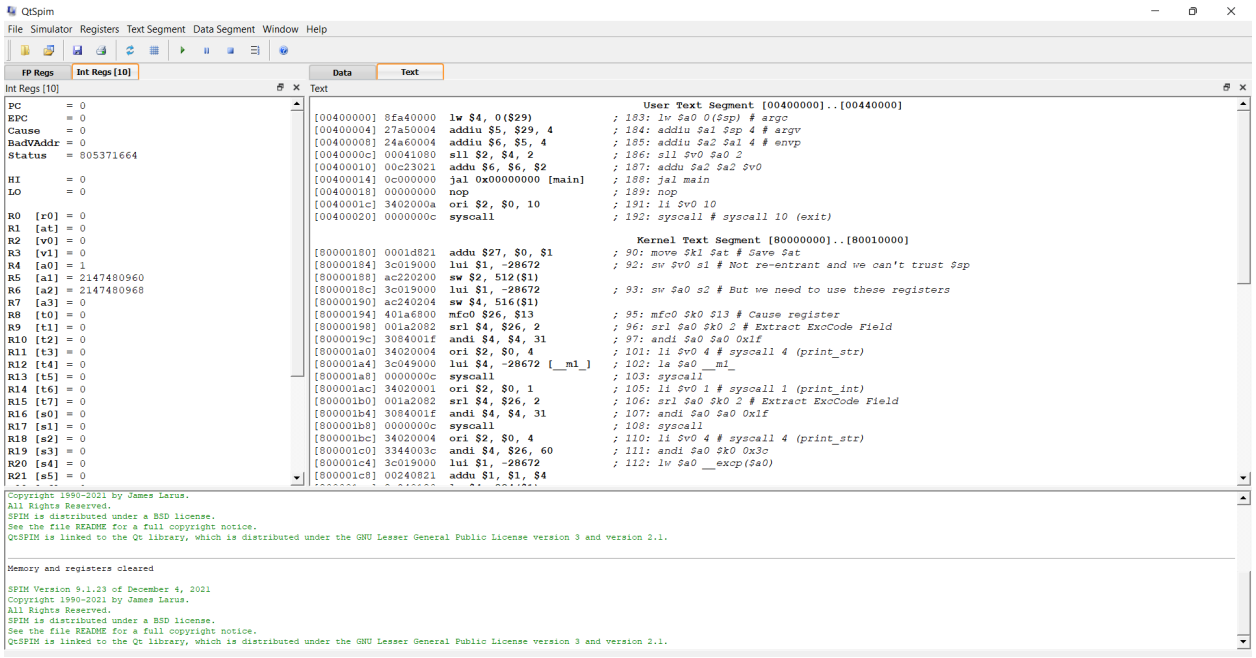
Lab3
CS211
Exceptions in MIPS

Name-Pranav Tambe
Roll No-2106339



Interface before loading any file

Q.1) Reverse a string entered by the user. (Hint: Ask the user to enter a string. After reading the string in a buffer, copy it in reversed order to a second buffer. Write out the reversed string.)

```
.data
```

