**MATRIX MANIPULATOR**

Laboratory Project Report submitted for

**Computer Organisation and Architecture**

**(CSE2011)**

Submitted by

|  |  |
| --- | --- |
| **PRERNA BHARTI**  **Registration No.: 1641012048** | **NIKHIL RANJAN NAYAK**  **Registration No.: 1641012040** |
| **SATYASMITH RAY**  **Registration No.: 1641012108** | **BAIBHAV SWAIN**  **Registration No.: 1641012206** |
| **SONU BINAY**  **Registration No.: 1641012316** | |

**(CSE ‘F’ 4th SEMESTER)**



**Department of Electronics & Communication Engineering**

**Institute of Technical Education and Research**

**(Faculty of Engineering)**

**SIKSHA ‘O’ ANUSANDHAN (Deemed to be University)**

**Bhubaneswar, Odisha, India**

**2018**

# Declaration

We, the undersigned students of B. Tech. of ComputerScience and Engineering Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled “**MATRIX MANIPULATOR**” submitted to **Siksha ‘O’ Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Organization and Architecture (CSE2011)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

|  |  |
| --- | --- |
| **SATYASMITH RAY**  **Registration No.: 1641012108** | **SONU BINAY**  **Registration No.: 1641012316** |
| **NIKHIL RANJAN NAYAK**  **Registration No.: 1641012040** | **BAIBHAV SWAIN**  **Registration No.: 1641012206** |

**PRERNA BHARTI**

**Registration No.: 1641012048**

**DATE: 9th April 2018**

**PLACE: Institute of Technical Education and Research, Siksha ‘O’ Anusandhan (Deemed to be University), Bhubaneswar**

# Abstract

In computation involving matrices it is frequently necessary to interchange or rearrange rows or columns of a matrix. If the work is being done longhand or with a desk calculator, it is desirable to be able to perform the rearrangement without having to erase or rewrite numbers. In mathematics, matrix addition or subtraction, multiplication or the matrix product or any other arithmetic operation is a binary operation that produces a matrix from two matrices.

The matrix manipulator was devised for the use in the Bureau’s statistical Engineering Laboratory for calculation with incidence matrices, i.e., matrices whose element are all 0’s or 1’s. So, through this project we will show how the matrix manipulation is taking place using MIPS programming language. MIPS is a reduced instruction set computer set architecture developed by MIPS Technologies. The early MIPS architecture was 32-bit with 64-bit versions added later. Also, the project is compiled to show Matrix addition of two different matrices, subtraction of two different matrices, multiplication, transpose, determinant, scaling of matrix.

# Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **Serial No.** | **Chapter No.** | **Title of the Chapter** | **Page No.** |
|  | 1 | Introduction | 1 |
|  | 2 | Problem Statement | 2 |
|  | 3 | Brief Description | 3 |
|  | 4 | Steps of the Algorithms/Flow Diagram | 4 |
|  | 5 | Source Code using MIPS | 6 |
|  | 6 | Testing | 20 |
|  | 7 | Conclusion | 25 |
|  |  | References | 26 |

# 1. Introduction

This project has been done keeping in mind all the basic concepts taught in Computer Organization and Architecture. This project deals with the matrix manipulation of two matrices. It is well known that matrices can be used in several key areas of science to provide meaning to data. Often these matrices can be manipulated to provide a solution based on the important properties of the matrices like Matrix addition, subtraction of two different matrices, also scaling up with a constant multiplication and transpose of matrix.

The user is prompted to enter two 3X3 matrixes. The user is also asked to enter the choice of operation to be done with the inputted matrices i.e.

1. Matrix Addition
2. Matrix Subtraction
3. Multiplication
4. Transpose
5. Determinant
6. Scaling

using user defined functions, and the output is displayed. If user enters invalid option, then an error message is displayed.

# 2. Problem Statement

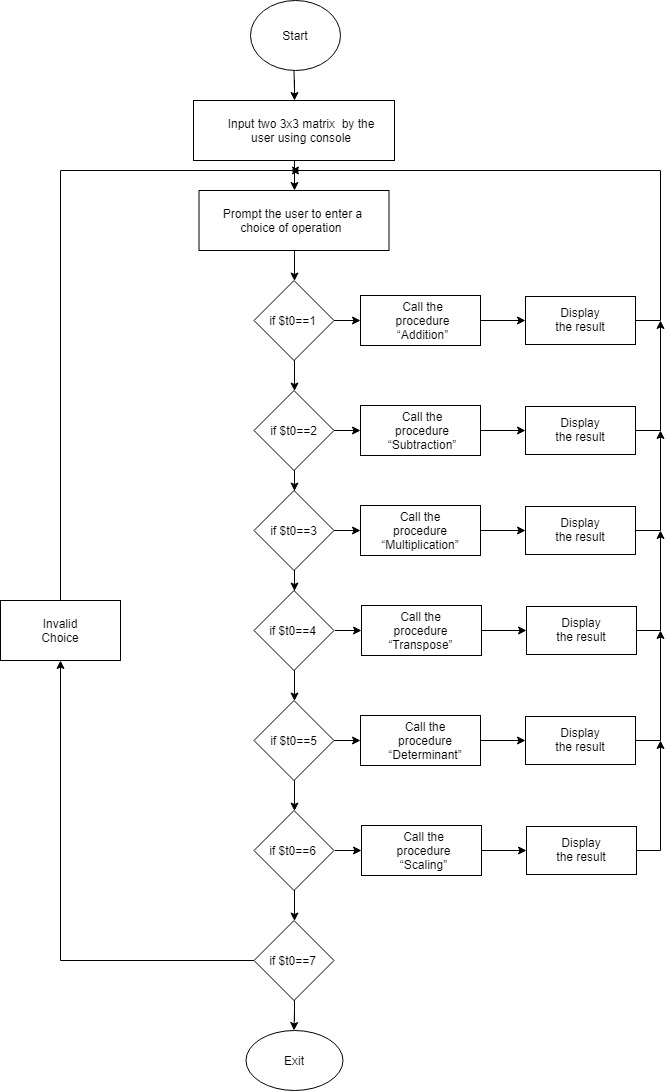
To develop matrix operation calculator using MIPS. It consists of basic operation like addition, subtraction, scaling and transpose and multiplication. Addition or subtraction is accomplished by adding or subtracting corresponding elements. Matrix addition is used to add up the respective elements present in two different matrices and store it into another matrix to show the addition property of the matrix. Similarly, Matrix subtraction is to subtract the respective elements present in two different matrices and store it in another matrix to show the subtraction property of the matrix. Scaling is the transformation, when applied to an object multiplies each of the local coordinates. Here Scaling is used to scale up/down the matrix with a constant using scalar multiplication. The determinant of a matrix is a scalar property of that matrix, which can be thought of physically as the volume enclosed by the row vectors of the matrix. Only square matrices have determinants. Transpose of matrix is that in which all the rows of a given matrix is transformed into columns and vice-versa. Multiplication of matrices is also done using MIPS code by rows of first matrix with the columns of second matrix.

