CS2610: Computer Organization and Architecture

Lab 6: Report

Devansh Singh Rathore(111701011)

Objective

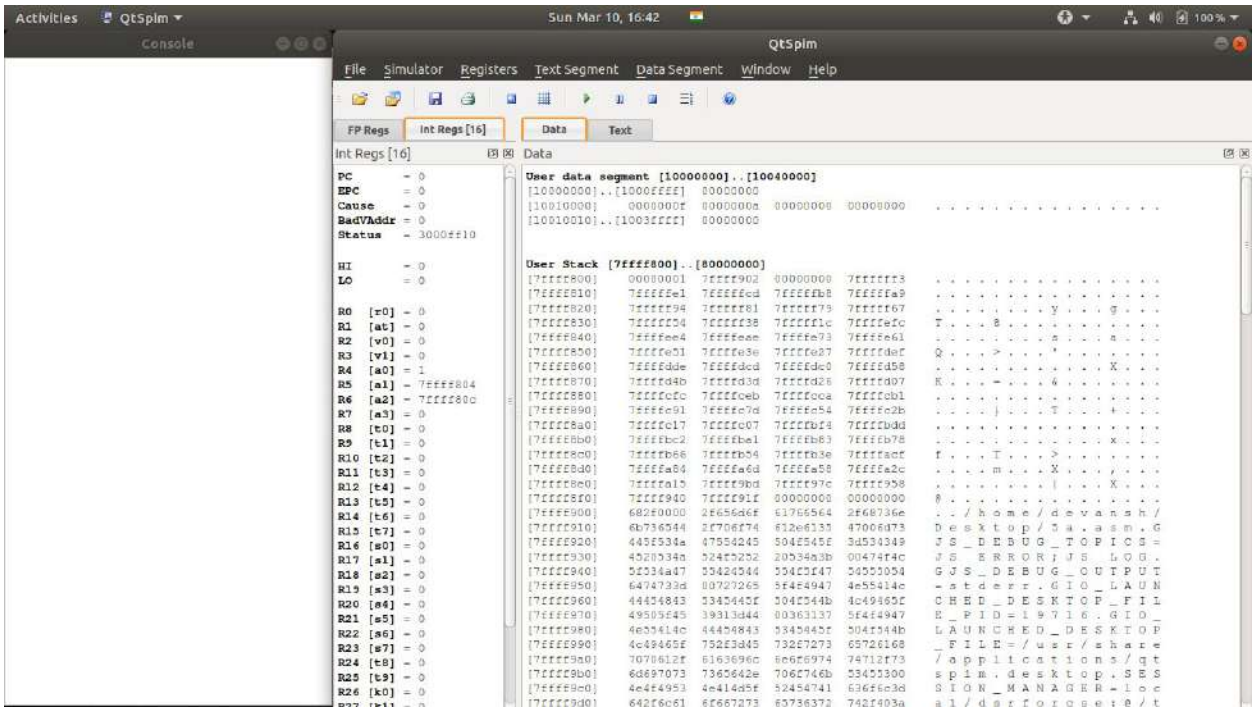
In this lab you will familiarize the basics of MIPS assembly programming using the QtSpim simulator.

Problem

Illustrate the following using QtSpim:
(1) Data transfer operations
(2) Storage of data in the data segment using ascii/asciiz
(3) Load and store a word, a half word and a byte
(4) Illustrate the use of syscall codes for the following operations
(5) Reading and printing an integer and string
(6) Reading and printing a floating point number

Implementation

Initial Snapshot of Registers, Data Segments, and Console:



(1) Data transfer operations::

CODE:

a)

```
#1a::

.data
F: .word 0xF
A: .word 0xA
.text
lw $t0, F
lw $t1, F
lw $t2, F
lw $t3, F
lw $t4, F
lw $t5, F
lw $t6, F
lw $t7, F
lw $t8, F
lw $t9, F
lw $s0, A
lw $s1, A
lw $s2, A
lw $s3, A
lw $s4, A
lw $s5, A
lw $s6, A
lw $s7, A
move $s4, $t6
move $t5, $s7
syscall
```

b)

```
#1b::

.data
F: .word 0xFFFF
A: .word 0xAAAA
.text
lw $t0, F
lw $t1, F
```

```
lw $t2, F
lw $t3, F
lw $t4, F
lw $t5, F
lw $t6, F
lw $t7, F
lw $t8, F
lw $t9, F
lw $s0, A
lw $s1, A
lw $s2, A
lw $s3, A
lw $s4, A
lw $s5, A
lw $s6, A
lw $s7, A
syscall
```

c)