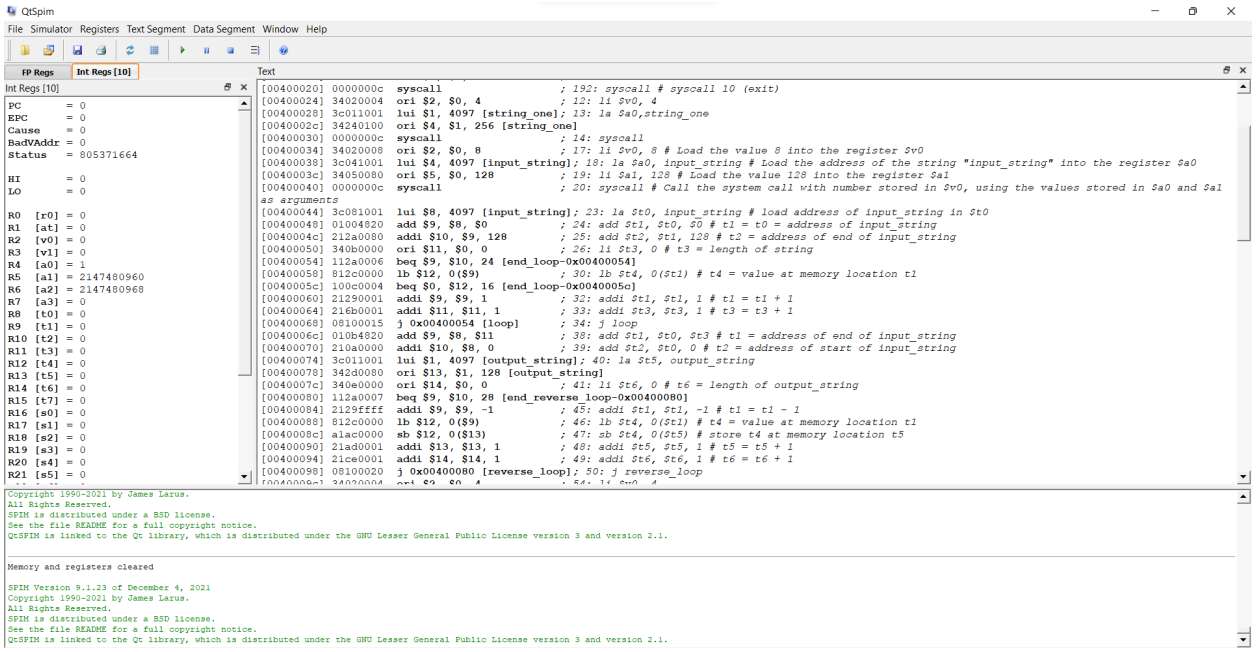
```
    input_string: .space 128 # allocate space for the input string
    output_string: .space 128 # allocate space for the reversed string
    string_one: .asciiz "Enter the string :"
    string_two: .asciiz "Reverse  string :"

.text # tells assembler to switch to the text segment or succeeding lines
contains instructions

main:  # start of code section

    # for printing   string
    li $v0, 4
    la $a0,string_one
    syscall

    # read the string into the input buffer
    li $v0, 8 # Load the value 8 into the register $v0
    la $a0, input_string  # Load the address of the string "input_string"
into the register $a0
    li $a1, 128 # Load the value 128 into the register $a1
    syscall # Call the system call with number stored in $v0, using the
values stored in $a0 and $a1 as arguments

    # find the length of the string
    la $t0, input_string # load address of input_string in $t0
    add $t1, $t0, $0 # t1 = t0 = address of input_string
    add $t2, $t1, 128 # t2 = address of end of input_string
    li $t3, 0 # t3 = length of string

    loop:
        beq $t1, $t2, end_loop # if t1 == t2, end loop
        lb $t4, 0($t1) # t4 = value at memory location t1
        beq $t4, 0, end_loop # if t4 == 0, end loop
        addi $t1, $t1, 1 # t1 = t1 + 1
        addi $t3, $t3, 1 # t3 = t3 + 1
        j loop

    end_loop:
        # copy the string in reversed order to the output buffer
        add $t1, $t0, $t3 # t1 = address of end of input_string
```

```
    add $t2, $t0, 0 # t2 = address of start of input_string
    la $t5, output_string
    li $t6, 0 # t6 = length of output_string

reverse_loop:
    beq $t1, $t2, end_reverse_loop # if t1 == t2, end loop
    addi $t1, $t1, -1 # t1 = t1 - 1
    lb $t4, 0($t1) # t4 = value at memory location t1
    sb $t4, 0($t5) # store t4 at memory location t5
    addi $t5, $t5, 1 # t5 = t5 + 1
    addi $t6, $t6, 1 # t6 = t6 + 1
    j reverse_loop

end_reverse_loop:

    li $v0, 4
    la $a0,string_two
    syscall
    # print the reversed string
    li $v0, 4
    la $a0, output_string
    syscall

# exit program
li $v0, 10
syscall
```

Brief overview of the code section

The code starts by allocating space for the input string and the reversed string using the .data directive.
Then, the code switches to the text segment using the .text directive, where the instructions for the code are stored.
The main section of the code starts with a prompt for the user to enter a string, which is then read into the input buffer using a system call.
The code then finds the length of the string and copies the string in reverse order to the output buffer.
Finally, the reversed string is printed and the program exits.

