

CS211: LAB Report

Assembly Related Labs

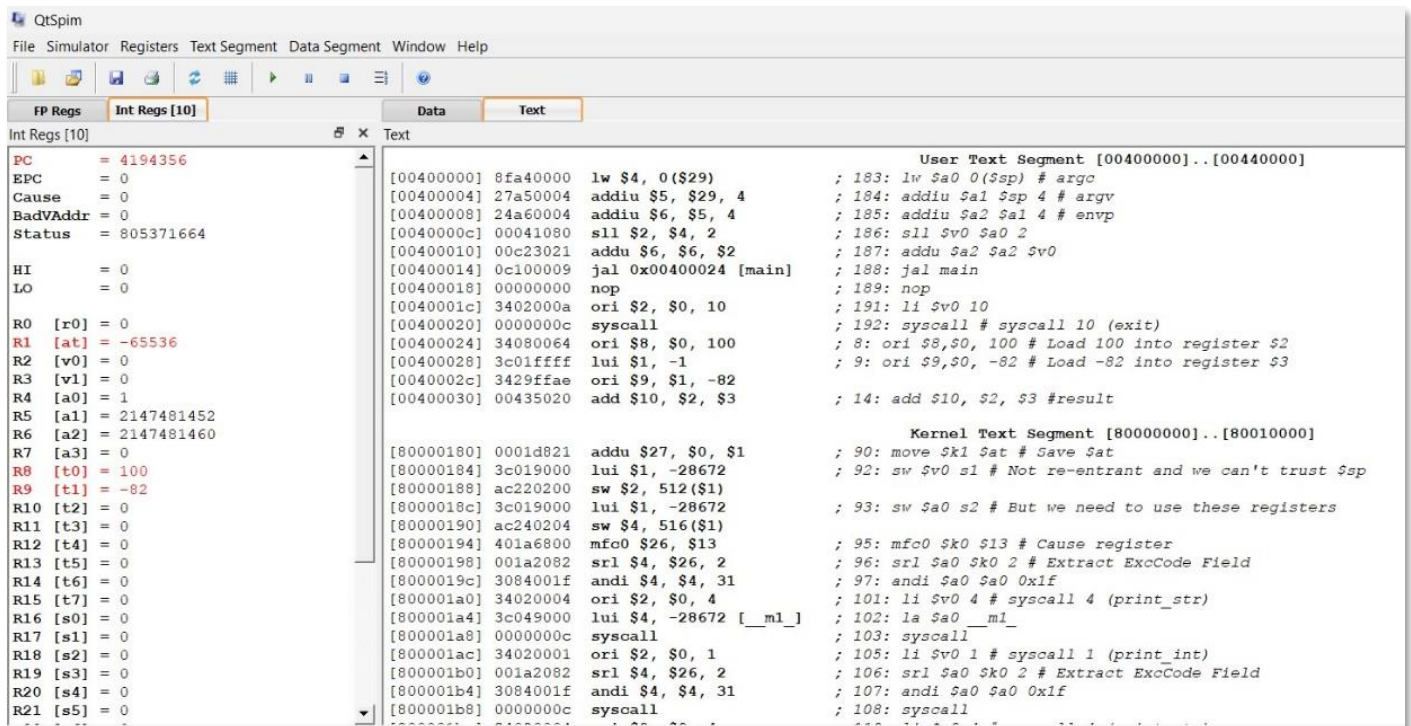
By Rupesh Bhusare, 2203106

LAB 1:

Question 1:

Add numbers 100 and -82 and store the result in register \$10. (Hint: Find 2's complement of 82 and add it to 100)

Output screenshot:



The screenshot shows the QtSpim MIPS simulator interface. The 'Int Regs [10]' window is open, displaying the following register values:

Register	Value
PC	4194356
EPC	0
Cause	0
BadVAddr	0
Status	805371664
HI	0
LO	0
R0 [r0]	0
R1 [at]	-65536
R2 [v0]	0
R3 [v1]	0
R4 [a0]	1
R5 [a1]	2147401452
R6 [a2]	2147481460
R7 [a3]	0
R8 [t0]	100
R9 [t1]	-82
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

The main window displays the assembly code for the 'User Text Segment [00400000]..[00440000]'. The code is as follows:

```
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # arg0
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 34080064 ori $8, $0, 100 ; 8: ori $8,$0, 100 # Load 100 into register $2
[00400028] 3c01ffff lui $1, -1 ; 9: ori $9,$0, -82 # Load -82 into register $3
[0040002c] 3429ffae ori $9, $1, -82
[00400030] 00435020 add $10, $2, $3 ; 14: add $10, $2, $3 #result
```

The 'Kernel Text Segment [80000000]..[80010000]' is also visible, containing code for saving and restoring registers.

Question 2:

Store FFFF in \$8. Left shift it 2 times and store the result in memory location 0x10000000.