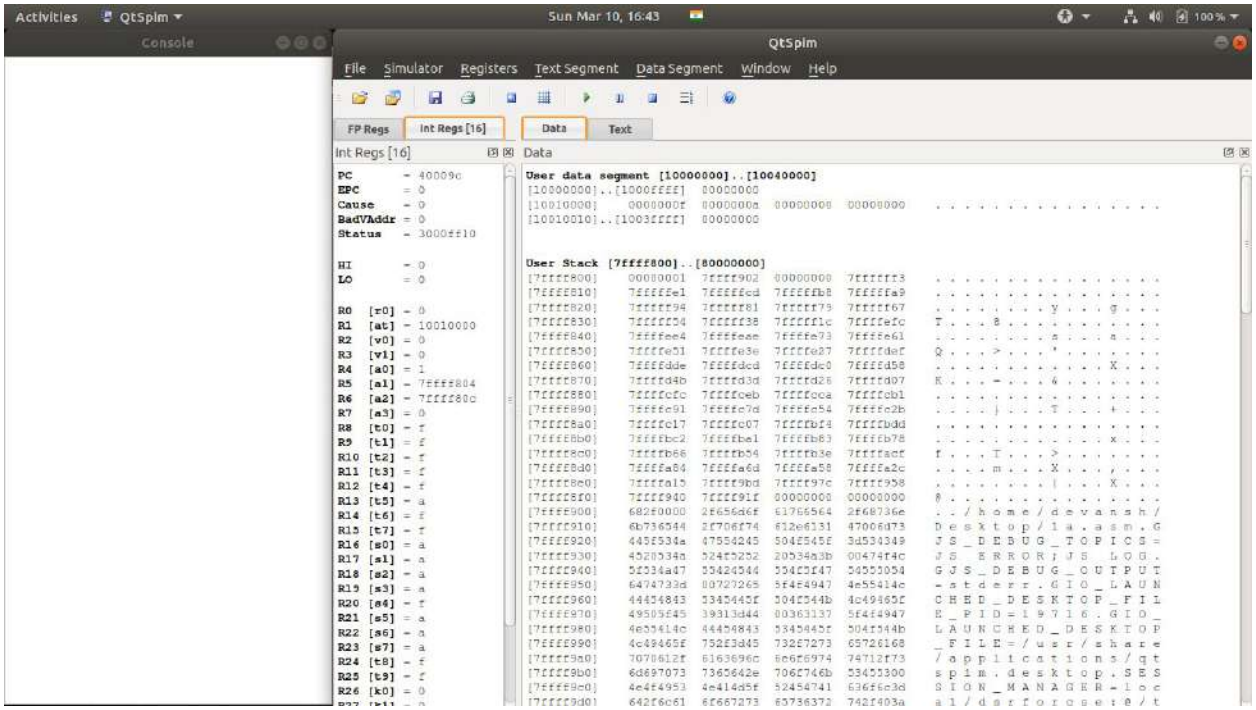
```
#1c::

.data
F: .word 0xFFFFFFFF
A: .word 0xAAAAAAAA
.text
lw $t0, F
lw $t1, F
lw $t2, F
lw $t3, F
lw $t4, F
lw $t5, F
lw $t6, F
lw $t7, F
lw $t8, F
lw $t9, F
lw $s0, A
lw $s1, A
lw $s2, A
lw $s3, A
lw $s4, A
lw $s5, A
lw $s6, A
```

```
lw $s7, A
syscall
```

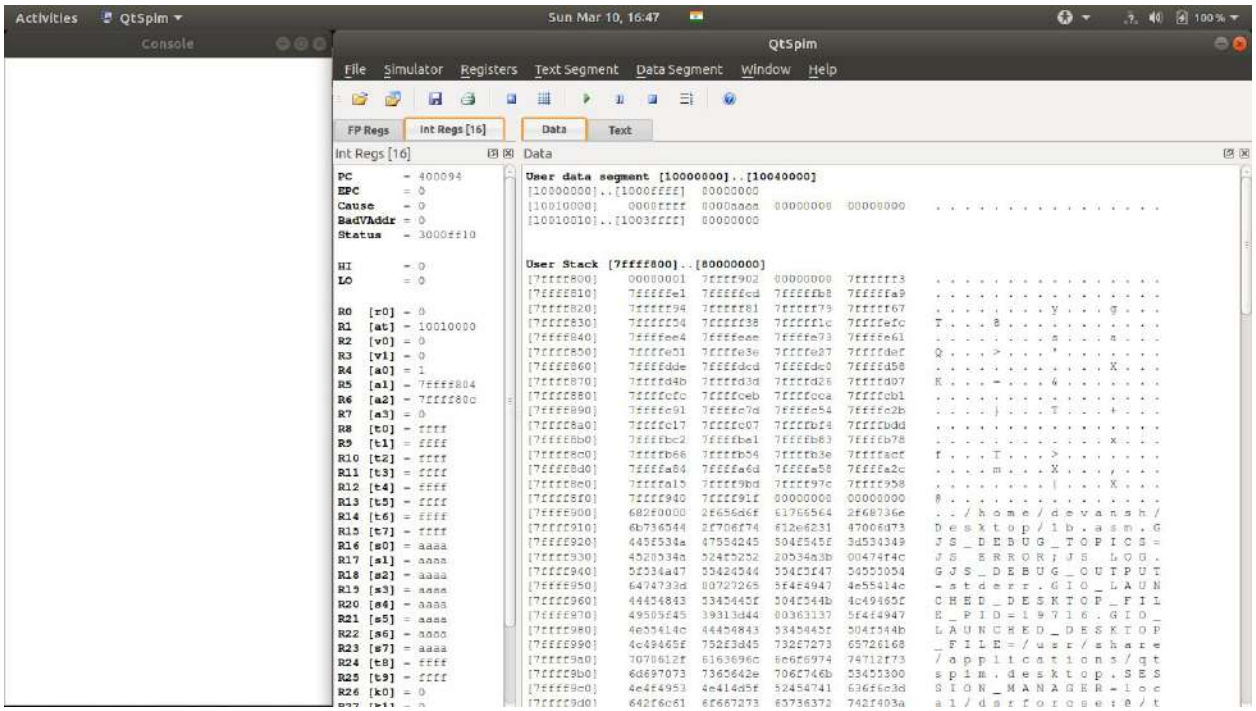
d)

```
#1d::

.data
F: .word 123456789

.text
lw $t0, F
lw $t1, F
lw $t2, F
lw $t3, F
lw $t4, F
lw $t5, F
lw $t6, F
lw $t7, F
lw $t8, F
lw $t9, F
lw $s0, F
lw $s1, F
lw $s2, F
lw $s3, F
lw $s4, F
lw $s5, F
lw $s6, F
lw $s7, F
syscall
```