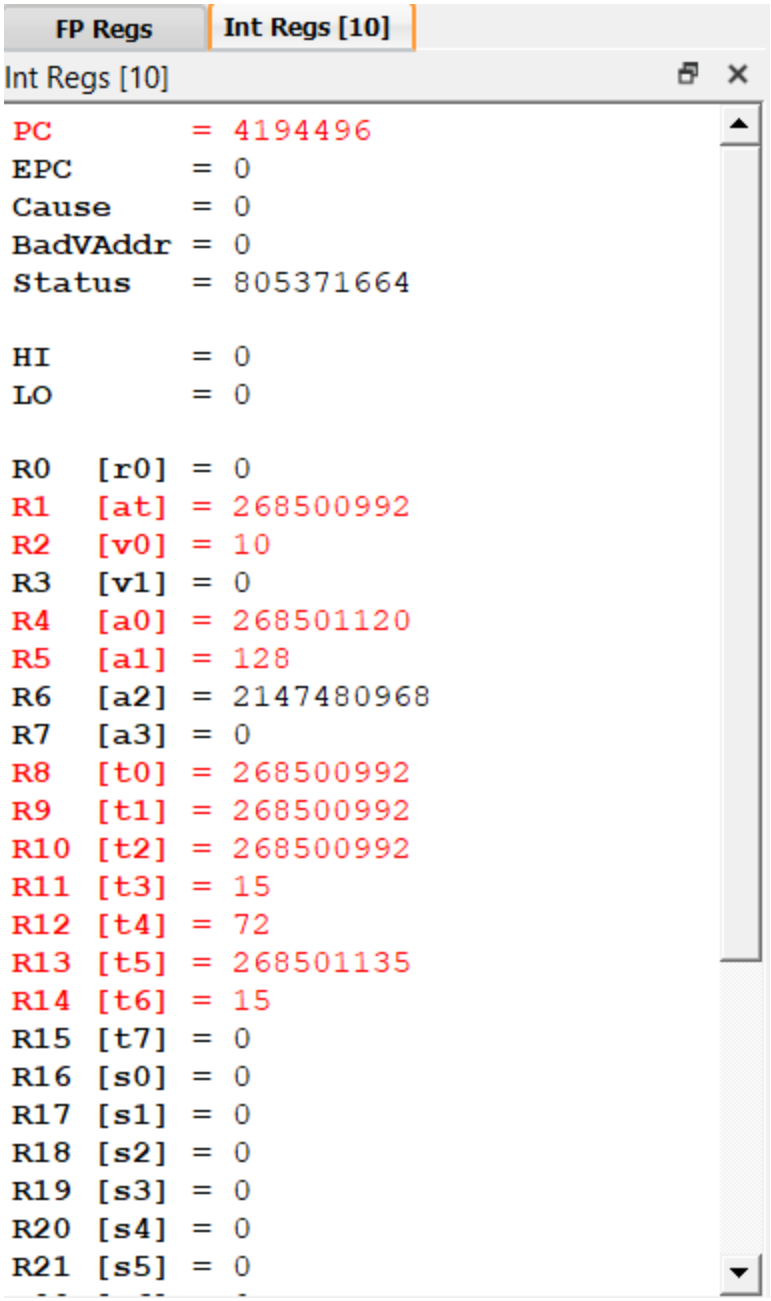
After loading the first file



console

After execution

Registers after execution of the code



Q.2.)Compute the dot product of two vectors each of length 5. Ask the user to enter the value of each
element of the two vectors. Display the dot product.
(Hint: The dot product of two vectors is the sum of products of the corresponding elements. For
example, (1,2,3) dot (4,5,6) is 1*4+2*5+3*6 = 32)

```
.data
    vector_1: .word 0 , 0, 0, 0, 0
    vector_2: .word 0 , 0, 0, 0, 0
    string_1: .asciiz "Enter the value in 1st vector at the index "
    string_2: .asciiz "Enter the value in 2nd vector at the index "
    string_3: .asciiz " :"
    string_4: .asciiz "The dot product is :"

.text # tells assembler to switch to the text segment or succeeding lines
contains instructions

main: # start of code section
        li $t0, 0 # initialize loop counter
        li $t1,0 # index of element in the vector

        # take use input for the first vector
    input_vector_1:
        li $v0, 4 # print string syscall
        la $a0,string_1 # load string address
        syscall # call operating system to perform the operation

        li $v0,1 # print the integer
        move $a0,$t1 # move value in $a0 from $t1
        syscall # call operating system to perform the operation

        li $v0, 4 # print string
        la $a0,string_3# load string address
        syscall # call operating system to perform the operation

        li $v0, 5 # read int syscall
        syscall # call operating system to perform the operation

        addi $t1,$t1,1 # increment the vector index
        sw $v0,vector_1($t0) # store input in vector_1
        addi $t0, $t0, 4 # increment loop counter by 4 bytes
        bne $t0, 20, input_vector_1 # repeat until all values are entered
by branch if not equal command

        li $t0, 0 # initialize loop counter
        li $t1,0  # index of element in the vector
```

