CS2610: Computer Organization and Architecture

Lab 8: Report

Devansh Singh Rathore(111701011)

Objective

In this lab you will familiarize with MIPS procedure calls, two dimensional arrays,delayed branches, and delayed loads using the QtSpim simulator.

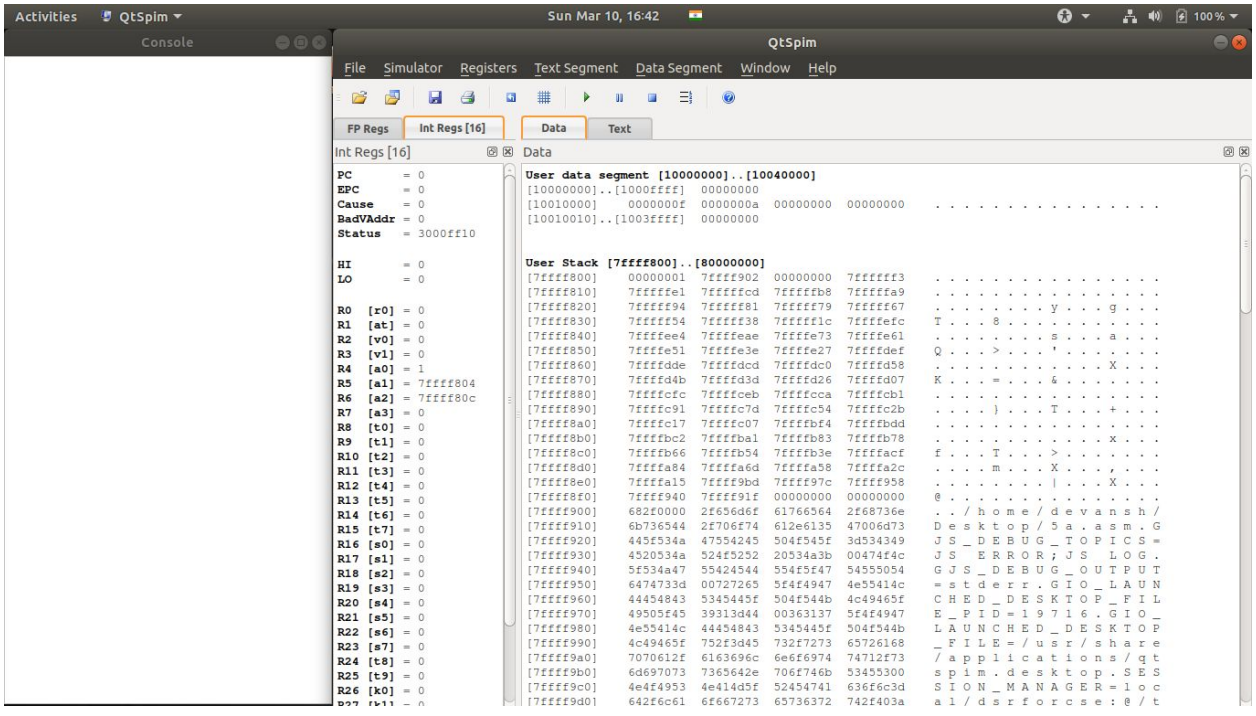
Problem

Illustrate the following using QtSpim:

(1) Introduction to procedures.
(2) Delayed branches.
(3) Delayed loads.
(4) Introduction to 2-D arrays.

Implementation

Initial Snapshot of Registers, Data Segments, and Console:

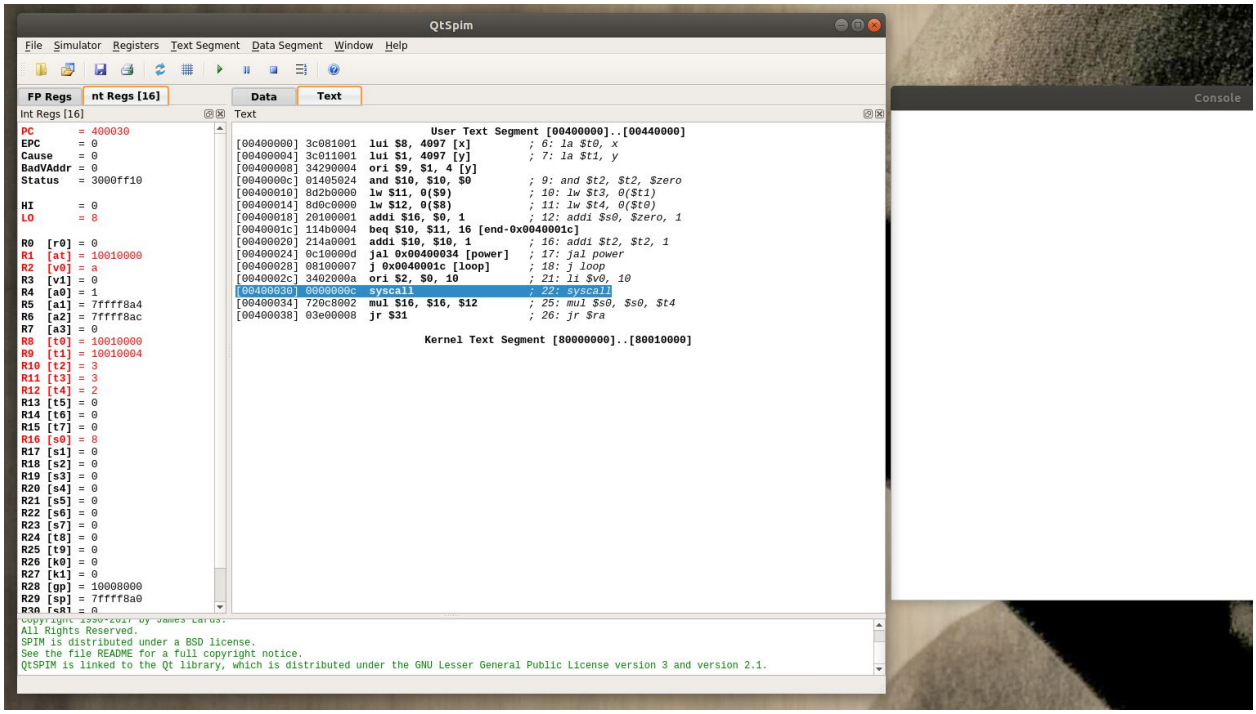


(1) Introduction to procedures::

CODE:

```
.data
x: .word 2 //two numbers taken as word.
y: .word 3

.text
la $t0, x //loading address of each of the two numbers
la $t1, y //in two different registers.

and $t2, $t2, $zero //initializing the loop index to 0.
lw $t3, 0($t1) //initializing the loop termination index as the value of
power.
lw $t4, 0($t0) //taking value of x in a register for multiplication in
loop.
addi $s0, $zero, 1 //initializing the product as 1.

loop: //loop module
beq $t2, $t3, end  //comparing loop index with termination index.
addi $t2, $t2, 1   //incrementing loop index.
jal power  //calling module power.
j loop   //continuing the loop.

end:
li $v0, 10 //for terminating the program.
syscall

power:
mul $s0, $s0, $t4 //for multiplying the current product with x value.
jr $ra //returning back to loop.
```

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



EXPLANATION:

The implementation involves use of power procedure to multiply x value to the total product initialized to one, and the same process is carried on y no.of times through a loop module.

(2) Illustration of Delayed Branches:

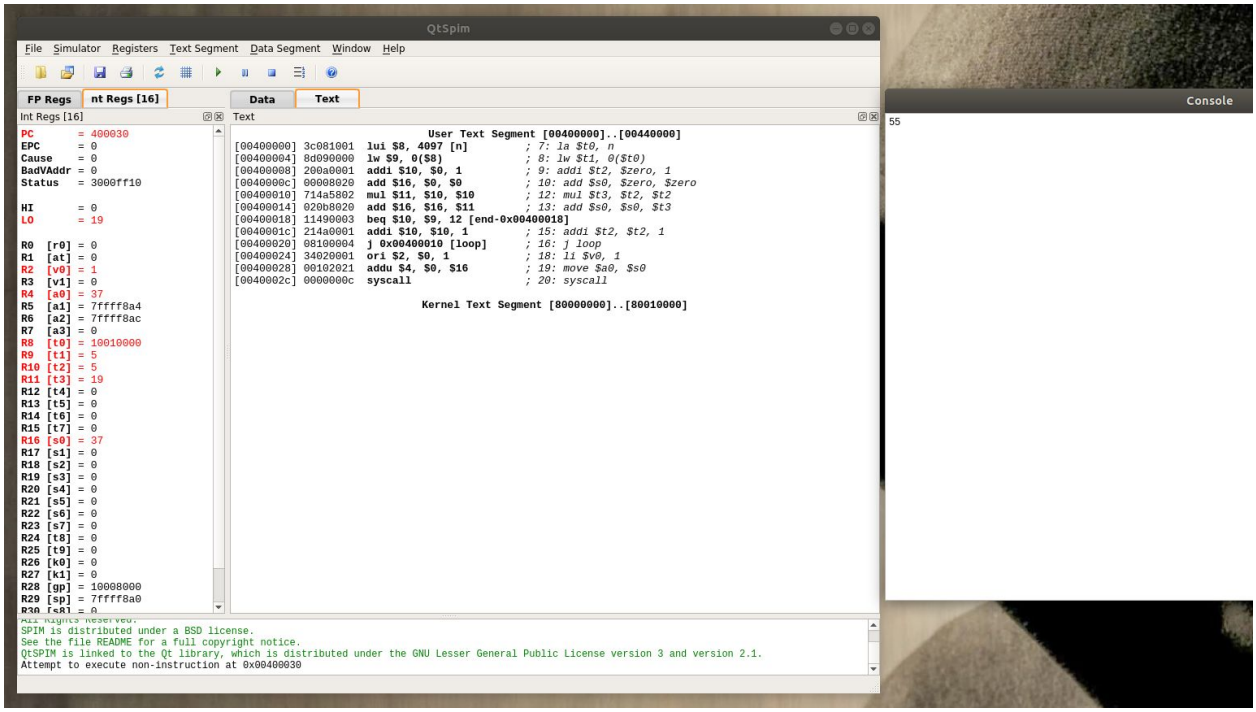
CODE:

```
.data
n: .word 5 //taking n as a word.

.text

main:  //main module
la $t0, n  //loading address of n.
lw $t1, 0($t0)  //storing value of n in a register.
add $t2, $zero, $zero  //initializing loop index to zero.
add $s0, $zero, $zero  //storing initial sum as zero.

loop:
beq $t2, $t1, end
addi $t2, $t2, 1  //incrementing loop index by one.
mul $t3, $t2, $t2  //taking square of loop index into another register.
add $s0, $s0, $t3  //adding the square to sum register.
j loop  //continuing the loop.
```

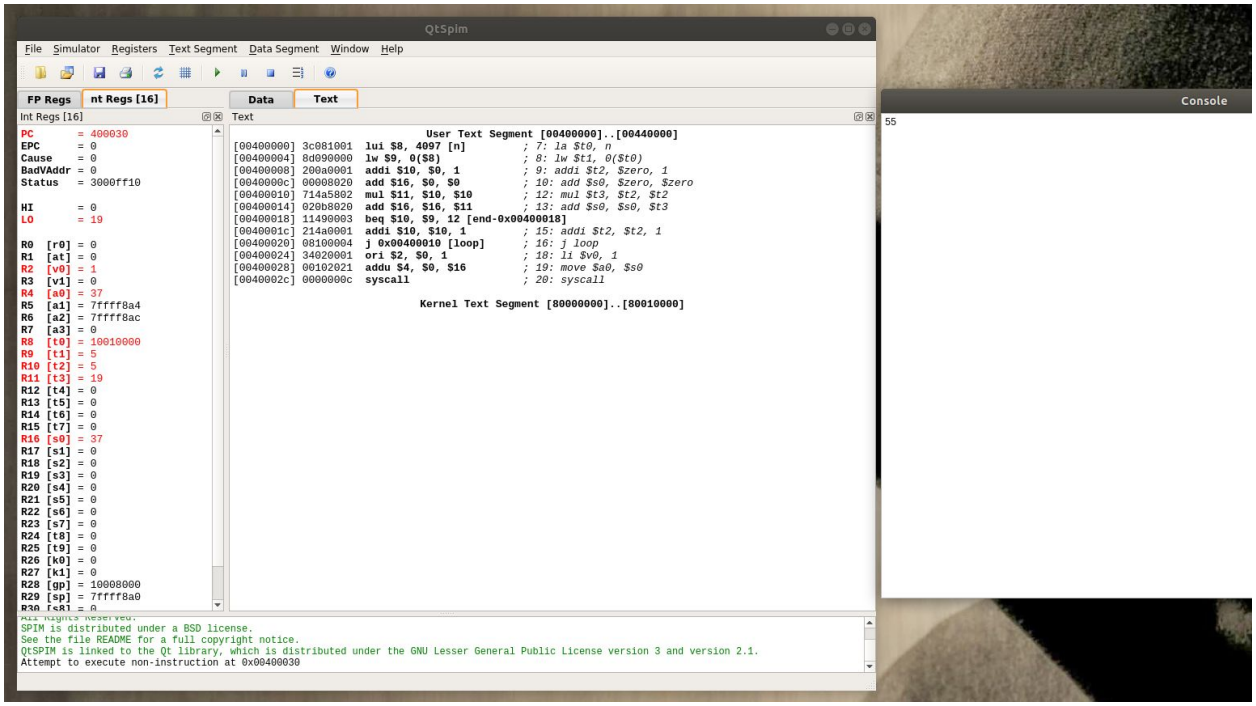
```
end:  //end module
li $v0, 1  //for printing the sum as a word.
move $a0, $s0
syscall
li $v0, 10  //for terminating the program.
syscall
```

