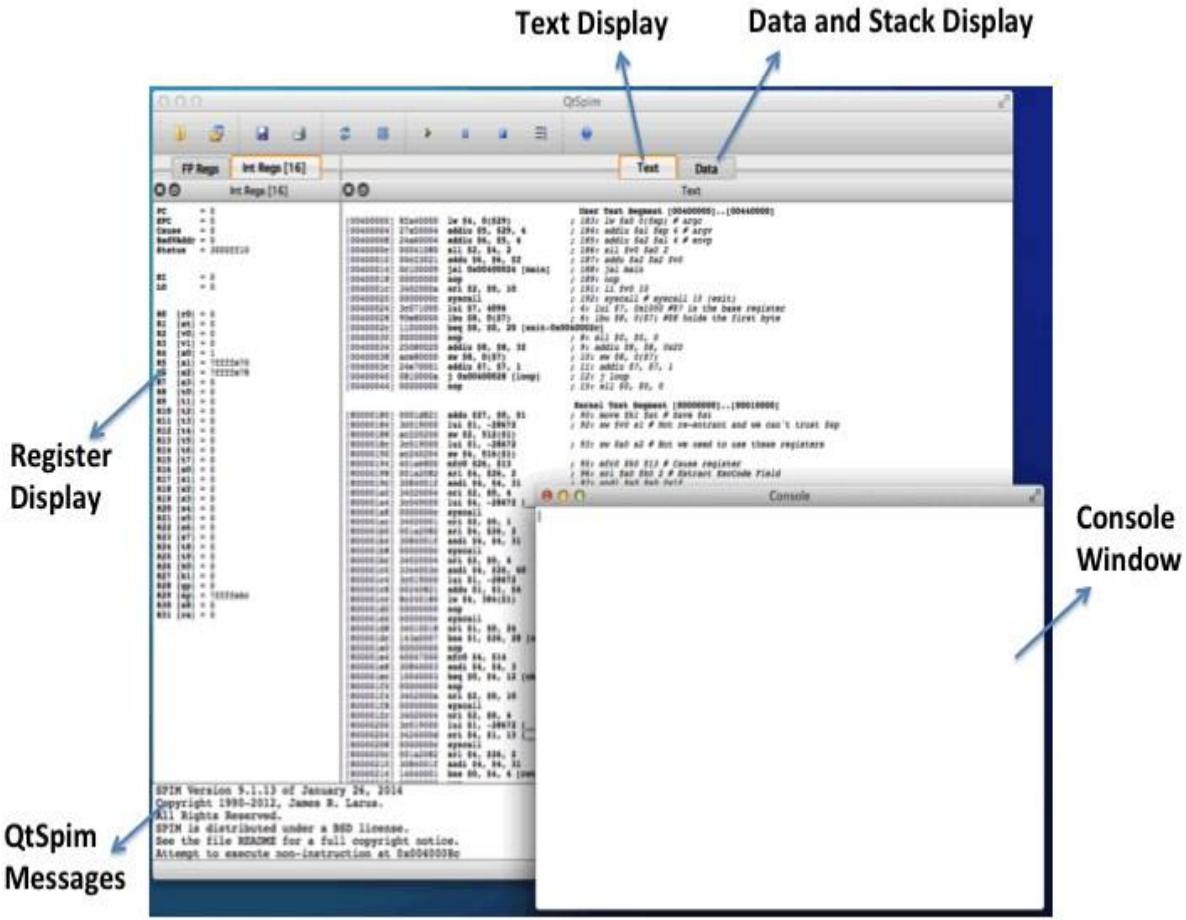
CENG 311

MIPS Examples



- **Register Display:** Shows the content of all 32 registers
- **Text Display:** Shows assembly language of source code and the corresponding machine instructions in hex, and the address of their memory location
- **Data and Stack Display:** Shows the sections of MIPS memory that holds the ordinary data, and the data which are pushed onto stack
- **Console Window:** characters output from simulated program is shown

How to install & use?

- http://spimsimulator.sourceforge.net/ Available for Linux, Mac, Windows
- write your assembly code in a text editor of your choice. Programing text editors supports mips highlighting (or have plugins that do)
- In spim go to File tab -> Reinitialize and Load File(alt+f+i) & select your .asm file from the browser

The simulator tab:

- To re-run you might want to clear the registers
- Either use run or step buttons (F5 or F10)

