has three parts:

- 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.

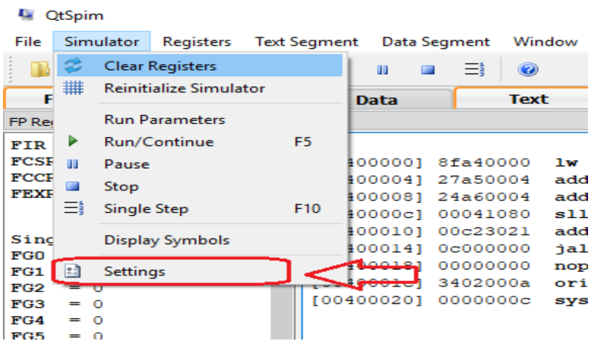


```
[00400000] 8fa40000 lw
[00400004] 27a50004 add
[00400008] 24a60004 add
[0040000c] 00041080 sll
[00400010] 00c23021 add
[00400014] 0c000000 jal
[00400018] 00000000 nop
```

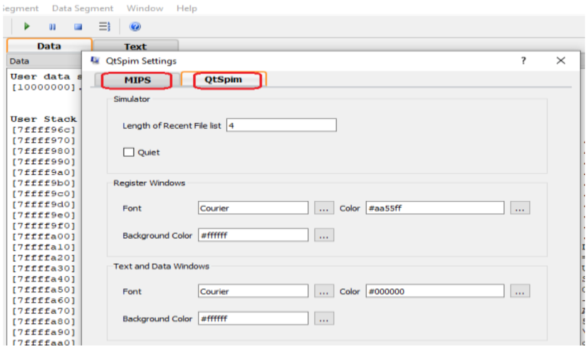
File Menu



Simulator Menu



Settings in Menu



Simulator

Length of Recent File list 4

□ Quiet

Register Windows

Font Courier ... Color #aa55ff ...

Background Color #ffffff ...

Text and Data Windows

Font Courier ... Color #000000 ...

Background Color #ffffff ...

Settings Window

**MIPS Settings**

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.

### 3.1. 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.

### 3.2. Running a Program

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.

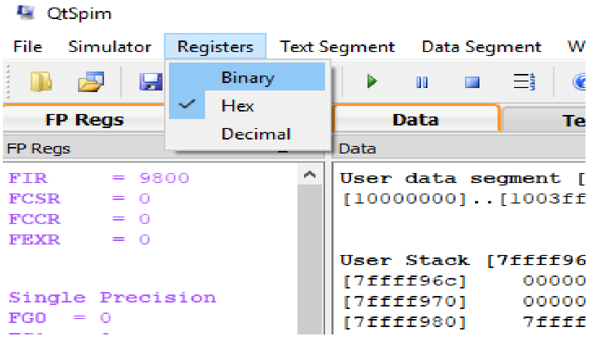
Sometimes, however, you need to run your program for a while before something goes wrong, and single stepping would be too slow. QtSpim lets you set a breakpoint at a specific instruction, which stops QtSpim before the instruction executes. So, if you think your problem is in a specific function in your program, set a breakpoint at the first instruction in the function, and QtSpim will stop every time 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.)

### 3.3. Display Options

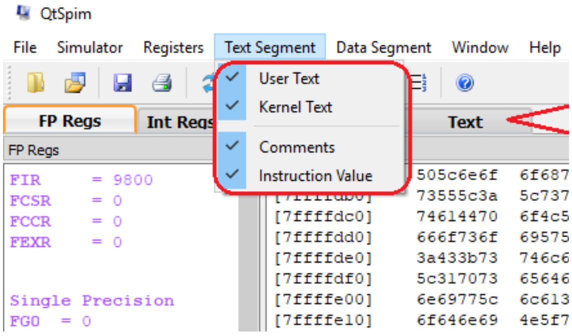
The three other menus -- Registers, Text Segment, and Data Segment -- control QtSpim's displays. For example, the Register menu controls the way QtSpim displays the contents of registers, either in binary, base 8 (octal), base 10 (decimal), or base 16 (hexadecimal). It is often quite convenient to flip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



```
FP Regs
FIR      = 9800
FCSR     = 0
FCCR     = 0
FEXR     = 0


Single Precision
FG0   = 0
```

```
Data
User data segment [
[10000000]..[1003ff


User Stack [7ffff96
[7ffff96c]      00000
[7ffff970]      00000
[7ffff980]      7ffff
```

Register Menu



```
FP Regs
FIR      = 9800
FCSR     = 0
FCCR     = 0
FEXR     = 0


Single Precision
FG0   = 0
```

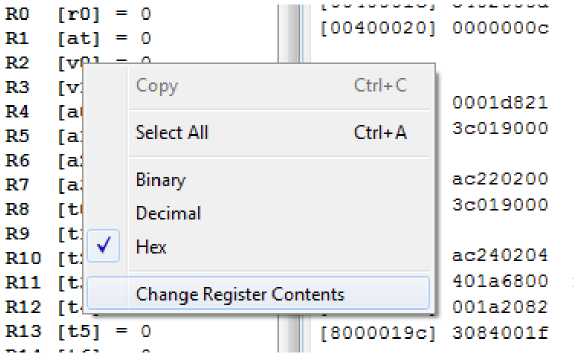
```
                 505c6e6f   6f687
                 73555c3a   5c737
[7ffffdc0]       74614470   6f4c5
[7ffffdd0]       666f736f   69575
[7ffffde0]       3a433b73   746c6
[7ffffdf0]       5c317073   65646
[7ffffe00]       6e69775c   6c613
[7ffffe10]       6f646e69   4e5f7
```

Text Segment Menu

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.

### 3.4. 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
R1   [at] = 0
R2   [v0] = 0
R3   [v
R4   [a
R5   [a
R6   [a
R7   [a
R8   [t
R9   [t
R10  [t
R11  [t
R12  [t
R13  [t5] = 0
```

```
[00400020] 0000000c


           0001d821
           3c019000

           ac220200
           3c019000

           ac240204
           401a6800
           001a2082
[8000019c] 3084001f
```

## 4. EXPERIMENTAL WORK

**Part 1**: Run the below code and observe the results

Following program multiplies two unsigned integers in the registers R8 by R9 and writes the 32-bit product to register R10. In order to understand the operation of your simulator program, type and execute the following MIPS assembly program in non-pseudo-instruction mode.

```
.data 0x10000000
.text 0x00400000
main:
    addi $8,$0,6
    addi $9,$0,12
    add $2,$0,$8
    add $10,$0,$0
# multiplication of $8 * $9 -> $10
mulloop:
    beq $2,$0,mulexit # if zero exit
    addi $2,$2,-1
    add $10,$10,$9
    j mulloop
mulexit:
    # multiplication loop is over,
    # is the result in $10 correct?
    sll $0,$0,0
    syscall
```

- First set the PC (prog.counter) to the starting address of the program if SPIM is set correctly the starting address is 0x00400000. To set the value use the key-sequence alt-s,v (or menu simulator>set value) to open the register-value assignment dialog box. Enter PC and the starting address in hexadecimal format.
- Next, use the **Fn10** key to execute one instruction at each key-press. You can also use the **Fn5** key to execute the complete program at once. Correct the starting address to 0x00400000 before clicking the OK button.

| LAB TASK |
|---|
| **Write a MIPS program that finds the summation of odd numbers between 11 and 37** (both included)<br>Store start value 11 in register **t4**, end value 37 in register **t5** and the result value in register **t6**. |