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

Q.2a) Without enabling Delayed Branches.



Q.2b) After enabling Delayed Branches.



EXPLANATION:

The result of the code was not affected by enabling the Delayed branch, and no error occurred. So there was no requirement to arrange the instruction sequence. Both the cases gave out in correct results. The technique used was to squaring the loop index itself and adding it to the total sum until the loop index gets equal to n and then exit the loop.

(3) Illustration of Delayed Loads:

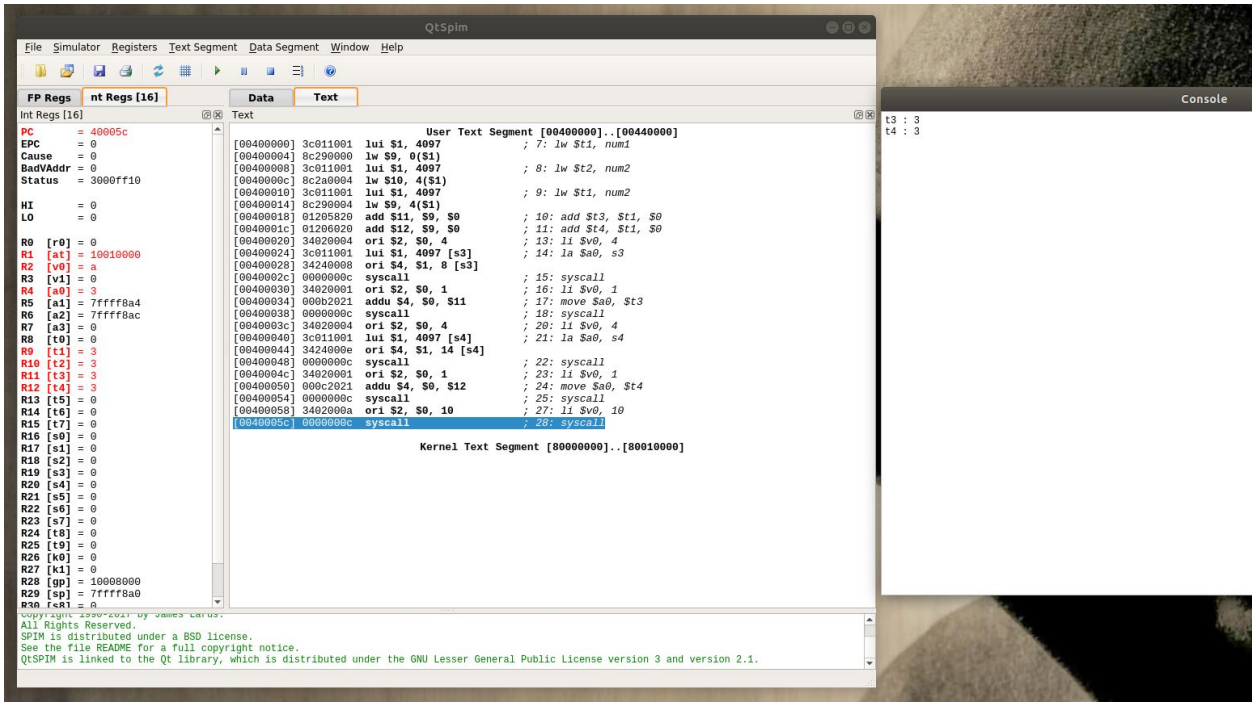
CODE:

```
.data
num1: .word 2  //taking num1 as a word.
num2: .word 3  //taking num2 as a word.
s3: .asciiz "t3 : "  //taking ascii strings for printing final results.
s4: .asciiz "\nt4 : "
.text
lw $t1, num1 //provided code snippet.
lw $t2, num2
lw $t1, num2
add $t3, $t1, $0
add $t4, $t1, $0

li $v0, 4  //for printing ascii string s3.
la $a0, s3
syscall
```