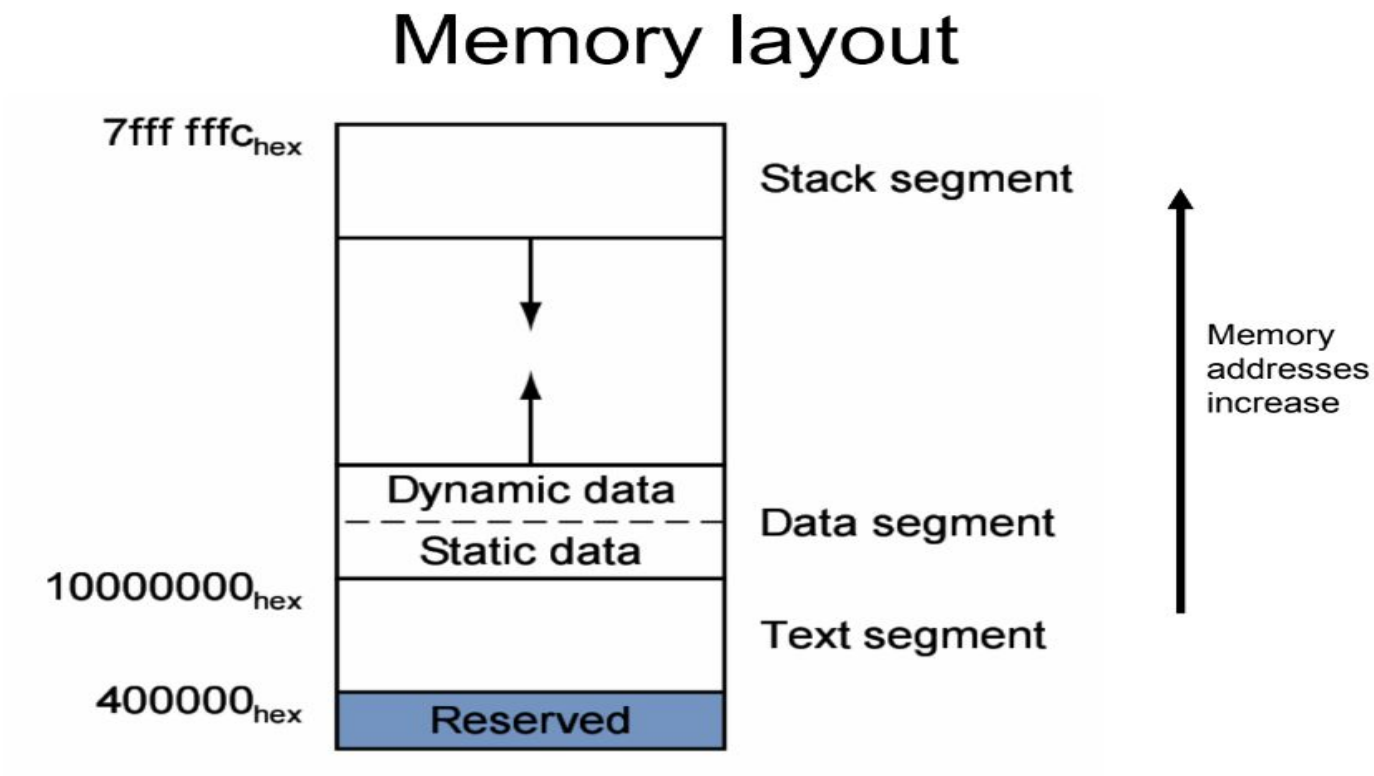
Other tabs:

- You can switch between binary, hex, and decimal at Registers/Data Segment tabs
- You can open & close the console at Window tab

REGISTERS:

Register Name	Register Number	Register Usage
$zero	$0	Hardware set to 0
$at	$1	Assembler temporary
$v0 - $v1	$2 - $3	Function result (low/high)
$a0 - $a3	$4 - $7	Argument Register 1
$t0 - $t7	$8 - $15	Temporary registers
$s0 - $s7	$16 - $23	Saved registers
$t8 - $t9	$24 - $25	Temporary registers
$k0 - $k1	$26 - $27	Reserved for OS kernel
$gp	$28	Global pointer
$sp	$29	Stack pointer
$fp	$30	Frame pointer
$ra	$31	Return address

Memory Layout



MIPS Assembly Template

```
# Comment giving name of program and description of function
# Template.s
# Bare-bones outline of MIPS assembly language program

            .data       # variable declarations follow this line
                        # ...

            .text       # instructions follow this line

main:                   # indicates start of code (first instruction to execute)
                        # ...

# End of program, leave a blank line afterwards to make SPIM happy
```

Variable Declarations

```
name:    storage_type    value(s)
```

```
var1:           .word   3       # create a single integer variable with initial value 3
array1:         .byte   'a','b' # create a 2-element character array with elements initialized
                                #   to  a  and  b
array2:         .space  40      # allocate 40 consecutive bytes, with storage uninitialized
                                #   could be used as a 40-element character array, or a
                                #   10-element integer array; a comment should indicate which!
```

Example-1 Addition of two numbers

```
.text  # section that contains instructions
.globl main # main function id declared as global
main:
    li $t0, 10 # 1st operand
    li $t1, 20 # 2nd operand
    li $s0, 0 # result will save in s0
    add $s0, $t0, $t1 #adds 2 numbers
    li $v0, 10 # to terminate the program choose the
    syscall # 10th systemcall service function(which is exit)
```

Example-1 Cont.

```
.data
num1: .word 10
num2: .word 20
result: .word 0


.text  # section that contains instructions
.globl main # main function id declared as global
main:
    la $t0, num1 # address of num1 label is loaded to to t0 reg
    lw $t3, 0($t0) # get the value from t0(stores num1 address) is loaded to t3 reg
    lw $t1, num2 # the variable(stored in address labeled with num2) is loaded in t1 reg
    lw $s0, result # result will save in s0
    add $s0, $t3, $t1 #adds 2 numbers
    li $v0, 10 # to terminate the program choose the
    syscall # 10th systemcall service function(which is exit)
```

Example-2

Take 2 numbers as input from user. Divide the numbers and print out the result

```
1  .text
2
3  main:
4
5  li $v0, 5 # syscall 5 reads integer from input and stores at $v0
6  syscall
7  move $t0, $v0 # save input integer for later use
8
9  li $v0, 5 # same operation will overwrite $v0,
10 syscall   # that's why we moved the value to another register
11 move $t1, $v0
12
13 divu $t0, $t1 # divide 2 number store the result in lo register
14 mflo $a0 # move lo register to $a0 for printing
15
16 li $v0, 1 # print integer
17 syscall
18
19 li $v0, 10 # end program
20 syscall
```

Try Yourself:

Calculate the average of 4 numbers 10, 20, 5, 2

Print the result