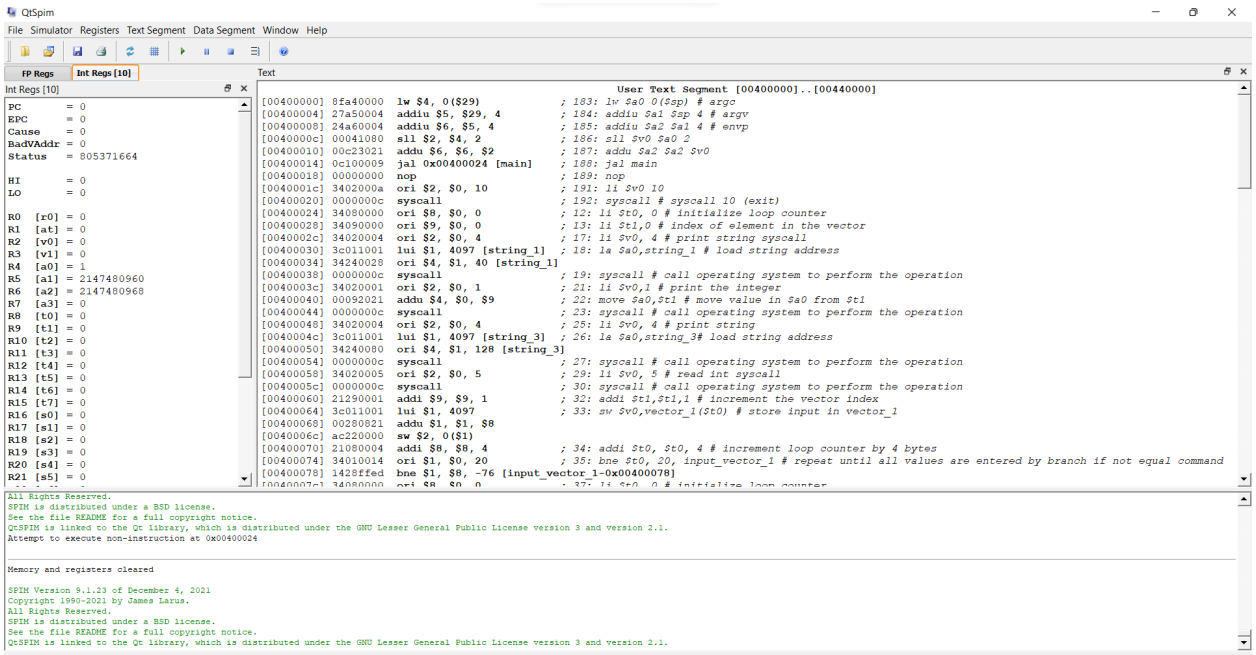
```
# take use input for the second vector
input_vector_2:
    li $v0, 4 # print string syscall
    la $a0,string_2 # load string address
    syscall # call operating system to perform the operation

    li $v0,1 # system call for printing the integer
    move $a0,$t1 # move that integer from $t1 to $a0
    syscall # call operating system to perform the operation

    li $v0, 4 # print string syscall
    la $a0,string_3# load string address
    syscall # call operating system to perform the operation

    li $v0, 5 # read int syscall
    syscall # call operating system to perform the operation

    addi $t1,$t1,1 # increment index of vector
    sw $v0,vector_2($t0) # store input in vec1
    addi $t0, $t0, 4 # increment loop counter
    bne $t0, 20, input_vector_2 # repeat until all values are entered

    li $t0, 0 # initialize loop counter
    li $t1, 0 # initialize accumulator

# block to execute the dot product
dot_product:
    lw $t2, vector_1($t0) # load element from vec1
    lw $t3, vector_2($t0) # load element from vec2
    mul $t4, $t2, $t3 # multiply elements
    add $t1, $t1, $t4 # add result to accumulator
    addi $t0, $t0, 4 # increment loop counter
    bne $t0, 20, dot_product # repeat until all values are multiplied

li $v0, 4 # print string syscall
la $a0,string_4# load string address
syscall # call operating system to perform the operation

li $v0,1 # system call for printing the integer
```

```
    move $a0,$t1 # move that integer from $t1 to $a0
    syscall # call operating system to perform the operation

    li $v0, 10 # System call code for exit
    syscall # Exit program
```