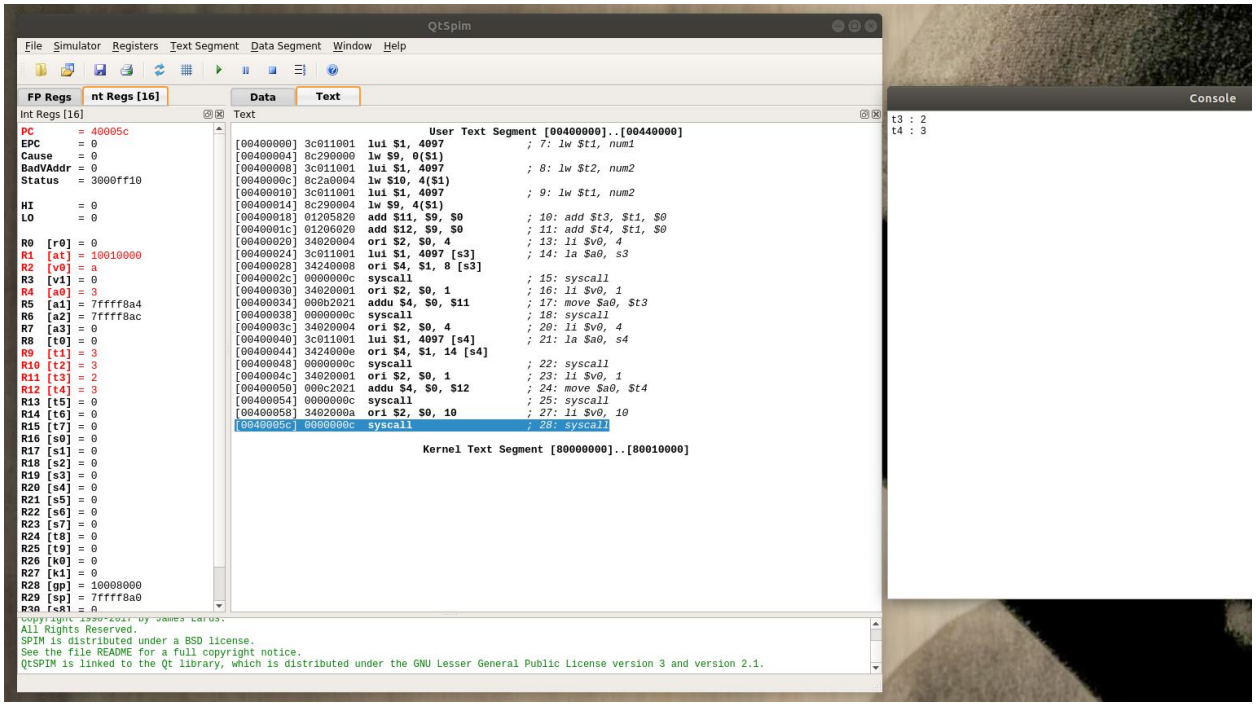
```
li $v0, 1  //for printing value of t3.
move $a0, $t3
syscall

li $v0, 4  //for printing ascii string s4.
la $a0, s4
syscall
li $v0, 1  for printing value of t4.
move $a0, $t4
syscall

li $v0, 10
syscall
```

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

Q.3a) Without Delayed Loads:



Q.3b) With Delayed Loads:



EXPLANATION:

In the first case, the result was t3=t4=3, and in the second case after enabling the Delayed loads t3=2, t4=3 was the result. It was because, in the prior case, there was no delayed branches and hence the same value(3) was stored in both of the registers, but in the second case, due to delayed branches, the instruction to load num1 in $t1 was delayed, and after the loading of num2 in $t1, and $t2 was done, num1 was again loaded to $t1, hence keeping the numbers distinct, and producing the final output accordingly.