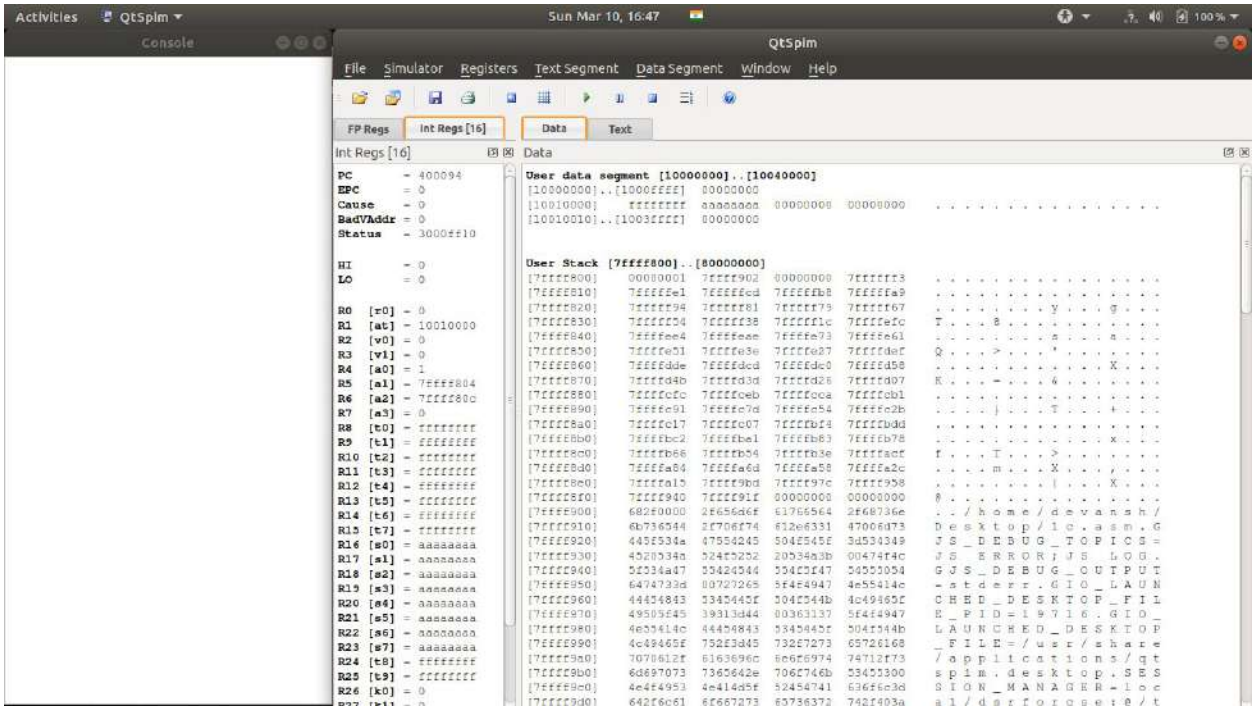
CHANGES IN REGISTERS/DATA SEGMENTS(Final Snapshots after program execution):
a)


Activities
QtSpim
Sun Mar 10, 16:43
Console
QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text
Int Regs [16]
PC = 40009c
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 7fffe804
R6 [a2] = 7fffe80c
R7 [a3] = 0
R8 [t0] = f
R9 [t1] = f
R10 [t2] = f
R11 [t3] = f
R12 [t4] = f
R13 [t5] = a
R14 [t6] = f
R15 [t7] = f
R16 [s0] = a
R17 [s1] = a
R18 [s2] = a
R19 [s3] = a
R20 [s4] = f
R21 [s5] = a
R22 [s6] = a
R23 [s7] = a
R24 [t8] = f
R25 [t9] = f
R26 [k0] = 0
Data
User data segment [10000000]..[10040000]
User Stack [7fffe800]..[80000000]

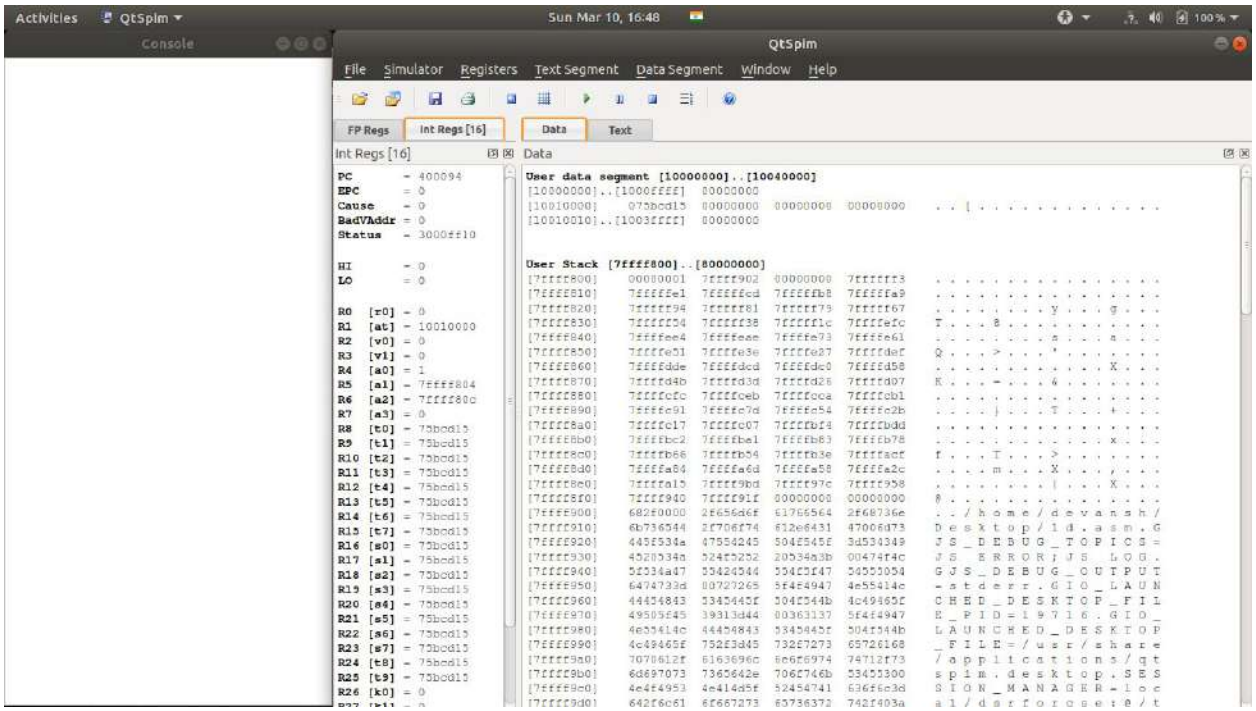
b)