# 3. Brief Description

As already mentioned in the introduction, matrix manipulator system will be required to carry out various operations and provide meaning to these combined structures. This section will outline the various features, implementations and user interactions provided by the system. During this project the basic matrix operations such as matrix, addition, subtraction, scaling, multiplication and transpose. In Matrix Addition and subtraction, we assume the two matrices given in the data segment. All we have to do is to start with the base address, add the respective elements of the two matrices, increment the address by four and do the same till the length is reached. While approaching for the Scaling of matrices there are many prospects to show scaling. We chose the one with scalar multiplication of the matrix with a constant. To find determinant we use the formula A[0][0] x ((A[1][1]xA[2][2])-(A[2][1]xA[1][2]))-A[0][1]x((A[1][0]xA[2][2])-(A[2][0]xA[1][2]))+A[0][2]x((A[1][0]xA[2][1])-(A[2][0]xA[1][1])) as we have considered a 3x3 squared matrix. The next operation assigned to us is transpose in which we just have to reverse the rows with the columns and the columns with the rows simultaneously. The last operation is multiplication of the matrix. If the user enters an invalid option then an error message is displayed.

# 4. Steps of the Algorithms/Flow Diagram

1. Input the two matrixes each of size 3x3 from the user in the console using various syscall services.
2. Prompt the user to enter the choice of the operation and store it in $t0 register.
   1. If the content of $t0 register=1 then perform addition of the two matrices (using addition function).
   2. If the content of $t0 register=2 then perform subtraction of the two matrices (using subtraction function).
   3. If the content of $t0 register=3 then perform multiplication of two matrices (using multiplicationfunction).
   4. If the content of $t0 register=4 then perform transpose of first matrix (using transpose function).
   5. If the content of $t0 register= 5 then finddeterminant of first matrix (using determinant function).
   6. If the content of $t0 register= 6 then performscaling for first matrix by taking a scaling element(using scalingfunction).
   7. If the content of $t0 register= 7 then exit.
   8. For any other choice, display the error message.



# 5. Source Code using MIPS

.data

dimension: .asciiz "Enter dimension of the matrix: "

enter\_first: .asciiz "Enter first matrix:\n"

enter\_second: .asciiz "Enter second matrix:\n"

show\_first: .asciiz "First matrix:\n"

show\_second: .asciiz "Second matrix:\n"

menu: .asciiz "\n1.Addition\n2.Subtraction\n3.Multiplication\n\nThe Following operations are for the first matrix\n4.Transpose\n5.Determinant\n6.Scaling\n7.Exit"

choice: .asciiz "\nEnter your choice :"

num\_scale:.asciiz "\nEnter a number for scaling:"

wrongchoice: .asciiz "\nInvalid Choice!"

exiting: .asciiz "\n\tExiting."

show\_result: .asciiz "Result:\n"

space: .asciiz " "

newline: .asciiz "\n"

.text

.globl main

main:

# Prompt and take dimension input

li $v0, 4

la $a0, dimension

syscall

li $v0, 5

syscall

addu $s0, $zero, $v0 # $s0 == N

# Size of the 2d array stored in $a0 for next three steps

mul $a0, $s0, $s0

mul $a0, $a0, 4

# Declaring matrix A in $s1

li $v0, 9

syscall

addu $s1, $zero, $v0

# Declaring matrix B in $s2

li $v0, 9

syscall

addu $s2, $zero, $v0

# Declaring resultant matrix in $s3

li $v0, 9

syscall

addu $s3, $zero, $v0

# $s4 = N^2

mul $s4, $s0, $s0

# Take input in matrix A

li $v0, 4

la $a0, enter\_first

syscall

xor $t1, $t1, $t1 # loop variable

move $t2, $s1 # pointer

loop1:

slt $t0, $t1, $s4

beq $t0, $zero, exit1

li $v0, 5

syscall

sw $v0, 0($t2)

addiu $t1, $t1, 1

addiu $t2, $t2, 4

j loop1

exit1:

# Take input in matrix B

li $v0, 4

la $a0, enter\_second

syscall

xor $t1, $t1, $t1 # loop variable

move $t2, $s2 # pointer

loop2:

slt $t0, $t1, $s4

beq $t0, $zero, exit2

li $v0, 5

syscall

sw $v0, 0($t2)

addiu $t1, $t1, 1

addiu $t2, $t2, 4

j loop2

exit2:

# Print matrix A

li $v0, 4

la $a0, show\_first

syscall

move $a1, $s1

jalprintMatrix

# Print matrix B

li $v0, 4

la $a0, show\_second

syscall

move $a1, $s2

jalprintMatrix

li $v0,4

la $a0, menu

syscall

INPUT:

li $v0,4

la $a0, choice

syscall

li $v0,5

syscall

move $t0, $v0

li $t1, 1

li $t2, 2

li $t3, 3

li $t4, 4

li $t5, 5

li $t6, 6

li $t7, 7

move $a0,$s0

move $a1,$s1

move $a2,$s2

move $a3,$s3

blez $t0, Label0

bgt $t0, $t7, Label0

beq $t0, $t1, Label1

beq $t0, $t2, Label2

beq $t0, $t3, Label3

beq $t0, $t4, Label4

beq $t0, $t5, Label5

beq $t0, $t6, Label6

beq $t0, $t7, Label7

Label0:

li $v0,4

la $a0,wrongchoice

syscall

j INPUT

Label1:

jal addition

j INPUT

Label2:

jal subtraction

j INPUT

Label3:

jal multiplication

j INPUT

Label4:

jal transpose

j INPUT

Label5:

jal determinant

j INPUT

Label6:

jal scaling

j INPUT

Label7:

li $v0,4

la $a0,exiting

syscall

li $v0,10

syscall

.end main

.globl addition

.ent addition

addition:

move $s0,$a0

move $s1,$a1

move $s2,$a2

xor $t1, $t1, $t1

L1:

slt $t0, $t1, $s0

beq $t0, $zero, endL1

xor $t2, $t2, $t2

L2:

slt $t0, $t2, $s0

beq $t0, $zero, endL2

mul $t4, $t1, $s0

addu $t4, $t4, $t2

sll $t4, $t4, 2

addu $t4, $t4, $s1

mul $t5, $t1, $s0

addu $t5, $t5, $t2

sll $t5, $t5, 2

addu $t5, $t5, $s2

lw $t6, 0($t4)

lw $t7, 0($t5)