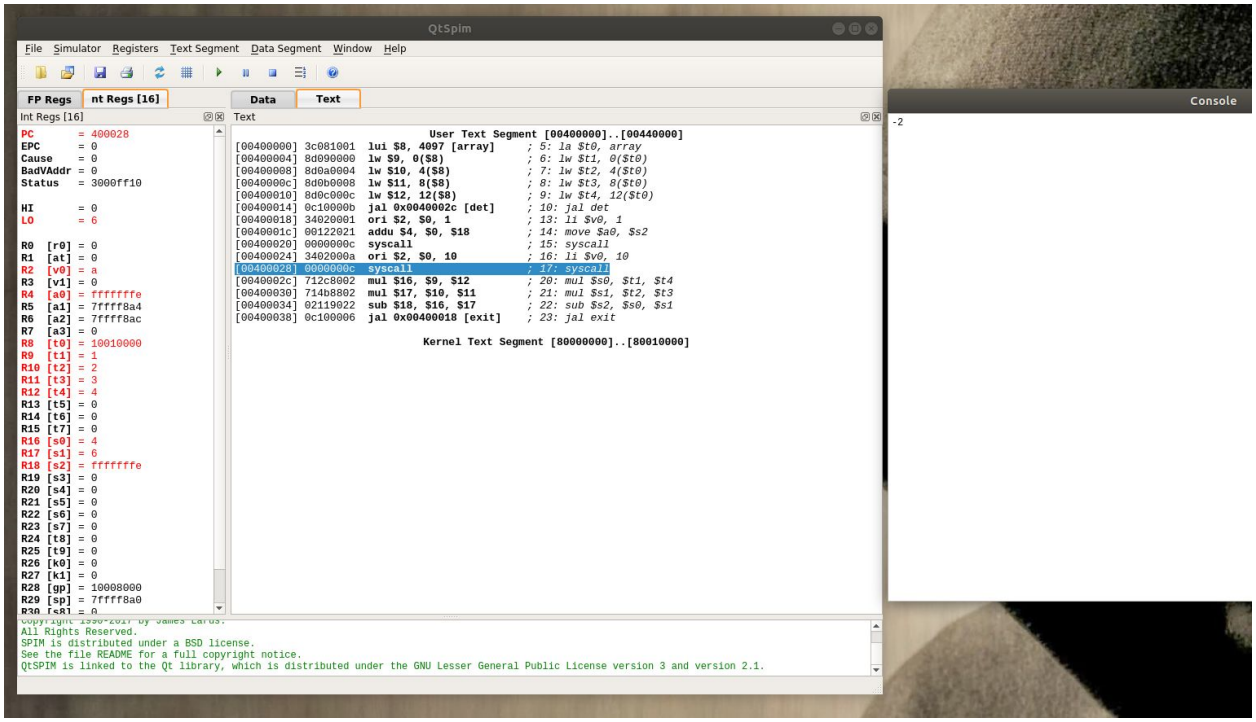
(4) Introduction to stack operations:

CODE:

```
.data
array: .word 1 2 3 4 //storing elements of 2x2 matrix in an
array.

.text
la $t0, array  //loading address of starting index of
array.
lw $t1, 0($t0) //storing values of 2x2 matrix in different
registers.
```

```
lw $t2, 4($t0)
lw $t3, 8($t0)
lw $t4, 12($t0)
jal det  //calling determinant(det) procedure.

exit:  //exit procedure
li $v0, 1 //for printing the final result
move $a0, $s2
syscall
li $v0, 10 //for terminating the program.
syscall

det:  //determinant(det) procedure.
mul $s0, $t1, $t4  //carrying out the required
calculations.
mul $s1, $t2, $t3
sub $s2, $s0, $s1
jal exit  //calling exit once done.
```

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



EXPLANATION:

The idea was to simply load the values from the array accordingly in four different registers and carry out the arithmetic operation, by procedure call to det(determinant) method.

Observations

Key observations in performing the above executions in mips:

1. For calling procedures and returning from procedures we need to use several jump operations jal, jr, j, etc.
2. Using procedure calls reduces the lines of code and makes the code look simpler.
3. We need to have some passing arguments in the procedures and some temporary registers for procedure call techniques.
4. Delayed operations allows to execute next few instructions (irrespective of changes created by certain specifies instructions) delaying the execution of certain specified instructions.
5. Delayed operations may result in causing different outputs as compared to when the delayed operations were not enabled.

Conclusions

We dealt with the basics of MIPS procedure calls, two dimensional arrays, delayed branches, and delayed loads using the QtSpim simulator.