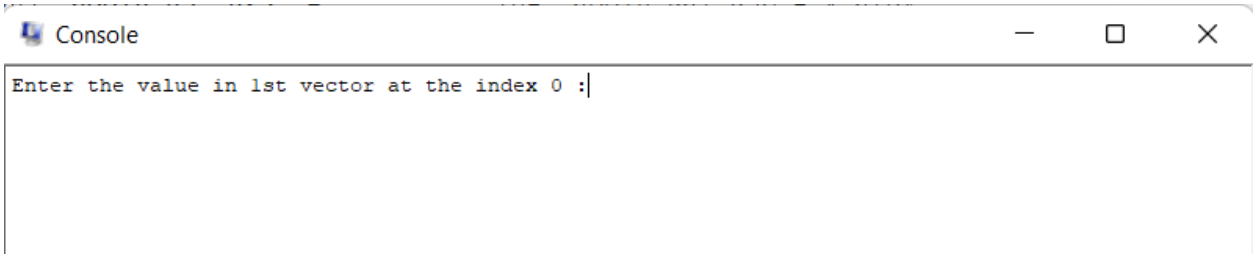
Brief overview of the code section

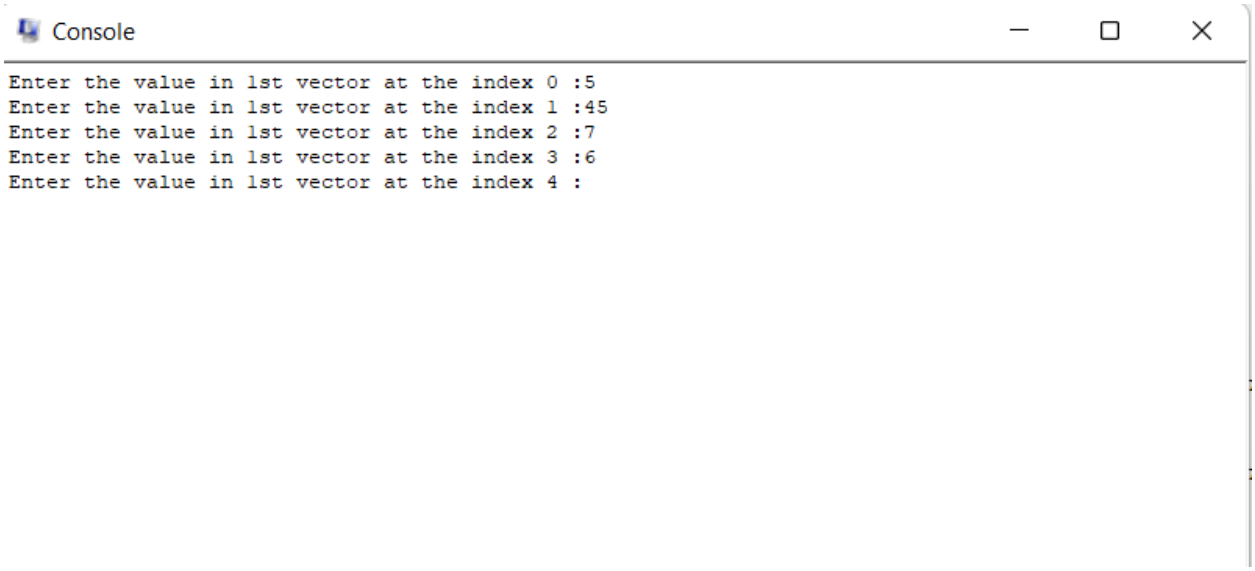
The vectors are stored as arrays in the .data segment of the code, named as vector_1 and vector_2, with 5 elements each initialized to 0.
The .text segment of the code contains the instructions for the program to run.
The code takes input from the user for the values of the two vectors and stores them in the arrays by two loops.
Then it calculates the dot product by multiplying the corresponding elements of the two arrays and adding up the results
. Finally, it prints the result of the dot product.