add $t8, $t6, $t7

li $v0,1

move $a0, $t8

syscall

li $v0,4

la $a0,space

syscall

addiu $t2, $t2, 1

j L2

endL2:

addiu $t1, $t1, 1

li $v0,4

la $a0, newline

syscall

j L1

endL1:

jr $ra

.end addition

.globl subtraction

.ent subtraction

subtraction:

move $s0,$a0

move $s1,$a1

move $s2,$a2

xor $t1, $t1, $t1

L3:

slt $t0, $t1, $s0

beq $t0, $zero, endL3

xor $t2, $t2, $t2

L4:

slt $t0, $t2, $s0

beq $t0, $zero, endL4

mul $t4, $t1, $s0

addu $t4, $t4, $t2

sll $t4, $t4, 2

addu $t4, $t4, $s1

mul $t5, $t1, $s0

addu $t5, $t5, $t2

sll $t5, $t5, 2

addu $t5, $t5, $s2

lw $t6, 0($t4)

lw $t7, 0($t5)

sub $t8, $t6, $t7

li $v0,1

move $a0,$t8

syscall

li $v0,4

la $a0,space

syscall

addiu $t2, $t2, 1

j L4

endL4:

li $v0,4

la $a0,newline

syscall

addiu $t1, $t1, 1

j L3

endL3:

jr $ra

.end subtraction

.globl multiplication

.ent multiplication

multiplication:

move $s0,$a0

move $s1,$a1

move $s2,$a2

move $s3,$a3

xor $t1, $t1, $t1 # loop 1 variable

L5:

slt $t0, $t1, $s0

beq $t0, $zero, endL5

xor $t2, $t2, $t2 # loop 2 variable

L6:

slt $t0, $t2, $s0

beq $t0, $zero, endL6

mul $t4, $t1, $s0 # address of resultant[i][j]

addu $t4, $t4, $t2

sll $t4, $t4, 2

addu $t4, $t4, $s3

xor $t3, $t3, $t3 # loop 3 variable

L7:

slt $t0, $t3, $s0

beq $t0, $zero, endL7

mul $t5, $t1, $s0 # address of matA[i][k]

addu $t5, $t5, $t3

sll $t5, $t5, 2

addu $t5, $t5, $s1

mul $t6, $t3, $s0 # address of matB[k][j]

addu $t6, $t6, $t2

sll $t6, $t6, 2

addu $t6, $t6, $s2

lw $t7, 0($t5) # loading matA[i][k]

lw $t8, 0($t6) # loading matB[k][j]

mul $t9, $t7, $t8 # matA[i][k] \* matB[k][j]

lw $t8, 0($t4)

addu $t9, $t9, $t8 # resultant += matA[i][k] \* matB[k][j]

sw $t9,0($t4)

addiu $t3, $t3, 1

j L7

endL7:

li $v0,1

move $a0,$t9

syscall

li $v0,4

la $a0,space

syscall

addiu $t2, $t2, 1

j L6

endL6:

li $v0,4

la $a0,newline

syscall

addiu $t1, $t1, 1

j L5

endL5:

jr $ra

.end multiplication

.globl transpose

.ent transpose

transpose:

move $s0,$a0

move $s1,$a1

xor $t1, $t1, $t1

L8:

slt $t0, $t1, $s0

beq $t0, $zero, endL8

xor $t2, $t2, $t2

L9:

slt $t0, $t2, $s0

beq $t0, $zero, endL9

mul $t4, $t2, $s0

addu $t4, $t4, $t1

sll $t4, $t4, 2

addu $t4, $t4, $s1

lw $t5, 0($t4)

li $v0,1

move $a0,$t5

syscall

li $v0,4

la $a0,space

syscall

addiu $t2, $t2, 1

j L9

endL9:

li $v0,4

la $a0,newline

syscall

addiu $t1, $t1, 1

j L8

endL8:

jr $ra

.end transpose

.globl determinant

.ent determinant

determinant:

move $s0,$a0

move $s1,$a1

li $t0,0

li $t1,1

li $t2,2

mul $t3,$t0,$s0

addu $t3,$t3,$t0

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t4,0($t3) # A[0][0]

mul $t3,$t1,$s0

addu $t3,$t3,$t1

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t5,0($t3) # A[1][1]

mul $t3,$t2,$s0

addu $t3,$t3,$t2

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t6,0($t3) # A[2][2]

mul $t5,$t5,$t6 #A[1][1]\*A[2][2]

mul $t3,$t2,$s0

addu $t3,$t3,$t1

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t6,0($t3) # A[2][1]

mul $t3,$t1,$s0

addu $t3,$t3,$t2

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t7,0($t3) # A[1][2]

mul $t6,$t6,$t7 #A[2][1]\*A[1][2]

sub $t5,$t5,$t6

mul $s7,$t4,$t5

mul $t3,$t0,$s0

addu $t3,$t3,$t1

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t4,0($t3) # A[0][1]

mul $t3,$t1,$s0

addu $t3,$t3,$t0

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t5,0($t3) # A[1][0]

mul $t3,$t2,$s0

addu $t3,$t3,$t2

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t6,0($t3) # A[2][2]

mul $t5,$t5,$t6 #A[1][0]\*A[2][2]

mul $t3,$t2,$s0

addu $t3,$t3,$t0

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t6,0($t3) # A[2][0]

mul $t3,$t1,$s0

addu $t3,$t3,$t2

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t7,0($t3) # A[1][2]

mul $t6,$t6,$t7 #A[2][0]\*A[1][2]

sub $t5,$t5,$t6

mul $s6,$t4,$t5

sub $s7,$s7,$s6

mul $t3,$t0,$s0

addu $t3,$t3,$t2

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t4,0($t3) # A[0][2]

mul $t3,$t1,$s0

addu $t3,$t3,$t0

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t5,0($t3) # A[1][0]

mul $t3,$t2,$s0

addu $t3,$t3,$t1

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t6,0($t3) # A[2][1]

mul $t5,$t5,$t6 #A[1][0]\*A[2][1]

mul $t3,$t2,$s0

addu $t3,$t3,$t0

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t6,0($t3) # A[2][0]

mul $t3,$t1,$s0

addu $t3,$t3,$t1

sll $t3,$t3,2

addu $t3,$t3,$s1

lw $t7,0($t3) # A[1][1]

mul $t6,$t6,$t7 #A[2][0]\*A[1][1]

sub $t5,$t5,$t6

mul $s6,$t4,$t5

add $s7,$s7,$s6

li $v0, 4

la $a0, show\_result

syscall

li $v0,1

move $a0, $s7

syscall

jr $ra

.end determinant

.globl scaling

.ent scaling

scaling:

move $s0,$a0

move $s1,$a1

li $v0,4

la $a0,num\_scale

syscall

li $v0,5

syscall

move $s7,$v0

xor $t1, $t1, $t1

L11:

slt $t0, $t1, $s0

beq $t0, $zero, endL11

xor $t2, $t2, $t2

L12:

slt $t0, $t2, $s0

beq $t0, $zero, endL12

mul $t4, $t1, $s0

addu $t4, $t4, $t2

sll $t4, $t4, 2

addu $t4, $t4, $s1

lw $t5,0($t4)

mul $t5,$t5,$s7

li $v0,1

move $a0,$t5

syscall

li $v0,4

la $a0,space

syscall

addiu $t2, $t2, 1

j L12

endL12:

li $v0,4

la $a0,newline

syscall

addiu $t1, $t1, 1

j L11

endL11:

jr $ra

.end scaling

.globlprintMatrix

.entprintMatrix

printMatrix:

xor $t1, $t1, $t1 # loop 1 variable

print1:

slt $t0, $t1, $s0

beq $t0, $zero, end\_print1

addiu $t1, $t1, 1

xor $t2, $t2, $t2 # loop 2 variable

print2:

slt $t0, $t2, $s0

beq $t0, $zero, end\_print2

addiu $t2, $t2, 1

li $v0, 1

lw $a0, 0($a1)

syscall

addiu $a1, $a1, 4 # increment pointer

li $v0, 4

la $a0, space

syscall

j print2

end\_print2:

li $v0, 4

la $a0, newline

syscall

j print1

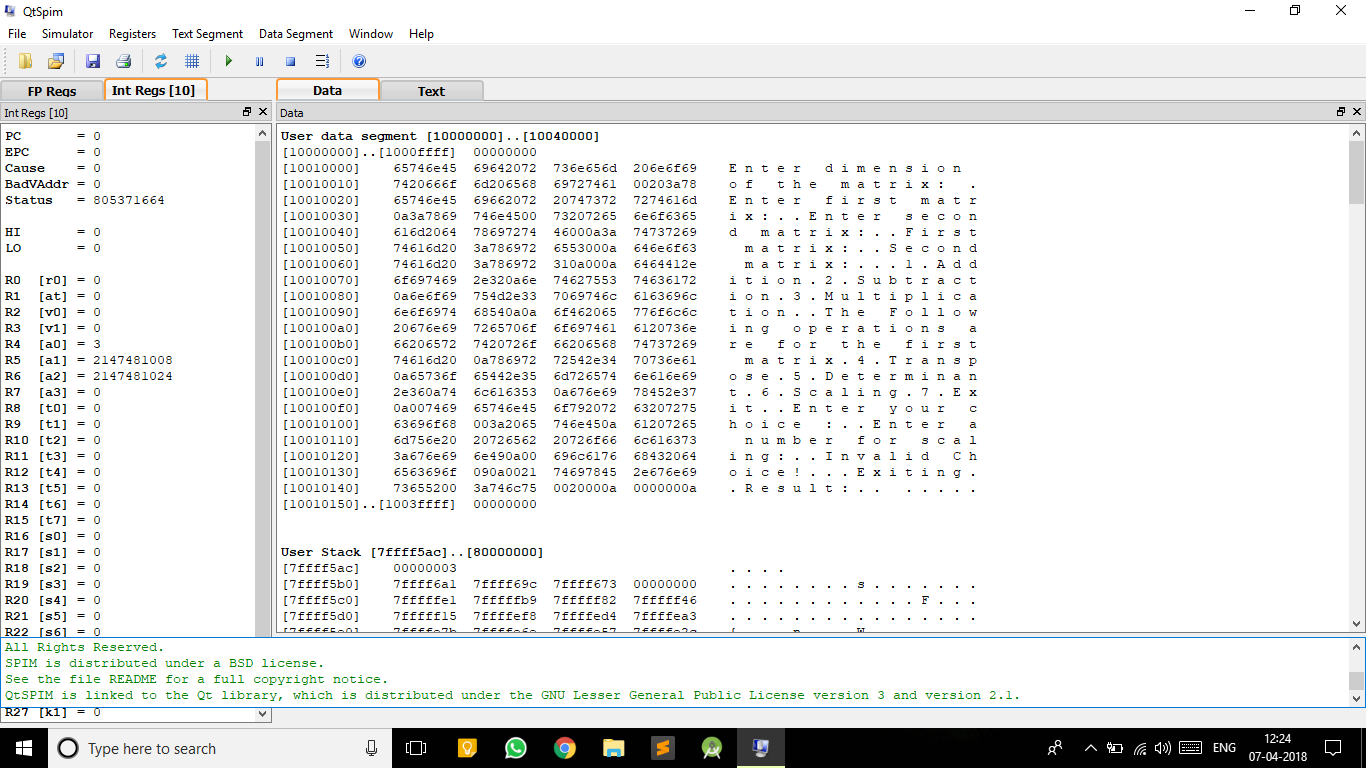
end\_print1:

jr $ra

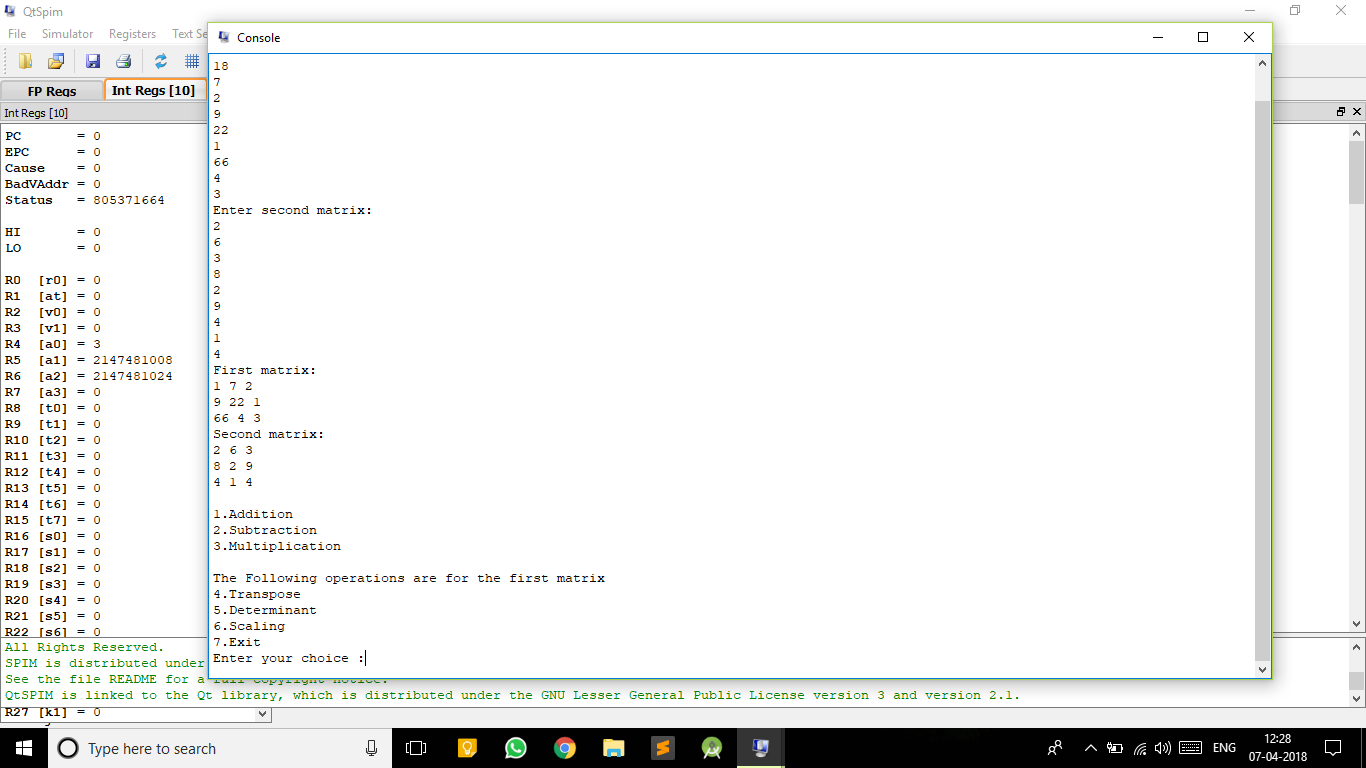
.endprintMatrix

**6. Testing**

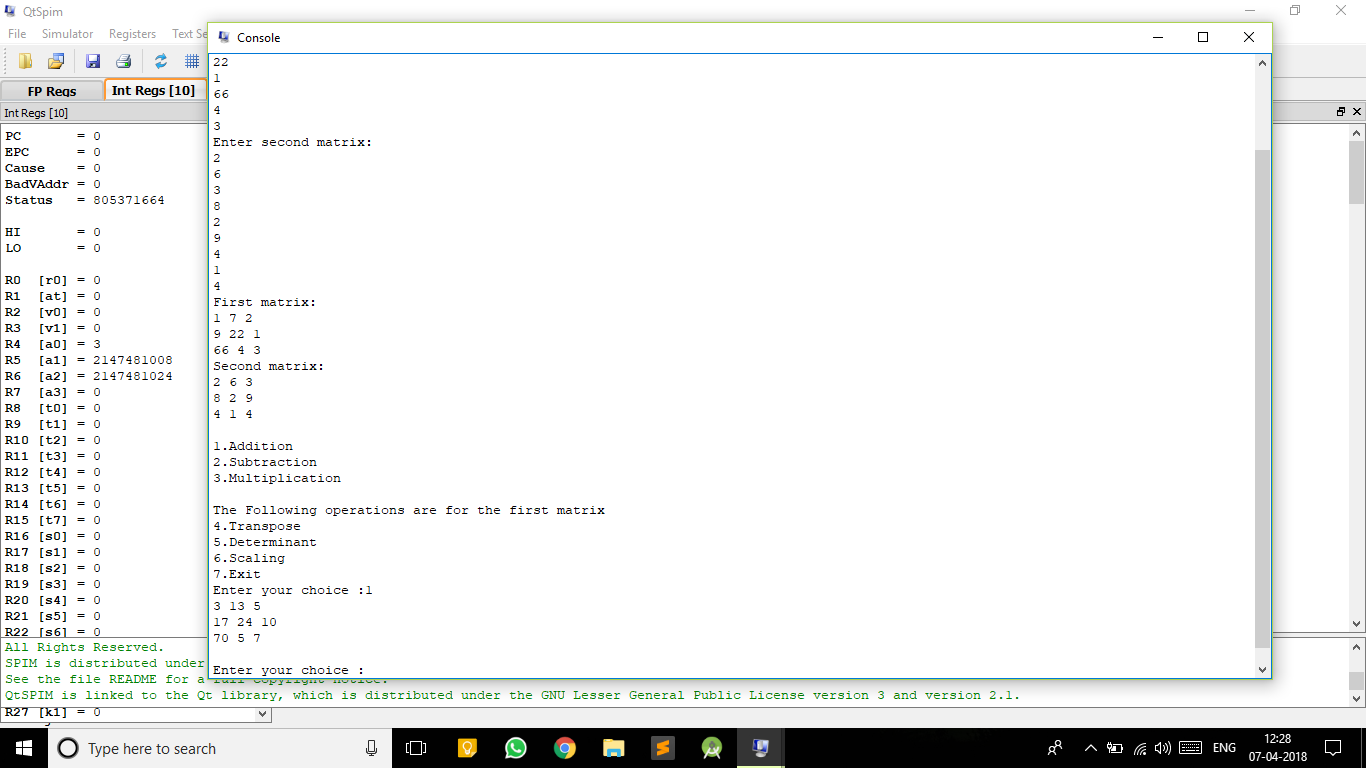
1. QtSPIM is reinitialized and the program is loaded.