Output screenshot:

The screenshot shows the QtSpim MIPS simulator interface. The 'Int Regs [10]' tab is selected, displaying the following register values:

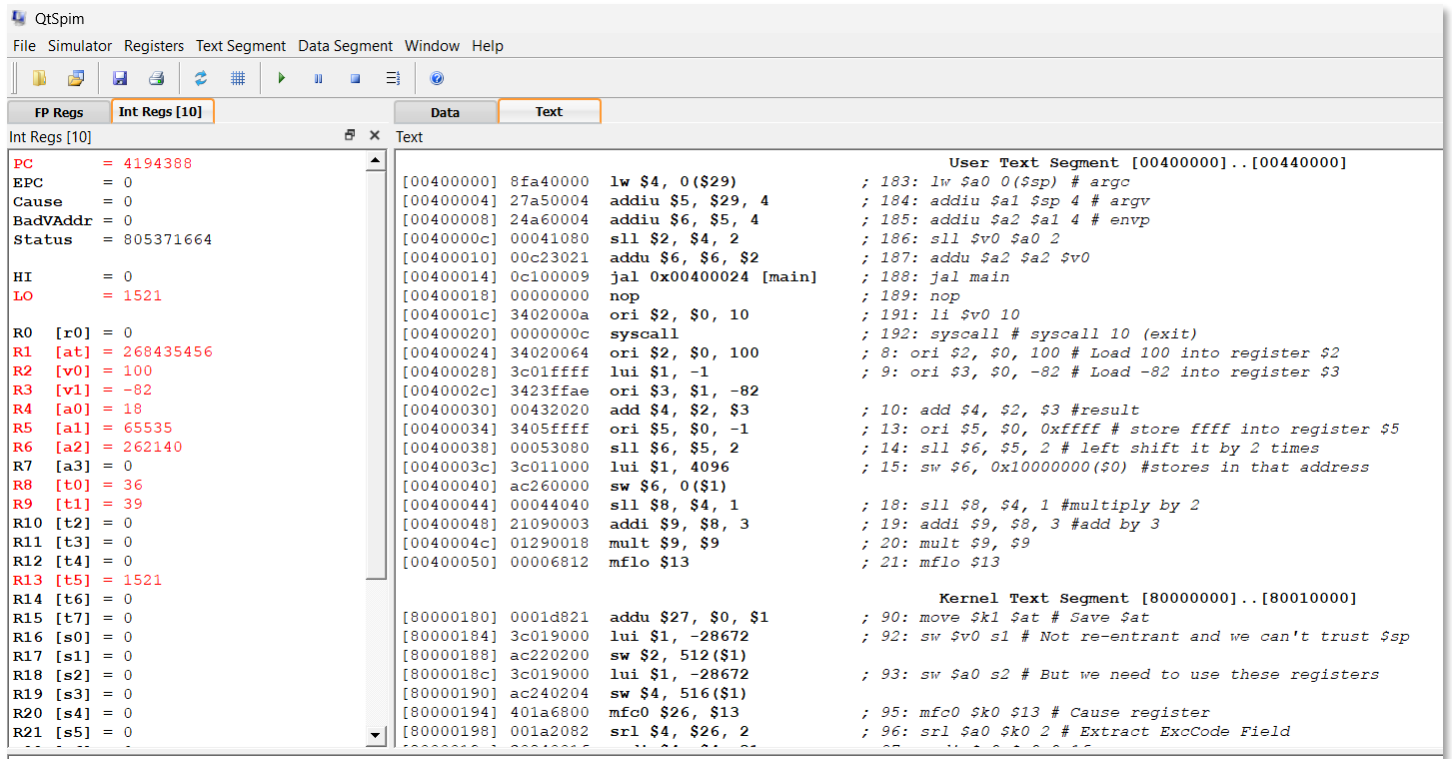
Register	Value
PC	4194356
EPC	0
Cause	0
BadVAddr	0
Status	805371664
HI	0
LO	0
R0 [r0]	0
R1 [at]	268435456
R2 [v0]	0
R3 [v1]	0
R4 [a0]	1
R5 [a1]	2147481452
R6 [a2]	2147481460
R7 [a3]	0
R8 [t0]	65535
R9 [t1]	262140
R10 [t2]	0
R11 [t3]	0
R12 [t4]	0
R13 [t5]	0
R14 [t6]	0
R15 [t7]	0
R16 [s0]	0
R17 [s1]	0
R18 [s2]	0
R19 [s3]	0
R20 [s4]	0
R21 [s5]	0

The 'Text' tab is also visible, showing the assembly code for the 'User Text Segment [00400000]..[00440000]'. The code includes instructions for loading arguments, setting up the environment, and performing system calls. The 'Kernel Text Segment [80000000]..[80010000]' is also visible, containing code for saving registers and handling exceptions.

Question 3:

Evaluate the expression $(2x+3)^2$ where x is the content in register \$10 based on exercise (a). Store the result in register \$13.

Output screenshot:



QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs Int Regs [10] Data Text

Int Regs [10]

```
PC = 4194388
EPC = 0
Cause = 0
BadVAddr = 0
Status = 805371664
HI = 0
LO = 1521
R0 [r0] = 0
R1 [at] = 268435456
R2 [v0] = 100
R3 [v1] = -82
R4 [a0] = 18
R5 [a1] = 65535
R6 [a2] = 262140
R7 [a3] = 0
R8 [t0] = 36
R9 [t1] = 39
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 1521
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
```