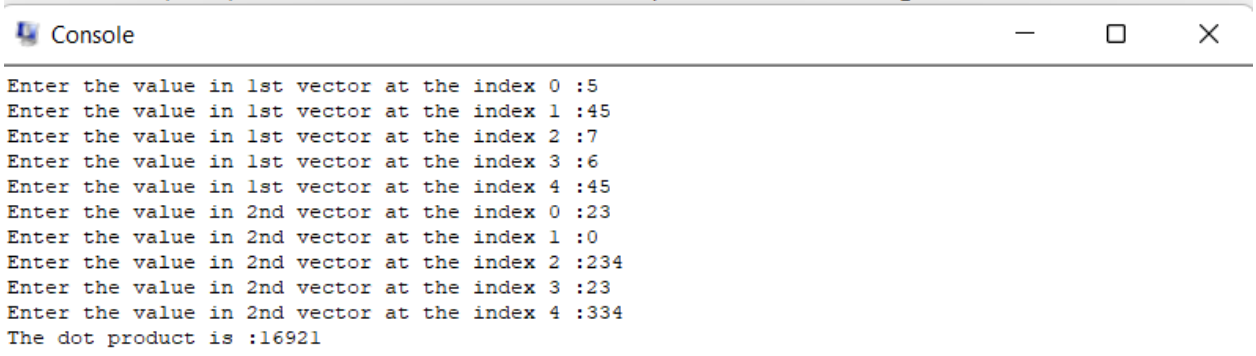
After loading the second file



Console images

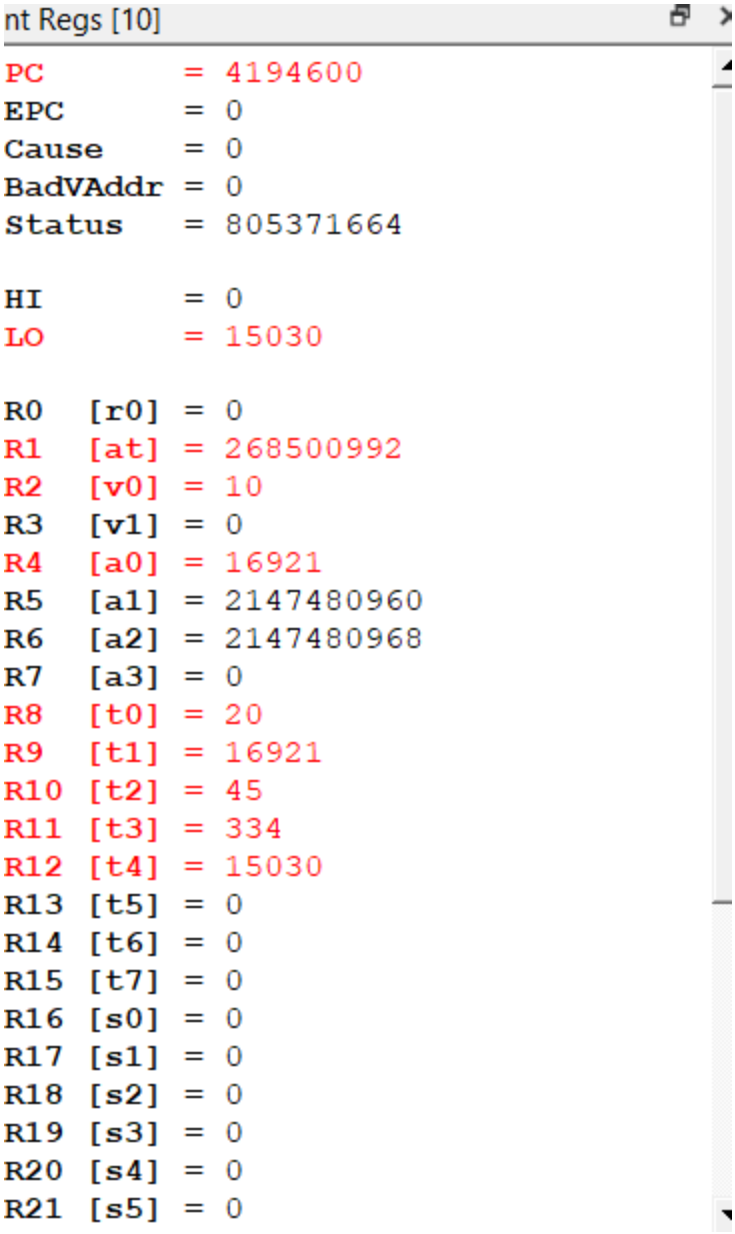






5*23+45*0+7*234+6*23+45*334==16921

Registers after execution of the code



Q.3. Use lb $t1, 5($zero) to cause an exception when attempting to load a byte from address 5. What is
the address of the lb instruction in your program? What is the value of the cause register, the
exception code, the vaddr, and the epc when the exception occurs?

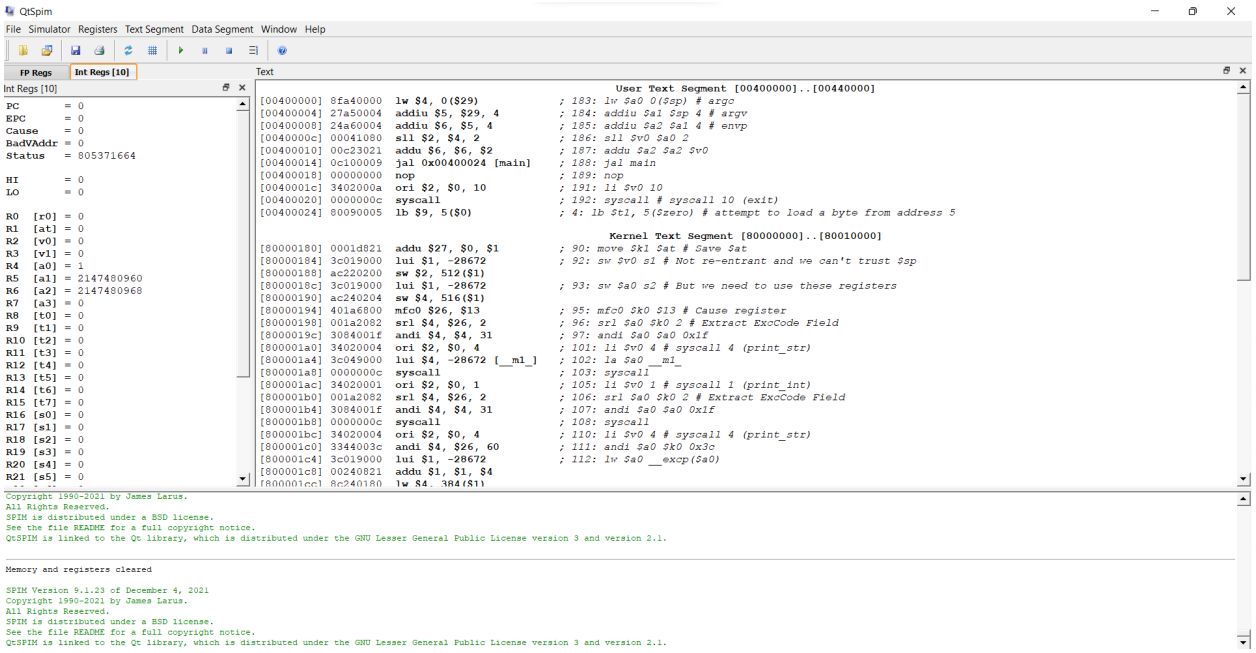
```
.text  # tells assembler to switch to the text segment or succeeding lines contains instructions
```

```
main: # start of code section
  lb $t1, 5($zero)    # attempt to load a byte from address 5
```