Activities
QtSpim
Sun Mar 10, 16:47
Console
QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text
Int Regs [16]
PC = 400094
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 7fffe804
R6 [a2] = 7fffe80c
R7 [a3] = 0
R8 [t0] = ffff
R9 [t1] = ffff
R10 [t2] = ffff
R11 [t3] = ffff
R12 [t4] = ffff
R13 [t5] = ffff
R14 [t6] = ffff
R15 [t7] = ffff
R16 [s0] = aaaa
R17 [s1] = aaaa
R18 [s2] = aaaa
R19 [s3] = aaaa
R20 [s4] = aaaa
R21 [s5] = aaaa
R22 [s6] = aaaa
R23 [s7] = aaaa
R24 [t8] = ffff
R25 [t9] = ffff
R26 [k0] = 0
Data
User data segment [10000000]..[10040000]
User Stack [7fffe800]..[80000000]

c)



d)



EXPLANATION:

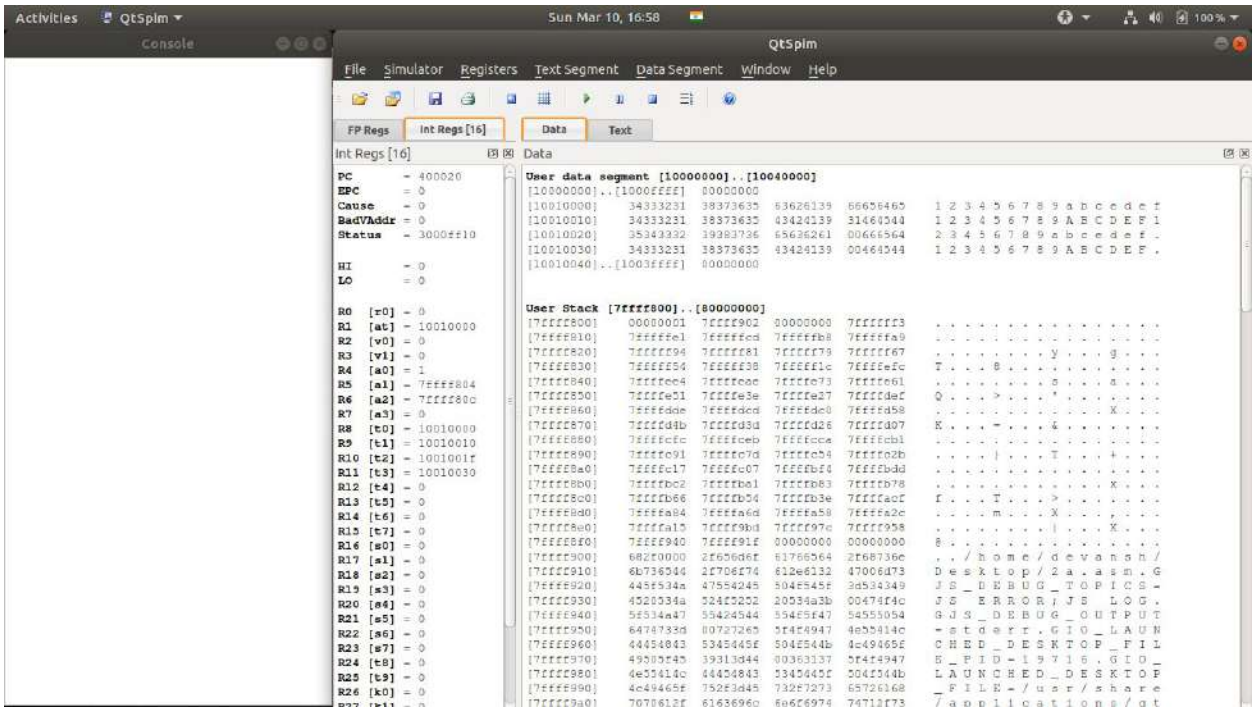
Firstly two data elements are created A and F having hex values 0xA and 0xF respectively. Then, respective values are stored in their respective desired places(registers) using the lw command and further shifts are made if required. In the part (d), the number stored in data element is then stored in the registers in its hexadecimal form.

(2) Storage of data in the data segment using ascii/asciiz:

CODE:

```
#2::

.data
str1: .ascii "123456789abcedef"
str2: .ascii "123456789ABCDEF"
str3: .asciiz "123456789abcedef"
str4: .asciiz "123456789ABCDEF"

.text
la $t0, str1
la $t1, str2
la $t2, str3
la $t3, str4
syscall
```

CHANGES IN REGISTERS/DATA SEGMENTS(Final Snapshots after program execution):



EXPLANATION:

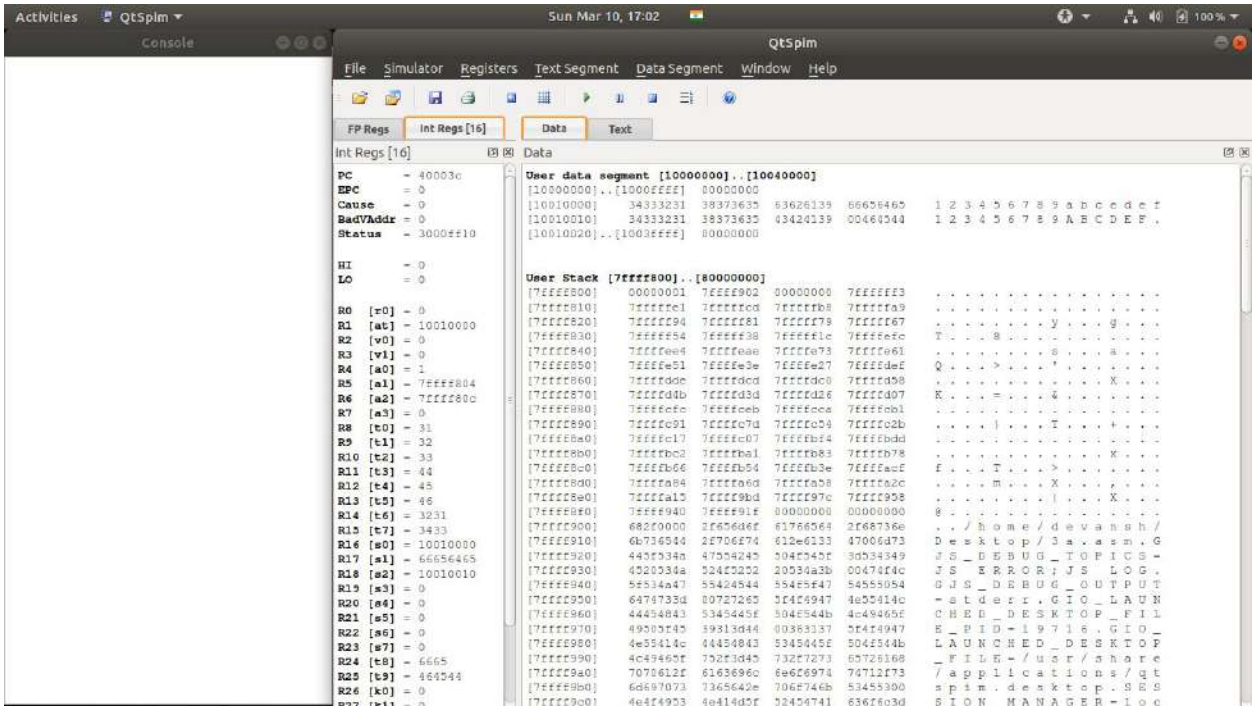
The address of ascii/asciiz data elements are loaded using the la command which is actually a combination of two mips command(lui+ori).