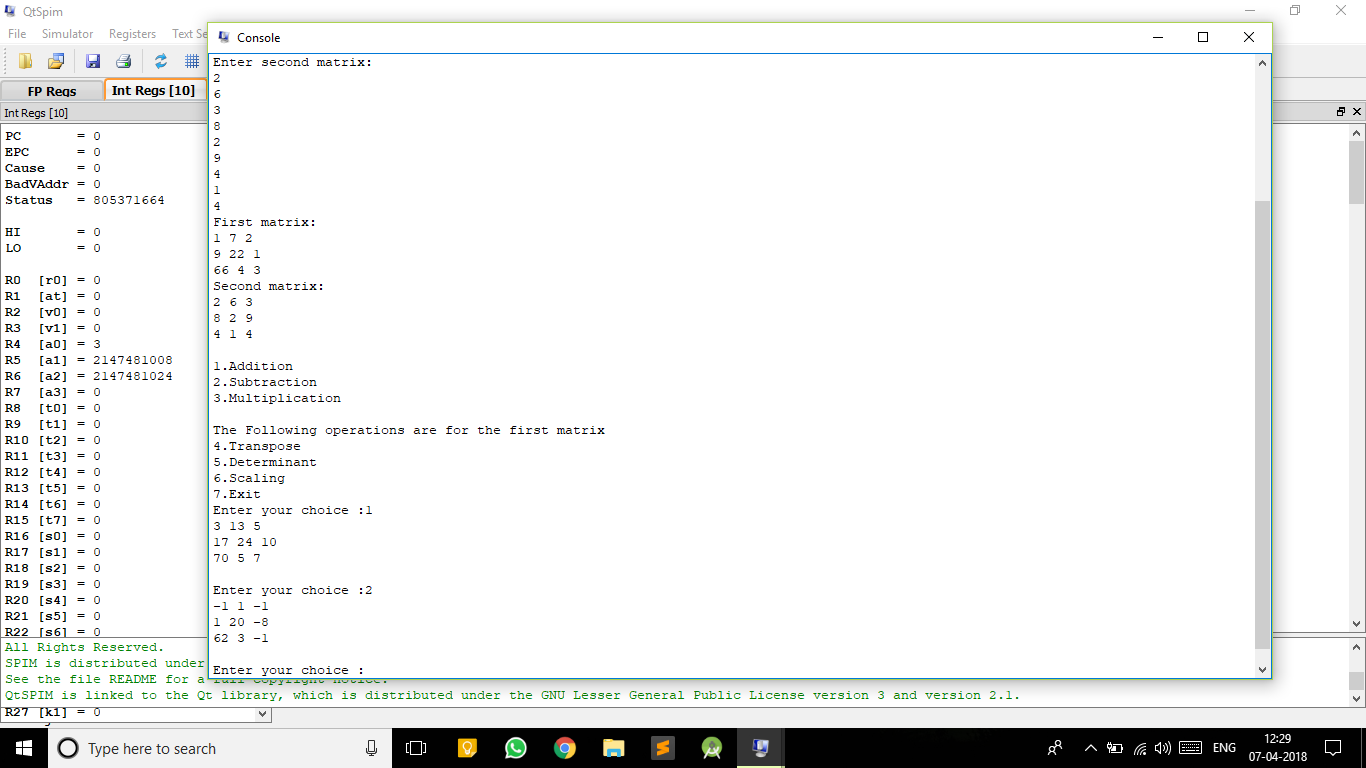
2. Two matrices are taken as input.



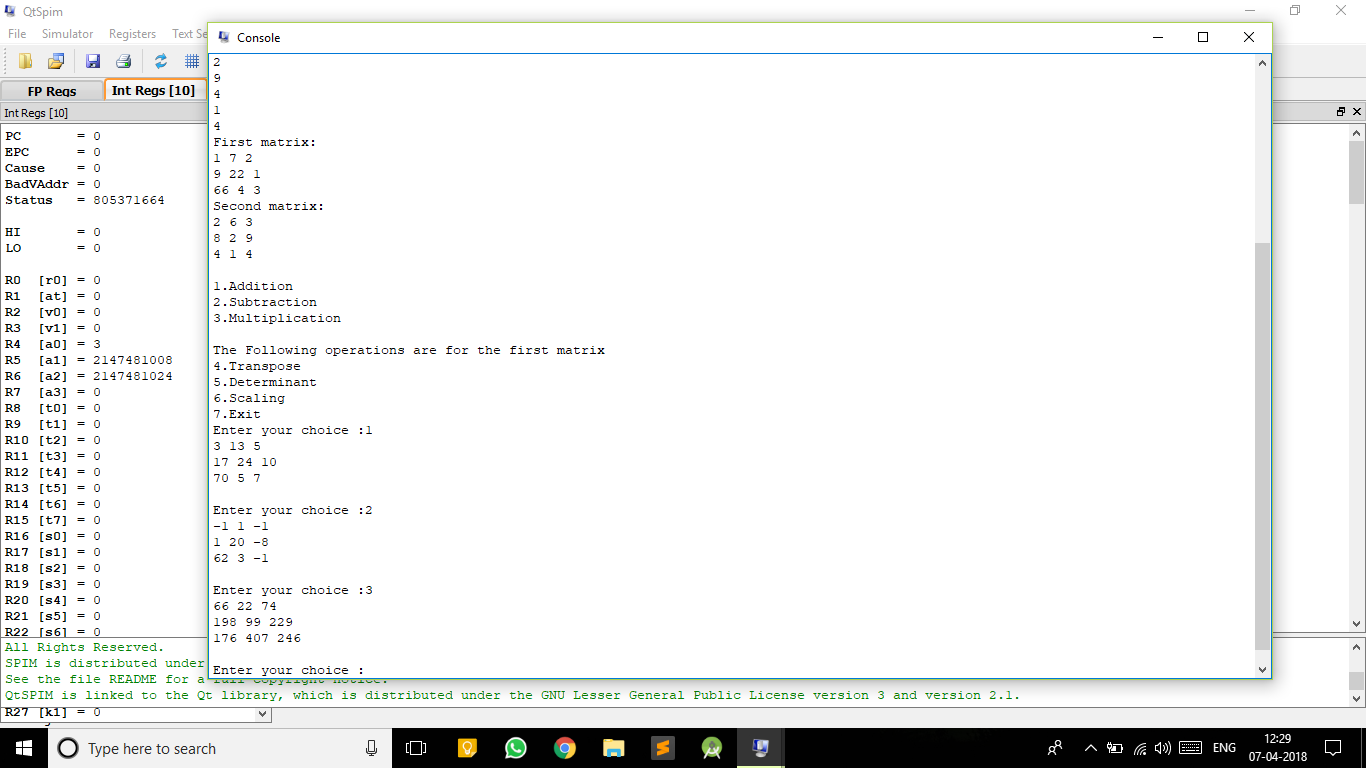
3. Addition is performed.