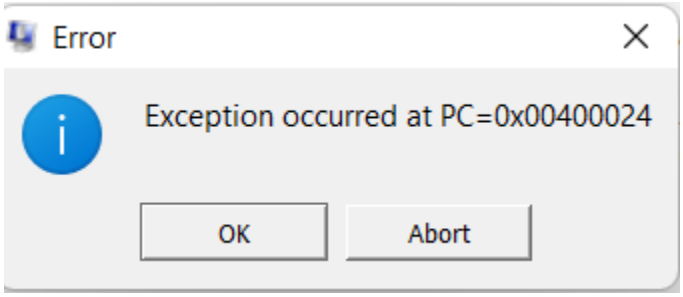
Brief overview of the code section

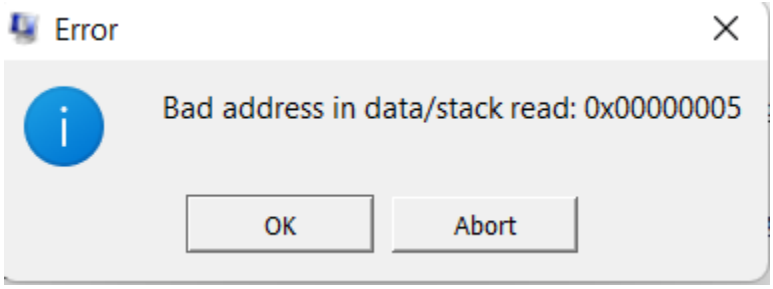
lb $t1, 5($zero) is used to cause an exception