User Text Segment [00400000]..[00440000]

```
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 34020064 ori $2, $0, 100 ; 8: ori $2, $0, 100 # Load 100 into register $2
[00400028] 3c01ffff lui $1, -1 ; 9: ori $3, $0, -82 # Load -82 into register $3
[0040002c] 3423ffae ori $3, $1, -82
[00400030] 00432020 add $4, $2, $3 ; 10: add $4, $2, $3 #result
[00400034] 3405ffff ori $5, $0, -1 ; 13: ori $5, $0, 0xffff # store ffff into register $5
[00400038] 00053080 sll $6, $5, 2 ; 14: sll $6, $5, 2 # left shift it by 2 times
[0040003c] 3c011000 lui $1, 4096 ; 15: sw $6, 0x10000000($0) #stores in that address
[00400040] ac260000 sw $6, 0($1)
[00400044] 00044040 sll $8, $4, 1 ; 18: sll $8, $4, 1 #multiply by 2
[00400048] 21090003 addi $9, $8, 3 ; 19: addi $9, $8, 3 #add by 3
[0040004c] 01290018 mult $9, $9 ; 20: mult $9, $9
[00400050] 00006812 mflo $13 ; 21: mflo $13
```

Kernel Text Segment [80000000]..[80010000]

```
[80000180] 0001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save $at
[80000184] 3c019000 lui $1, -28672 ; 92: sw $v0 $1 # Not re-entrant and we can't trust $sp
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672 ; 93: sw $a0 $2 # But we need to use these registers
[80000190] ac240204 sw $4, 516($1)
[80000194] 401a6800 mfc0 $26, $13 ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082 srl $4, $26, 2 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
```

LAB 2:

Question 1:

- 1) Complete the following code snippet to add 10 numbers stored consecutively in data memory. Print the result.

```
.data
array: .word 10,12,15,-10,13,82,-9,4,3,-7      #array={10,12,15,-10,13,82,-9,4,3,-7}
length: .word 10                               #load the length of the array as 10
sum: .word 0                                    #initialise sum to 0
.text
main:
la $t3, array                                  # load base address of the array
# $t3 has the base address of data. All the subsequent data can be accessed using respective offset
values.
#Add your code here
```

Count the total number of machine instructions executed to complete this task.

Output screenshot:

The screenshot shows the QtSpim MIPS simulator. The main window displays assembly code with comments. The left pane shows register values, and the bottom pane shows the console output.

Register Values (Left Pane):

- PC = 4194424
- EPC = 0
- Cause = 0
- BadVAddr = 0
- Status = 805371664
- HI = 0
- LO = 0
- R0 [r0] = 0
- R1 [at] = 268500992
- R2 [v0] = 10
- R3 [v1] = 0
- R4 [a0] = 113
- R5 [a1] = 2147481452
- R6 [a2] = 2147481460
- R7 [a3] = 0
- R8 [t0] = 113
- R9 [t1] = -7
- R10 [t2] = 0
- R11 [t3] = 268501032
- R12 [t4] = 10
- R13 [t5] = 268501032
- R14 [t6] = 10
- R15 [t7] = 0
- R16 [s0] = 0
- R17 [s1] = 0
- R18 [s2] = 0
- R19 [s3] = 0
- R20 [s4] = 0
- R21 [s5] = 0

Assembly Code (Main Window):

```
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 34080000 ori $8, $0, 0 ; 15: li $t0, 0 # initially sum is zero in $t0
[00400028] 3c0b1001 lui $11, 4097 [array] ; 16: la $t3, array # load base address of the array in $t3
[0040002c] 3c011001 lui $1, 4097 [length] ; 17: la $t5, length # load base address of the length in $t5
[00400030] 342d0028 ori $13, $1, 40 [length]
[00400034] 8dae0000 lw $14, 0($13) ; 18: lw $t6, 0($t5) # load length in $t6
[00400038] 340c0000 ori $12, $0, 0 ; 19: li $t4, 0 # counter
[0040003c] 018e082a slt $1, $12, $14 ; 22: bge $t4, $t6, end_sum_loop # branch if counter >= length
[00400040] 10200006 beq $1, $0, 24 [end_sum_loop-0x00400040]
[00400044] 8d690000 lw $9, 0($11) ; 24: lw $t1, 0($t3) # load word in $t1
[00400048] 01094020 add $8, $8, $9 ; 25: add $t0, $t0, $t1 # add to sum
[0040004c] 216b0004 addi $11, $11, 4 ; 26: addi $t3, $t3, 4 # increase address by 4
[00400050] 218c0001 addi $12, $12, 1 ; 27: addi $t4, $t4, 1 # increase counter by 1
[00400054] 0810000f j 0x0040003c [sum_loop] ; 28: j sum_loop # jump back to the beginning of the loop
[00400058] 3402000a ori $2, $0, 4 ; 32: li $v0, 4 # system call code for printing string -4
[0040005c] 3c011001 lui $1, 4097 [out_string] ; 33: la $a0, out_string # load addr of string to be printed
[00400060] 34240030 ori $4, $1, 48 [out_string]
[00400064] 0000000c syscall ; 34: syscall #call op system
[00400068] 34020001 ori $2, $0, 1 ; 36: li $v0, 1 # system call code for printing integer - 1
[0040006c] 00082021 addu $4, $0, $8 ; 37: move $a0, $t0 # load the sum into $a0
[00400070] 0000000c syscall ; 38: syscall # call op system
[00400074] 3402000a ori $2, $0, 10 ; 40: li $v0, 10 #terminate program
[00400078] 0000000c syscall ; 41: syscall
```

Console Output (Bottom Pane):

```
Total sum is:
113
```

Ans: Count of total number of instructions: 59

Question 2:

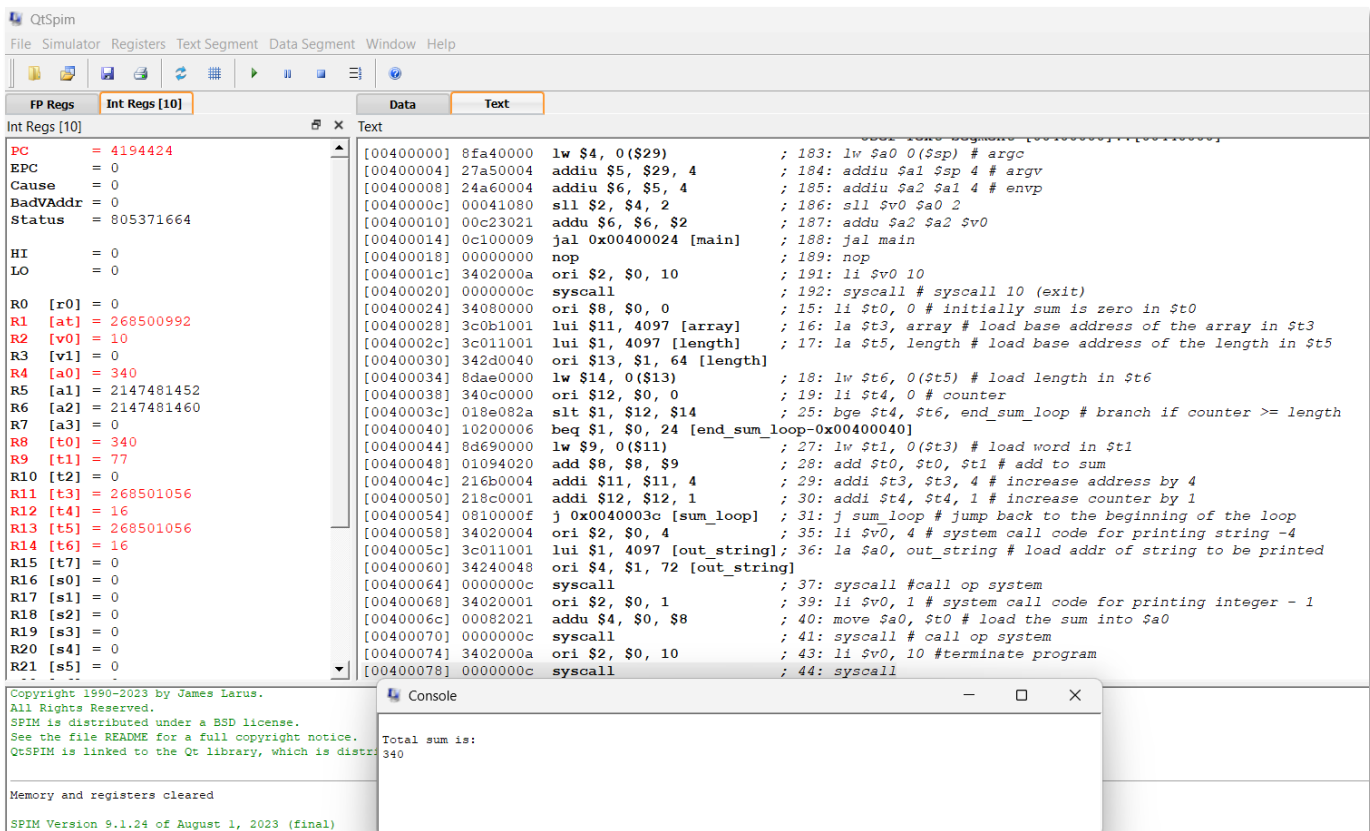
- 2) Include the following numbers in the array data segment of question 1.
10,20,30,40,50,77

Now you have 16 numbers residing in the array (data memory). Add these numbers and display the result.

Count the total number of instructions.

Compare and analyse the relation between the number of data elements and total number of machine instructions executed.

Output screenshot:



Ans: Count of the total number of instructions: 89.

The relationship between the number of data elements and the total number of machine instructions executed is linear. The number of instructions inside the loop is executed once for each data element. So, if you increase the number of data elements, the total number of instructions executed will also increase proportionally.

Question 3:

3) Compute the Euler Phi function for the number 21.

Euler's Phi function for an input n , denoted as $\phi(n)$ is the count of numbers in $\{1, 2, 3, \dots, n\}$ that are relatively prime to n , i.e, the numbers whose GCD (Greatest Common Divisor) with n is 1.

Examples:

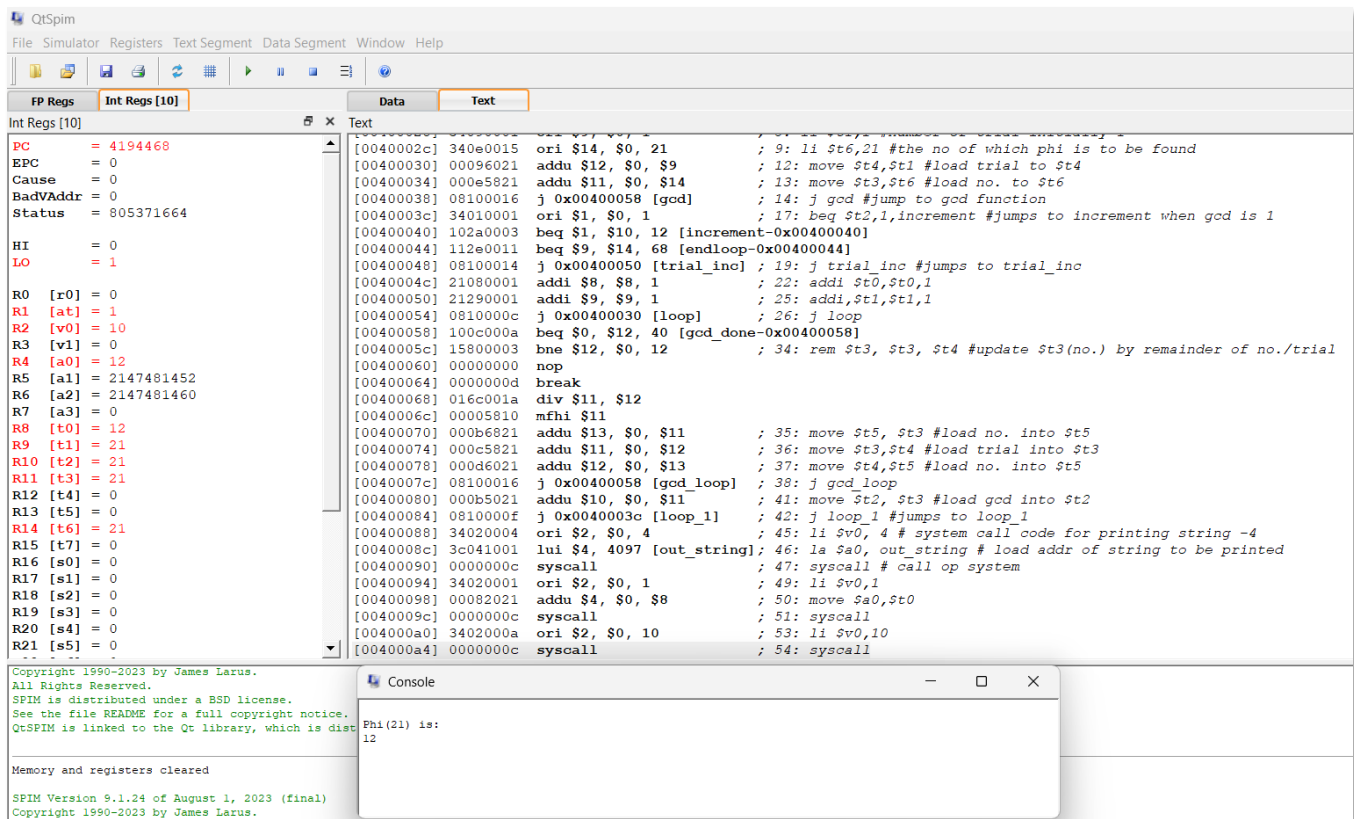
$\phi(1)=1$, $(\gcd(1,1)=1)$
 $\phi(2)=1$, $(\gcd(1,2)=1, \text{ but } \gcd(2,2)=2)$
 $\phi(3)=2$, $(\gcd(1,3)=1, \gcd(2,3)=1, \gcd(3,3)=3)$
 $\phi(4)=2$, $(\gcd(1,4)=1, \gcd(2,4)=2, \gcd(3,4)=1, \gcd(4,4)=4)$
 $\phi(5)=4$, $(\gcd(1,5)=1, \gcd(2,5)=1, \gcd(3,5)=1, \gcd(4,5)=1, \gcd(5,5)=5)$

(Hint: The logic for your code would be as follows

```
phi = 0;
trial = 1;
while ( trial < N)           #where N is the number under consideration (given : 21)
{
    if ( gcd(N,trial) == 1 ) phi++;
}
```

The computed value, which is $\phi(21)$, should be printed on the screen.

Output screenshot:



LAB 3:

Question 1:

Reverse a string entered by user. (Hint: Ask 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.)

Output screenshot:

The screenshot displays the QtSpim MIPS simulator interface. The main window is divided into several panes. On the left, the 'Int Regs [10]' pane shows the current state of the integer registers. The 'Text' pane displays the assembly code being executed. At the bottom, a 'Console' window shows the interaction with the user.

Int Regs [10]:

Register	Value
PC	4194428
EPC	0
Cause	0
BadVAddr	0
Status	805371664
HI	0
LO	0
R0 [r0]	0
R1 [at]	268500992
R2 [v0]	10
R3 [v1]	0
R4 [a0]	268501081
R5 [a1]	268501092
R6 [a2]	2147481372
R7 [a3]	0
R8 [t0]	268501004
R9 [t1]	268501080
R10 [t2]	0
R11 [t3]	0
R12 [t4]	0
R13 [t5]	11
R14 [t6]	0
R15 [t7]	0
R16 [s0]	0
R17 [s1]	0
R18 [s2]	0
R19 [s3]	0
R20 [s4]	0
R21 [s5]	0