(3) Load and store a word, a half word and a byte:

CODE:

```
#3a::

.data
str1: .ascii "123456789abcedef"
str2: .ascii "123456789ABCDEF"

.text
la $s0, str1
la $s2, str2

lb $t0, 0($s0)
lb $t1, 1($s0)
lb $t2, 2($s0)
lb $t3, 12($s2)
lb $t4, 13($s2)
lb $t5, 14($s2)
lh $t6, 0($s2)
lh $t7, 2($s2)
lh $t8, 14($s0)
lw $t9, 12($s2)
lw $s1, 12($s0)
syscall
```

CHANGES IN REGISTERS/DATA SEGMENTS(Final Snapshots after program execution):



EXPLANATION:

In ascii data type each single element is of one byte, so to store bytes lb operation is used, to store half-words lh operations is used, and to load words lw operation is used. And according to their indices, the values are then stored in desired registers by keeping the immediate values accordingly.

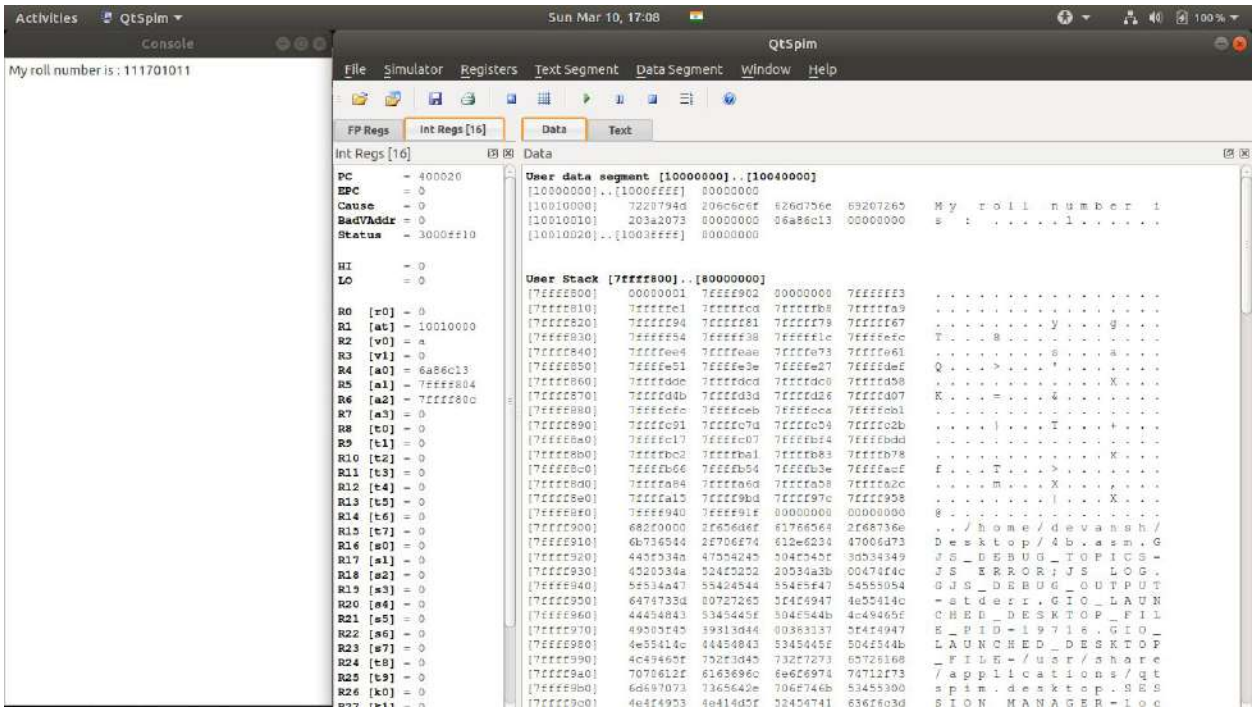
(4) Illustrate the use of syscall codes for the following operations:

CODE:

```
#4::

.data
text:   .asciiz "My roll number is : "
roll: .word 111701011
.text
main:
# Printing out the text
li $v0, 4
la $a0, text
syscall
```

```
# Printing the roll no.
li $v0, 1
lw $a0, roll
syscall

# End Program
li $v0, 10
syscall
```

CHANGES IN REGISTERS/DATA SEGMENTS(Final Snapshots after program execution):



EXPLANATION:

Firstly, we have created two containers(data elements), one having the text to be printed and other having my roll number to be printed. The opcode of printing the text is 4, and for printing the word is 1, and hence the v0 value is operated so as to perform the required operation.The program is terminated by opcode value of 10 for v0.