After loading the third file

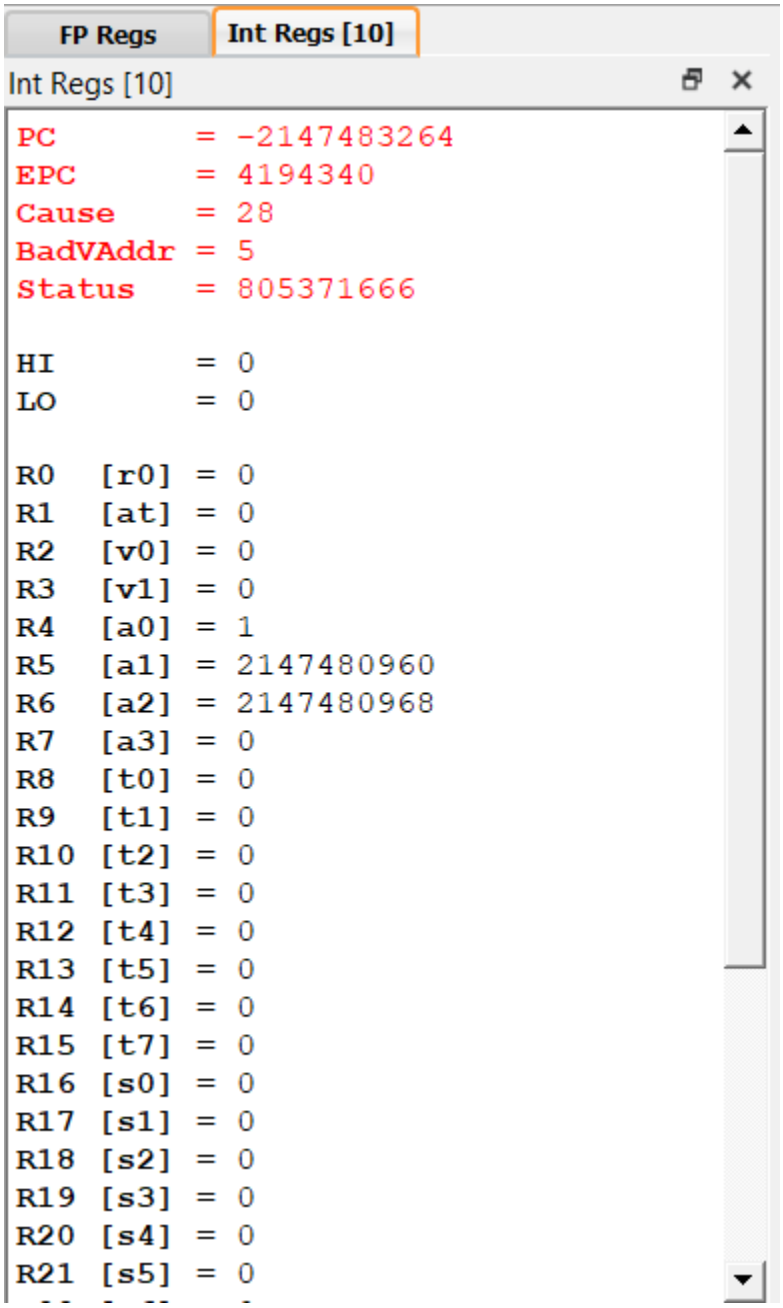


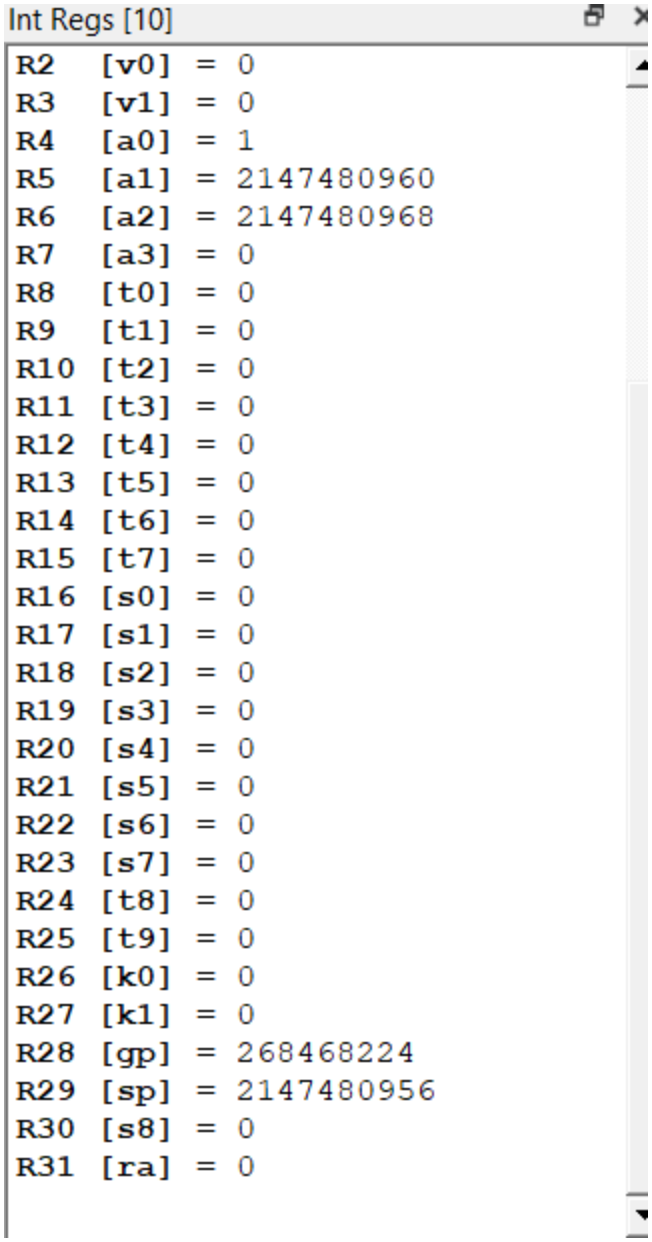
Error messages

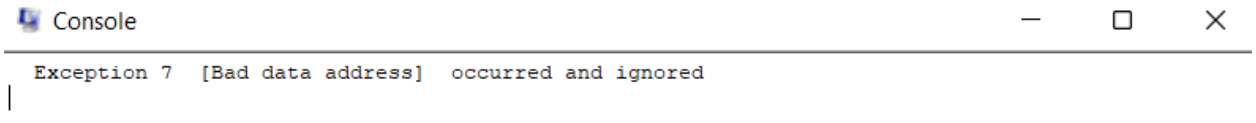




Register values




Int Regs [10]
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 2147480960
R6 [a2] = 2147480968
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 268468224
R29 [sp] = 2147480956
R30 [s8] = 0
R31 [ra] = 0


Console
Exception 7 [Bad data address] occurred and ignored

```
[0040001c] 3402000a  ori $2, $0, 10          ; 191: li $v0 10
[00400020] 0000000c  syscall                 ; 192: syscall # syscall 10 (exit)
[00400024] 80090005  lb $9, 5($0)            ; 4: lb $t1, 5($zero) # attempt to load a byte from address 5
```

Address of lb instruction is 0040024 in my program
Value of cause register = 28
Exception code = 7
VAddr=5
EPC= 4194340
Status =805371666

An exception is a signal that indicates an error or abnormal condition has occurred during program execution
Exception code in MIPS Assembly refers to the code that is executed in response to an exception or interrupt.
The "Cause" register in MIPS architecture is a register that stores information about the cause of the most recent exception or interrupt. It holds information about the type of exception or interrupt that occurred and the associated status bits.
EPC stands for "Exception Program Counter" in MIPS architecture, it's a register that holds the address of the instruction that caused an exception or an interrupt. The EPC register is used by the exception/interrupt handler to determine the location of the faulting instruction, so that the processor can return to that instruction once the exception/interrupt has been handled.
A "Bad Data Address" exception is a type of exception that occurs when a program attempts to access memory that is not accessible or that it does not have permission to access. This can occur when a program tries to access a memory location outside of its assigned memory space, or when it tries to access a protected memory location that is reserved for the operating system or another program.