Text Pane (Assembly Code):

```
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 34020004 ori $2, $0, 4 ; 10: li $v0, 4 #to print string
[00400028] 3c011001 lui $1, 4097 [str] ; 11: la $a0, str
[0040002c] 342400c8 ori $4, $1, 200 [str] ; 
[00400030] 0000000c syscall ; 12: syscall
[00400034] 34020008 ori $2, $0, 8 ; 14: li $v0, 8 #to read user input
[00400038] 3c041001 lui $4, 4097 [buffer] ; 15: la $a0, buffer #load addr of buffer into $a0
[0040003c] 34050064 ori $5, $0, 100 ; 16: li $a1, 100 #maximum no. of character to read
[00400040] 0000000c syscall ; 17: syscall
[00400044] 34020004 ori $2, $0, 4 ; 19: li $v0, 4 #to print string
[00400048] 3c011001 lui $1, 4097 [str_1] ; 20: la $a0, str_1
[0040004c] 342400d8 ori $4, $1, 216 [str_1] ; 
[00400050] 0000000c syscall ; 21: syscall
[00400054] 3c041001 lui $4, 4097 [buffer] ; 24: la $a0, buffer #load address of the input buffer into $a0
[00400058] 3c011001 lui $1, 4097 [reversed] ; 25: la $a1, reversed #load address of the reversed buffer into $a1
[0040005c] 34250064 ori $5, $1, 100 [reversed] ; 
[00400060] 0c100020 jal 0x00400080 [reverse_string]; 26: jal reverse_string #jump and link to reverse_string
[00400064] 34020004 ori $2, $0, 4 ; 29: li $v0, 4
[00400068] 3c011001 lui $1, 4097 [reversed] ; 30: la $a0, reversed
[0040006c] 34240064 ori $4, $1, 100 [reversed] ; 
[00400070] 00842022 sub $4, $4, $13 ; 31: sub $a0, $a0, $t5
[00400074] 0000000c syscall ; 32: syscall
[00400078] 3402000a ori $2, $0, 10 ; 35: li $v0, 10
[0040007c] 0000000c syscall ; 36: syscall
```

Console:

```
Enter a string:Hello world
Reversed string is:
dlrow olleH
```

Memory and registers cleared

SPIM Version 9.1.24 of August 1, 2023 (final)
Copyright 1990-2023 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.

Question 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 sum of product of the corresponding elements. For example, (1,2,3) dot (4,5,6) is $1*4+2*5+3*6 = 32$)

Output screenshot:

The screenshot shows the QtSPIM simulator interface. The main window displays MIPS assembly code for calculating the dot product of two vectors. The code includes instructions for loading registers, performing arithmetic, and using system calls for input/output. The console window at the bottom shows the user's input and the resulting dot product.

```
PC = 4194572
EPC = 0
Cause = 0
BadVAddr = 0
Status = 805371664
HI = 0
LO = 18
R0 [r0] = 0
R1 [at] = 268500992
R2 [v0] = 10
R3 [v1] = 0
R4 [a0] = 32
R5 [a1] = 2147481364
R6 [a2] = 2147481372
R7 [a3] = 0
R8 [t0] = 3
R9 [t1] = 268501004
R10 [t2] = 268501044
R11 [t3] = 3
R12 [t4] = 6
R13 [t5] = 18
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 32
R17 [s1] = 3
R18 [s2] = 0
R19 [s3] = 3
R20 [s4] = 3
R21 [s5] = 0
```

```
[00400094] 34020004 ori $2, $0, 4 ; 49: li $v0, 4
[00400098] 3c011001 lui $1, 4097 [arr_input] ; 50: la $a0, arr_input # load the address of the string to be printed
[0040009c] 34240083 ori $4, $1, 131 [arr_input]
[004000a0] 0000000c syscall ; 51: syscall
[004000a4] 12910007 beq $20, $17, 28 [end_read2-0x004000a4]
[004000a8] 34020005 ori $2, $0, 5 ; 56: li $v0, 5 # read integer
[004000ac] 0000000c syscall ; 57: syscall
[004000b0] af220000 sw $2, 0($25) ; 59: sw $v0, 0($t9) # store the integer in memory
[004000b4] 23390004 addi $25, $25, 4 ; 60: addi $t9, $t9, 4 # increment the address
[004000b8] 22940001 addi $20, $20, 1 ; 61: addi $s4, $s4, 1 # increment the counter
[004000bc] 08100029 j 0x004000a4 [read_arr2] ; 62: j read_arr2 # jump to read_arr2
[004000c0] 34080000 ori $8, $0, 0 ; 66: li $t0, 0 # counter for loop
[004000c4] 34100000 ori $16, $0, 0 ; 67: li $s0, 0 # store the sum of the product
[004000c8] 11110009 beq $8, $17, 36 [end_loop-0x004000c8]
[004000cc] 9d2b0000 lw $11, 0($9) ; 72: lw $t3, 0($t1) # load the element of arr1 in $t3
[004000d0] 8d4c0000 lw $12, 0($10) ; 73: lw $t4, 0($t2) # load the element of arr2 in $t4
[004000d4] 716c6802 mul $13, $11, $12 ; 75: mul $t5, $t3, $t4 # multiply the elements
[004000d8] 020d8020 add $16, $16, $13 ; 76: add $s0, $s0, $t5 # add the product to sum
[004000dc] 21290004 addi $9, $9, 4 ; 78: addi $t1, $t1, 4 # increment the address of arr1
[004000e0] 214a0004 addi $10, $10, 4 ; 79: addi $t2, $t2, 4 # increment the address of arr2
[004000e4] 21080001 addi $8, $8, 1 ; 80: addi $t0, $t0, 1 # increment the counter
[004000e8] 08100032 j 0x004000c8 [loop] ; 81: j loop # jump to loop
[004000ec] 34020004 ori $2, $0, 4 ; 85: li $v0, 4 # to print the string
[004000f0] 3c011001 lui $1, 4097 [ans] ; 86: la $a0, ans # load the address of the string to be printed
[004000f4] 34240098 ori $4, $1, 152 [ans]
[004000f8] 0000000c syscall ; 87: syscall
[004000fc] 34020001 ori $2, $0, 1 ; 89: li $v0, 1 # to print integer
[00400100] 00102021 addu $4, $0, $16 ; 90: move $a0, $s0 # load the sum in $a0
[00400104] 0000000c syscall ; 91: syscall
[00400108] 3402000a ori $2, $0, 10 ; 93: li $v0, 10 # exit
[0040010c] 0000000c syscall ; 94: syscall
```

```
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under
spim: (parser) Label is defined for the second time on line 11 of
main:
^
Memory and registers cleared

SPIM Version 9.1.24 of August 1, 2023 (final)
Copyright 1990-2023 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the
GNU Lesser General Public License version 3 and version 2.1
```

```
Console
Length of vectors:3
Enter the elements:
1
2
3
Enter the elements:
4
5
6
Dot product of vectors:32
```