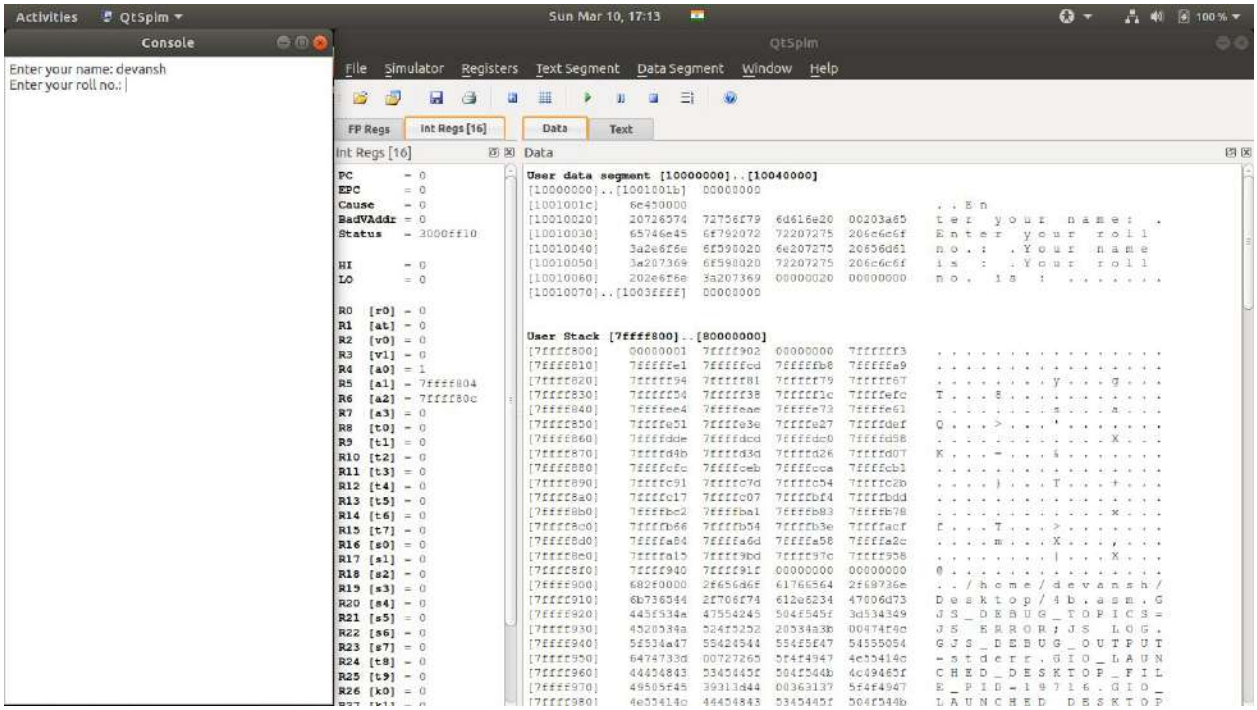
(5) Reading and printing an integer and string:

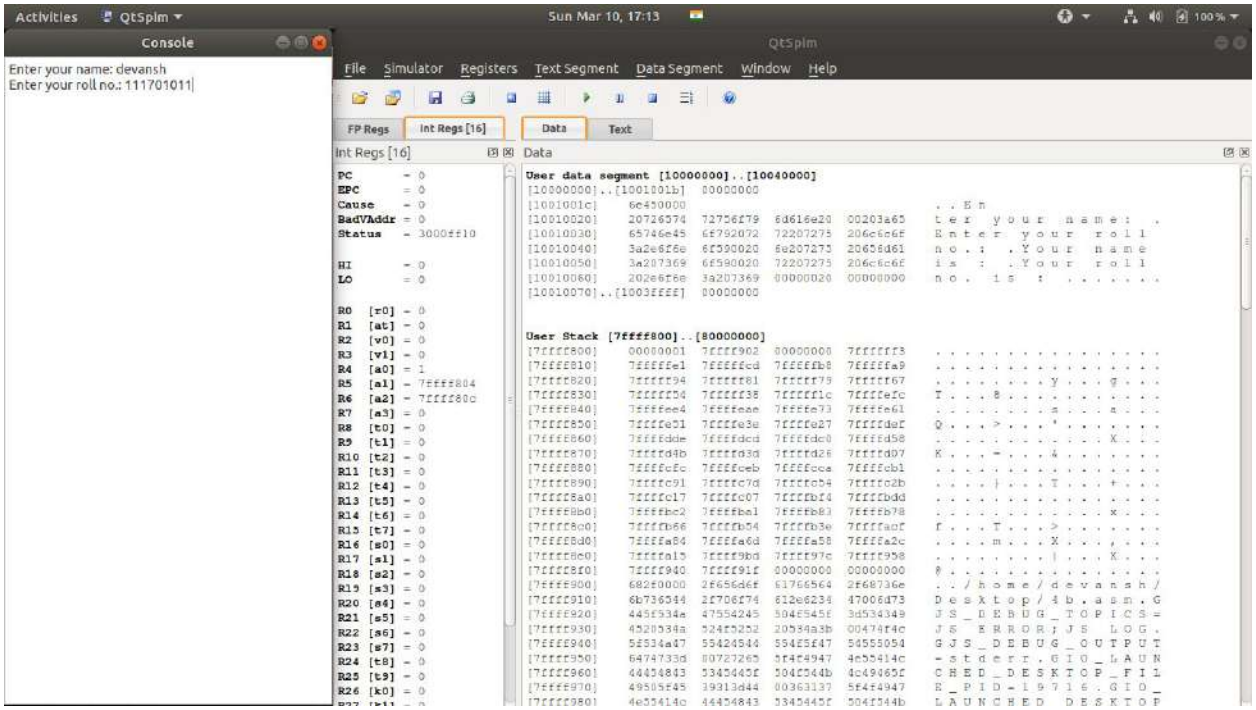
CODE:

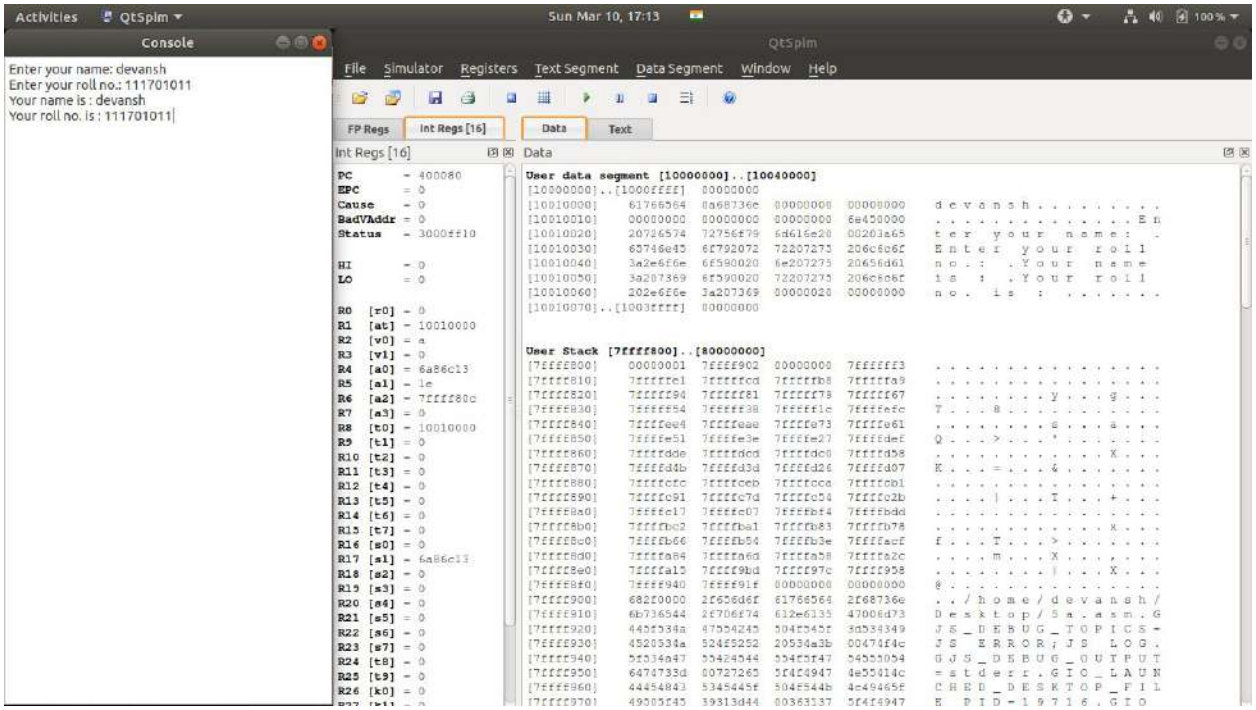
```
#5::

.data
buffer: .space 30
Ename: .asciiz "Enter your name: "
Eroll: .asciiz "Enter your roll no.: "
tname: .asciiz "Your name is : "
troll: .asciiz "Your roll no. is : "

.text

main:
# Printing out the text for name
li $v0, 4
la $a0, Ename
syscall

# Getting user input
li $v0, 8       #to take string input
la $a0, buffer  #load byte space into address
li $a1, 30      #allot byte space for string
move $t0, $a0   #save string to t0
syscall

# Printing out the text for rollno
li $v0, 4
la $a0, Eroll
syscall

# Getting user input
li $v0, 5
```

```
syscall
move $s1, $v0

# Printing out the name text
li $v0, 4
la $a0, tname
syscall

# Printing the name
li $v0, 4
la $a0, buffer  #reload byte space to primary address
move $a0, $t0   #move t0 value to primary address
syscall

# Printing out the roll no. text
li $v0, 4
la $a0, troll
syscall

# Printing the roll no.
li $v0, 1
move $a0, $s1
syscall

# End Program
li $v0, 10
syscall
```

CHANGES IN REGISTERS/DATA SEGMENTS(Final Snapshots after program execution):







EXPLANATION:

We have created 5 data elements one for the buffer for the input of the name string, and other four corresponding to the four lines to be printed on console. Firstly we print the text to allow input of name by loading immediate value of 4 in v0 register. Then we get input for name with allowed buffer space, by loading immediate value of 8 to v0. Similarly for the input of the roll no but with no buffer complexity. Now, we need to print texts for “Your name”, and reload byte

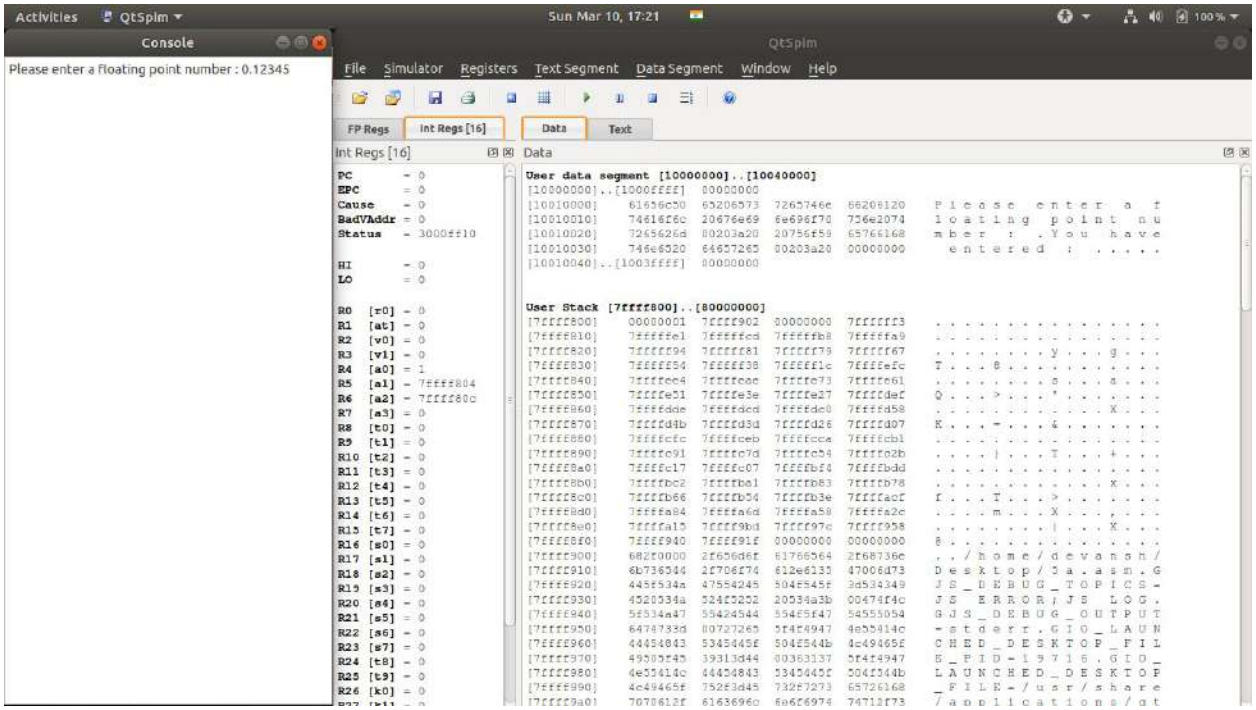
space to primary address, followed by name string at primary address. Similarly we print the roll no.