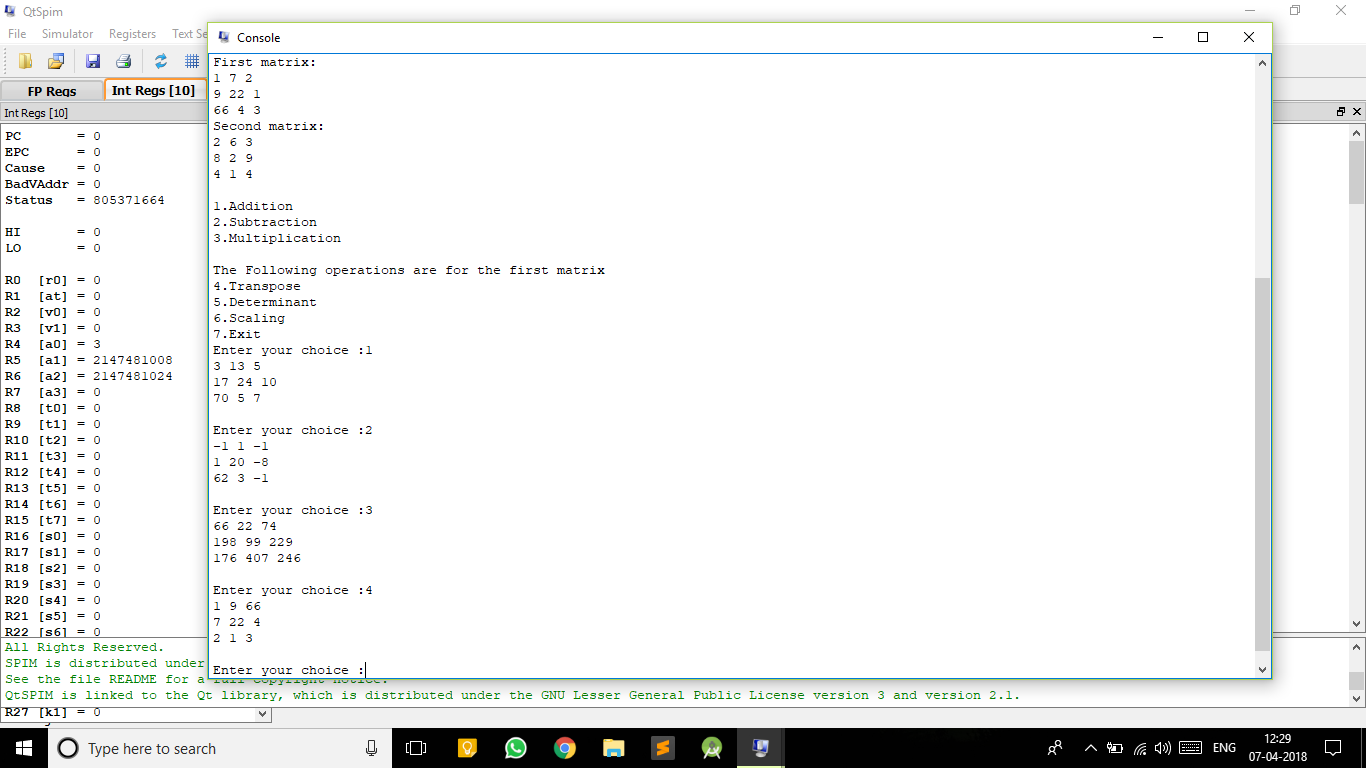
4. Subtraction is performed.



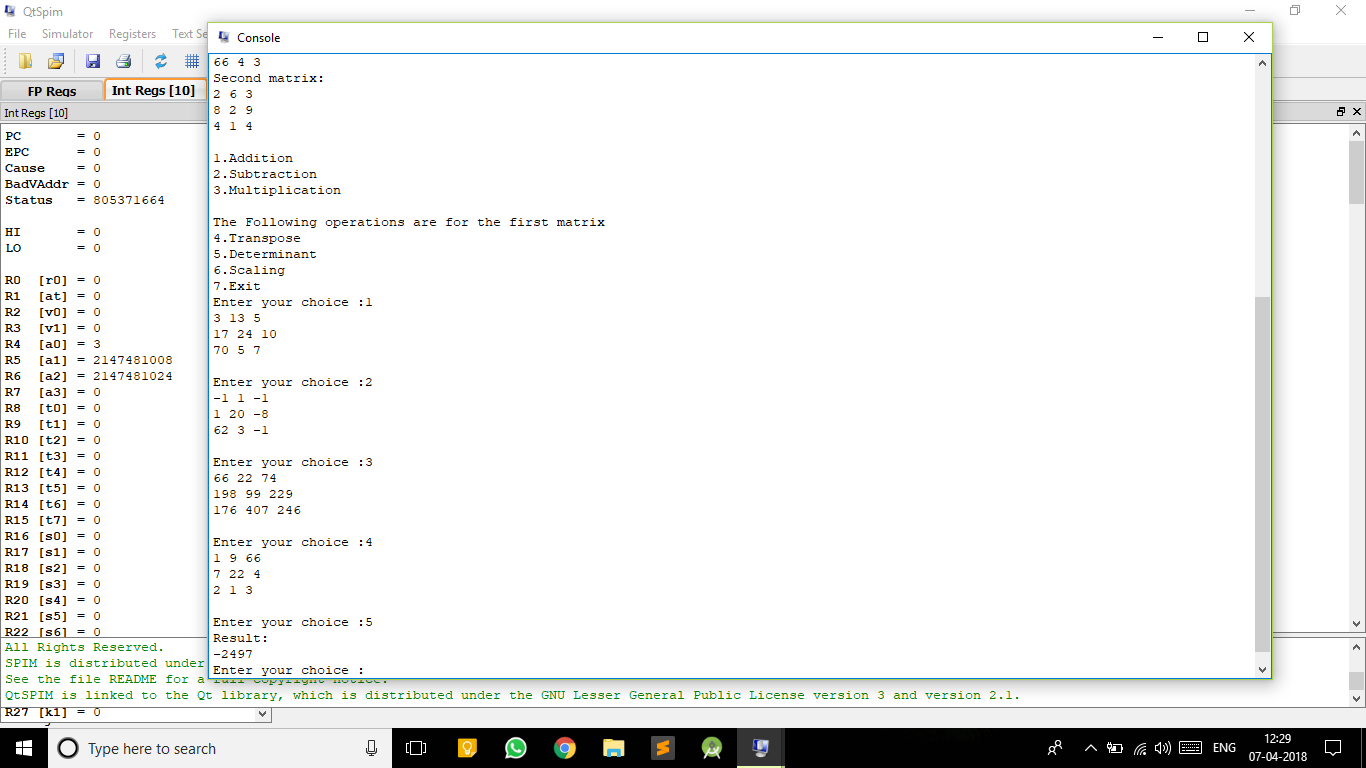
5. Multiplication is performed.