Question 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?

Output screenshot:

The screenshot shows the QtSPIM simulator interface. The 'Text' window displays assembly code. The 'Int Regs [10]' window shows the state of registers. The 'Console' window shows an exception message.

Int Regs [10]

Register	Value
PC	4194344
EPC	4194344
Cause	0
BadVAddr	5
Status	805371665
HI	0
LO	0
R0 [r0]	0
R1 [at]	0
R2 [v0]	0
R3 [v1]	0
R4 [a0]	1
R5 [a1]	2147481364
R6 [a2]	2147481372
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

Text Window

```
00400000: 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
00400004: 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
00400008: 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
0040000c: 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
00400010: 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
00400014: 0c100009 jal 0x00400024 [main] ; 188: jal main
00400018: 00000000 nop ; 189: nop
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)

80000180: 0001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save $at
80000184: 3c019000 lui $1, -28672 ; 92: sw $v0 $1 # Not re-entrant and we can't trust $sp
80000188: ac220200 sw $2, 512($1) ; 93: sw $a0 $2 # But we need to use these registers
8000018c: 3c019000 lui $1, -28672
80000190: ac240204 sw $4, 516($1)
80000194: 401a6800 mfc0 $26, $13 ; 95: mfc0 $k0 $13 # Cause register
80000198: 001a2082 srl $4, $26, 2 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
8000019c: 3084001f andi $4, $4, 31 ; 97: andi $a0 $a0 0x1f
800001a0: 3402000a ori $2, $0, 4 ; 101: li $v0 4 # syscall 4 (print_str)
800001a4: 3c049000 lui $4, -28672 [__m1_] ; 102: la $a0 __m1_
800001a8: 0000000c syscall ; 103: syscall
800001ac: 34020001 ori $2, $0, 1 ; 105: li $v0 1 # syscall 1 (print_int)
800001b0: 001a2082 srl $4, $26, 2 ; 106: srl $a0 $k0 2 # Extract ExcCode Field
800001b4: 3084001f andi $4, $4, 31 ; 107: andi $a0 $a0 0x1f
800001b8: 0000000c syscall ; 108: syscall
800001bc: 3402000a ori $2, $0, 4 ; 110: li $v0 4 # syscall 4 (print_str)
800001c0: 3344003c andi $4, $26, 60 ; 111: andi $a0 $k0 0x3c
800001c4: 3c019000 lui $1, -28672 ; 112: lw $a0 __excp($a0)
```

Console

```
Exception 7 [Bad data address] occurred and ignored
```

Ans:

Cause register is 0.

Exception code is 7.

Vaddr is 5.

Epc when exception occurred is 4194344.

LAB 4:

Question 1:

Evaluate the expression 'ab-10a+20b+16'. Consider that only \$t0 and \$t1 are available to store temporary values. Store a=10 and b=20 in data section. Use stack for other memory requirements. Display the sum.

Output screenshot:

The screenshot shows the QtSPIM MIPS simulator interface. The 'Registers' window on the left displays the state of various registers. The 'Text' window in the center shows the assembly code for the program. The 'Console' window at the bottom shows the output of the program.

Registers:

Register	Value
PC	4194464
EPC	0
Cause	0
BadVAddr	0
Status	805371664
HI	0
LO	100
R0 [r0]	0
R1 [at]	268500992
R2 [v0]	1
R3 [v1]	0
R4 [a0]	516
R5 [a1]	2147481372
R6 [a2]	2147481380
R7 [a3]	0
R8 [t0]	516
R9 [t1]	16
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