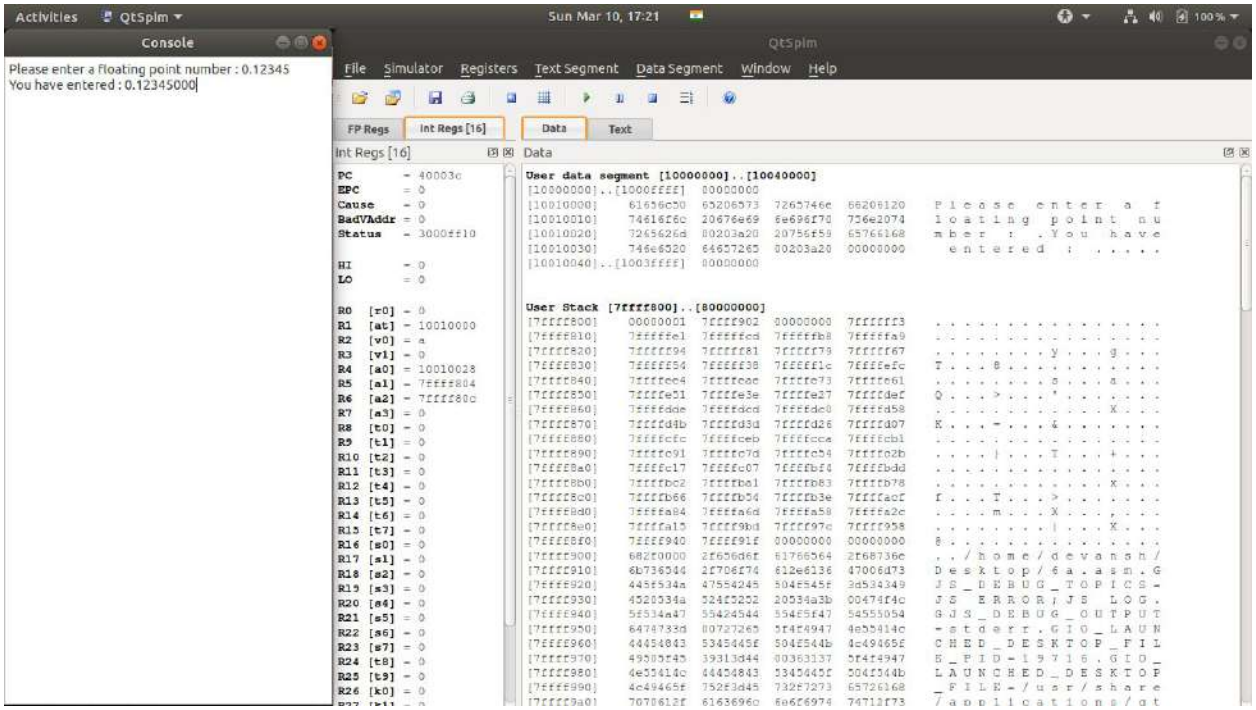
(6) Reading and printing a floating point number:

CODE:

```
#6a:

.data
E: .asciiz "Please enter a floating point number : "
W: .asciiz "You have entered : "
fl: .float 0.0
.text

main:
lwc1 $f4, fl

# Printing out the text for E
li $v0, 4
la $a0, E
syscall

# Getting user input for float value
li $v0, 6
syscall

# Printing out the W text
li $v0, 4
la $a0, W
syscall

# Printing the float
li $v0, 2
```

```
add.s $f12, $f0, $f4
syscall

# End Program
li $v0, 10
syscall
```





EXPLANATION:

Program writes the required text by loading 4 as immediate to the v0, reads float value with opcode 6, and prints the float value with opcode 5. Lwc1 is used to load the data element into a register to store the user input float value. Additionally we print the float value using add.s(for single precision) floating point numbers.

Observations

Key observations in performing the above executions in mips:

1.

```
li $v0, 1  #for printing the word
li $v0, 2  #for printing the float
li $v0, 4  #for printing the text(ascii/asciiz)
li $v0, 5  #taking input for text
li $v0, 6  #for taking float input
li $v0, 8  #for taking string input
li $v0, 10  #for terminating the program
```

2. Asciiz is different from ascii, as asciiz is terminated by '\0' and hence it is often referred as C-string.

3. To take string input in mips, we need to create an extra buffer space in primary address and then store the actual input in it.

4. "Lwc1" command is an coprocessor command used to load word as a floating value. Floating point handled by co-processor 1, one of 4 co-processors. MIPS floating point registers also called co-processor 1 registers. MIPS floating point instructions called co-processor 1 instructions.
Registers named f0-f31.
Each register is 32 bits. (For MIPS-32)

5. "Add.s " instruction enables single precision floating point addition.

Conclusions

We dealt with the basics of MIPS assembly programming using the QtSpim simulator and performed operations like loading, storing, taking input, printing output, and related tasks.