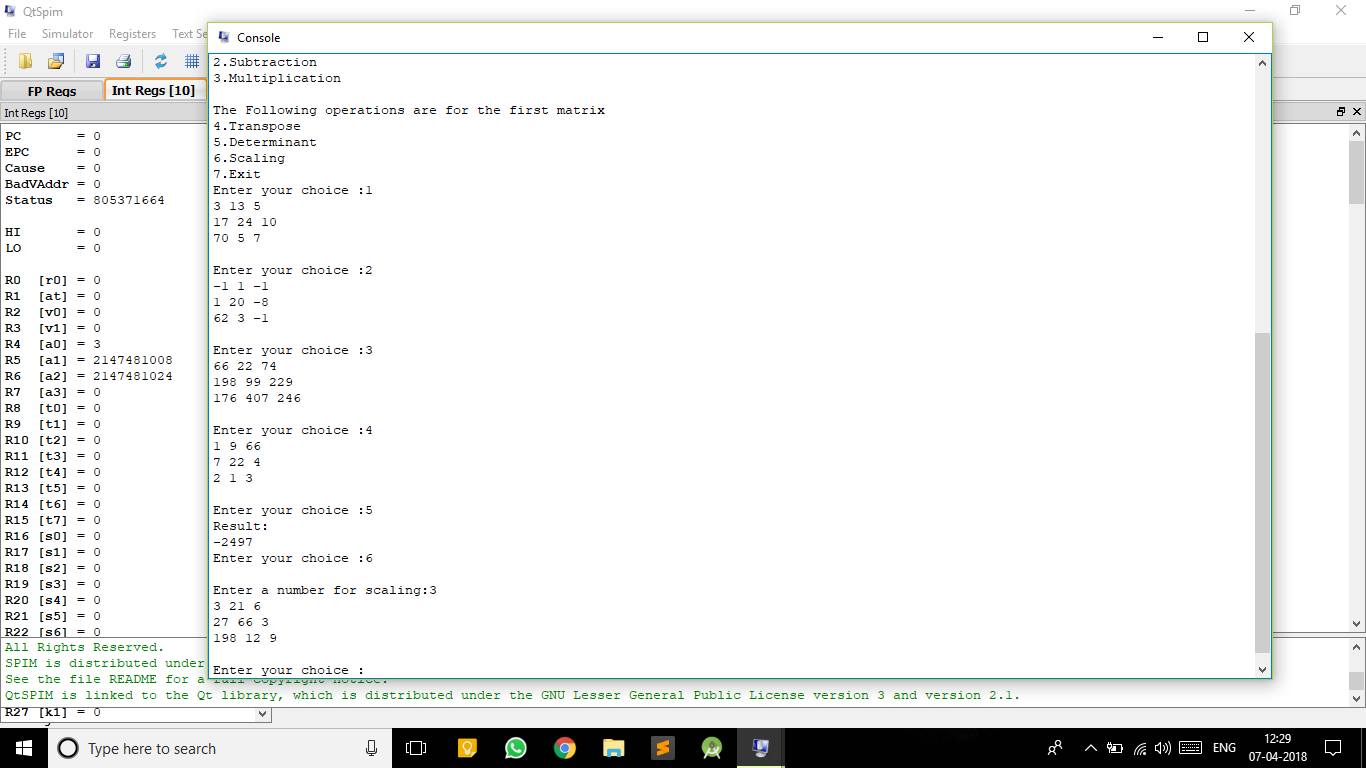
6. The transpose of the first matrix is found out.



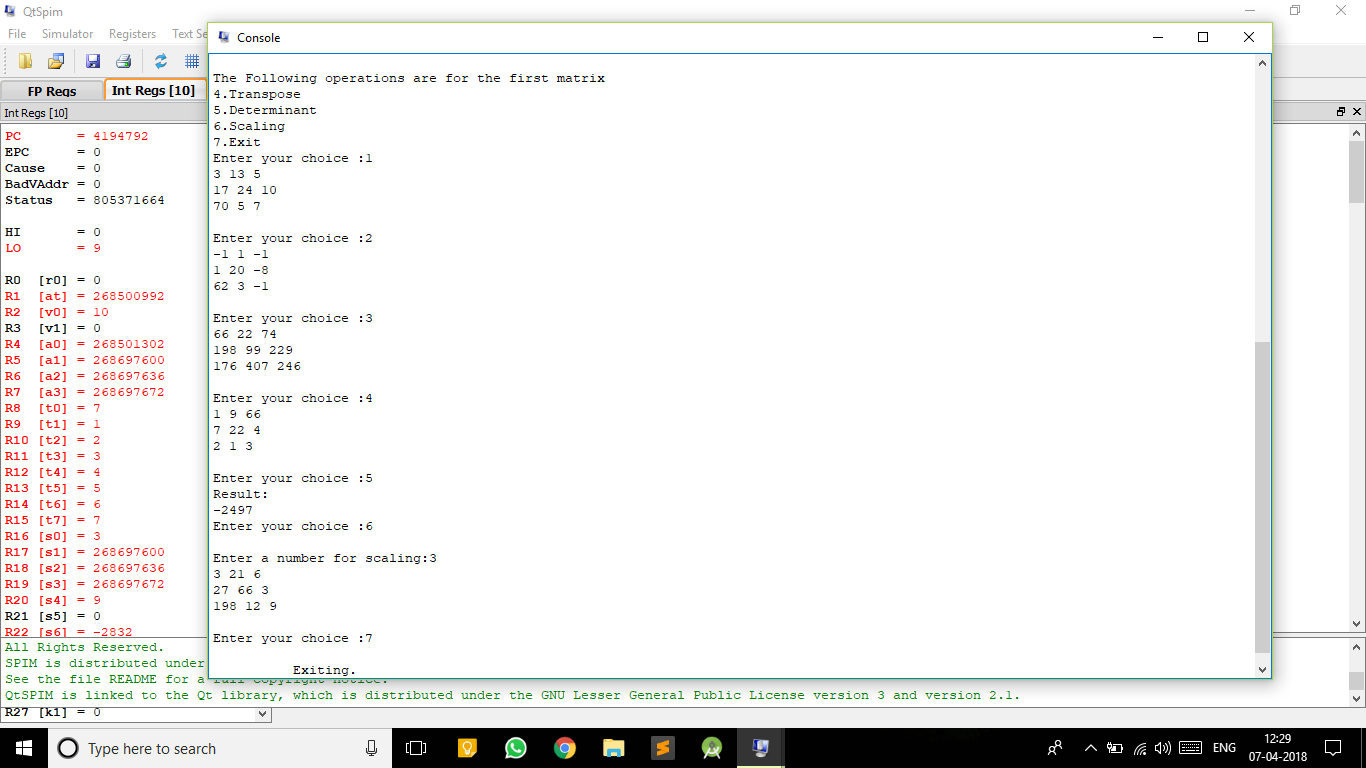
7. The determinant is found of the first matrix.



8. The first matrix is scaled by taking a number to for scaling.



9. The program is terminated.



# 7. Conclusion

In order to simplify matrix manipulation, we designed a MIPS code for Matrix Manipulator to test and implement different arithmetic operations like Matrix Addition, Matrix Subtraction, matrix scaling and matrix transpose and matrix multiplication. This project comprises of all the arithmetic operations of the matrix assigned to us.

The limitations that came across while designing the project was

1. The matrices must be a square matrix.

2. For scaling and transposing, and determinant two matrixes are needed to be taken as input, but the operations are performed on first matrix.

# 8. References

* Computer Organization and Design
  + By David A Patterson and John L Hennessy.
* New Jersey Institute of Technology eLab
  + http://ecelabs.njit.edu/ece459/lab1.php
* University of Pittsburgh e-Library
  + http://people.cs.pitt.edu/~xujie/cs447/AccessingArray.htm
* Draw.io
  + Online diagram software for making flowcharts.