Text Segment [00400000]..[00440000]:

```
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 23bdfbf0 addi $29, $29, -16 ; 9: addi $sp,$sp,-16 # Allocate 16 bytes on the stack
[00400028] 3c011001 lui $1, 4097 ; 11: lw $t0,n1 # Load n1 into $t0
[0040002c] 8c280000 lw $8, 0($1) ;
[00400030] 3c011001 lui $1, 4097 ;
[00400034] 8c290004 lw $9, 4($1) ; 12: lw $t1,n2 # Load n2 into $t1
[00400038] 71284002 mul $8, $9, $8 ; 14: mul $t0,$t1,$t0 # Multiply $t1 and $t0, store result in $t0
[0040003c] afa80000 sw $8, 0($29) ; 16: sw $t0,0($sp) # Store result in $t0 on the stack at offset 0
[00400040] 34080014 ori $8, $0, 20 ; 18: li $t0,20 # Load immediate value 20 into $t0
[00400044] 3c011001 lui $1, 4097 ; 19: lw $t1,n2 # Load n2 into $t1
[00400048] 8c290004 lw $9, 4($1) ;
[0040004c] 71094002 mul $8, $8, $9 ; 21: mul $t0,$t0,$t1 # Multiply $t0 and $t1, store result in $t0
[00400050] 8fa90000 lw $9, 0($29) ; 23: lw $t1,0($sp) # Load value from stack at offset 0 into $t1
[00400054] 01094020 add $8, $8, $9 ; 24: add $t0,$t0,$t1 # Add $t0 and $t1, store result in $t0
[00400058] afa80000 sw $8, 0($29) ; 26: sw $t0,0($sp) # Store result in $t0 on the stack at offset 0
[0040005c] 3c011001 lui $1, 4097 ; 28: lw $t0,n1 # Load n1 into $t0
[00400060] 8c280000 lw $8, 0($1) ;
[00400064] 3409000a ori $9, $0, 10 ; 29: li $t1,10 # Load immediate value 10 into $t1
[00400068] 71094002 mul $8, $8, $9 ; 31: mul $t0,$t0,$t1 # Multiply $t0 and $t1, store result in $t0
[0040006c] 8fa90000 lw $9, 0($29) ; 32: lw $t1,0($sp) # Load value from stack at offset 0 into $t1
[00400070] 01284022 sub $8, $9, $8 ; 33: sub $t0,$t1,$t0 # Subtract $t0 from $t1, store result in $t0
[00400074] afa80000 sw $8, 0($29) ; 35: sw $t0,0($sp) # Store result in $t0 on the stack at offset 0
```

Console:

```
Ans is:516
```

Question 2:

Find the maximum of the three expressions: $x*x$; $x*y$; $y*5$. Take x and y as input from user. Write a global subroutine, in another file, to calculate values of these expressions. Write a subroutine to find maximum of two integers and use it to find the maximum of these three expressions. Display the result.

Output screenshot:

The screenshot displays the QtSpim MIPS simulator interface. The 'Int Regs [10]' window shows the following register values:

Register	Value
PC	4194440
EPC	0
Cause	0
BadVAddr	0
Status	805371664
HI	0
LO	50
R0 [r0]	0
R1 [at]	268500992
R2 [v0]	10
R3 [v1]	0
R4 [a0]	625
R5 [a1]	10
R6 [a2]	2147481380
R7 [a3]	0
R8 [t0]	25
R9 [t1]	10
R10 [t2]	625
R11 [t3]	250
R12 [t4]	50
R13 [t5]	5
R14 [t6]	0
R15 [t7]	0
R16 [s0]	625
R17 [s1]	0
R18 [s2]	0
R19 [s3]	0
R20 [s4]	0
R21 [s5]	0

The 'Text' window shows the assembly code for the program:

```
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 34020004 ori $2, $0, 4 ; 8: li $v0, 4 # for print string
[00400028] 3c041001 lui $4, 4097 [input_x] ; 9: la $a0, input_x # Set string to print
[0040002c] 0000000c syscall ; 10: syscall
[00400030] 34020005 ori $2, $0, 5 ; 12: li $v0, 5 # for read int x
[00400034] 0000000c syscall ; 13: syscall
[00400038] 00024021 addu $8, $0, $2 ; 14: move $t0, $v0 # Store x
[0040003c] 34020004 ori $2, $0, 4 ; 16: li $v0, 4 # for print string
[00400040] 3c011001 lui $1, 4097 [input_y] ; 17: la $a0, input_y # Set string to print
[00400044] 3424000a ori $4, $1, 10 [input_y] ; 18: syscall
[00400048] 0000000c syscall ; 20: li $v0, 5 # for read int y
[0040004c] 34020005 ori $2, $0, 5 ; 21: syscall
[00400050] 0000000c syscall ; 22: move $t1, $v0 # Store y
[00400054] 00024821 addu $9, $0, $2 ; 24: move $a0, $t0 # Set first arg for find_max
[00400058] 00082021 addu $4, $0, $8 ; 25: move $a1, $t1 # Set second arg for find_max
[0040005c] 00092821 addu $5, $0, $9 ; 26: jal find_max # Call find_max
[00400060] 0c100023 jal 0x0040008c [find_max] ; 27: move $s0, $v0 # Store result
[00400064] 00028021 addu $16, $0, $2 ; 29: li $v0, 4 # for print string
[00400068] 34020004 ori $2, $0, 4 ; 30: la $a0, largest # string to print
[0040006c] 3c011001 lui $1, 4097 [largest] ; 31: syscall
[00400070] 34240014 ori $4, $1, 20 [largest] ; 33: li $v0, 1 # for print int
[00400074] 0000000c syscall ; 34: move $a0, $s0 # Set int to print
[00400078] 34020001 ori $2, $0, 1 ; 35: syscall
[0040007c] 00102021 addu $4, $0, $16 ; 37: li $v0, 10 # for exit
[00400080] 0000000c syscall ; 38: syscall
[00400084] 3402000a ori $2, $0, 10
[00400088] 0000000c syscall
```

The console window shows the following output:

```
Enter x: 25
Enter y: 10
Largest of x*x, x*y and y*5 is: 625
```

The bottom status bar indicates an exception occurred at PC=0x00400024, with a bad address in data/stack read at 0x00000005 and an attempt to execute non-instruction at 0x00400028.