

Name-Pranav Tambe
Roll No-2106339



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

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

IP RegsInt Regs [10]

x

PC = 0

EPC = 0

Cause = 0

BadVAddr = 0

Status = 805371664

HI = 0

LO = 0

R0 [r0] = 0

R1 [at] = 0

R2 [v0] = 0

R3 [v1] = 0

R4 [a0] = 1

R5 [a1] = 2147480960

R6 [a2] = 2147480960

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 [a0] = 0

R17 [s1] = 0

R18 [s2] = 0

R19 [s3] = 0

R20 [s4] = 0

R21 [s5] = 0

x

[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)

[00400024] 34020004 ori \$2, \$0, 4 ; 12: li \$v0, 4

[00400028] 3c011001 lui \$1, 4097 [string_one]; 13: la \$a0,string_one

[0040002c] 34240100 ori \$4, \$1, 256 [string_one]

[00400030] 0000000c syscall ; 14: syscall

[00400034] 34020008 ori \$2, \$0, 8 ; 17: li \$v0, 8 # Load the value 8 into the register \$v0

[00400038] 3c041001 lui \$4, 4097 [input_string]; 18: la \$a0, input_string # Load the address of the string "input_string" into the register \$a0

[0040003c] 34050080 ori \$5, \$0, 128 ; 19: li \$a1, 128 # Load the value 128 into the register \$a1

[00400040] 0000000c syscall ; 20: syscall # Call the system call with number stored in \$v0, using the values stored in \$a0 and \$a1

as arguments

[00400044] 3c081001 lui \$8, 4097 [input_string]; 23: la \$t0, input_string # load address of input_string in \$t0

[00400048] 01004820 add \$9, \$8, \$0 ; 24: add \$t1, \$t0, \$0 # t1 = t0 = address of input_string

[0040004c] 212a0080 addi \$10, \$9, 128 ; 25: add \$t2, \$t1, 128 # t2 = address of end of input_string

[00400050] 340b0000 ori \$11, \$0, 0 ; 26: li \$t3, 0 # t3 = length of string

[00400054] 112a0006 beq \$9, \$10, 24 [end_loop-0x00400054]

[00400058] 812c0000 lb \$12, 0(\$9) ; 30: lb \$t4, 0(\$t1) # t4 = value at memory location t1

[0040005c] 100c0004 beq \$0, \$12, 16 [end_loop-0x0040005c]

[00400060] 21290001 addi \$9, \$9, 1 ; 32: addi \$t1, \$t1, 1 # t1 = t1 + 1

[00400064] addi \$11, \$11, 1 ; 33: addi \$t3, \$t3, 1 # t3 = t3 + 1

[00400068] 08100015 j 0x00400054 [loop] ; 34: j loop

[0040006c] 010b4820 add \$9, \$8, \$11 ; 38: add \$t1, \$t0, \$t3 # t1 = address of end of input_string

[00400070] 210a0000 addi \$10, \$8, 0 ; 39: add \$t2, \$t0, 0 # t2 = address of start of input_string

[00400074] 3c011001 lui \$1, 4097 [output_string]; 40: la \$t5, output_string

[00400078] 342d0080 ori \$13, \$1, 128 [output_string]

[0040007c] 340e0000 ori \$14, \$0, 0 ; 41: li \$t6, 0 # t6 = length of output_string

[00400080] 112a0007 beq \$9, \$10, 28 [end_reverse_loop-0x00400080]

[00400084] addi \$9, \$9, -1 ; 45: addi \$t1, \$t1, -1 # t1 = t1 - 1

[00400088] 812c0000 lb \$12, 0(\$9) ; 46: lb \$t4, 0(\$t1) # t4 = value at memory location t1

[0040008c] alac0000 sb \$12, 0(\$13) ; 47: sb \$t4, 0(\$t5) # store t4 at memory location t5

[00400090] 21a80001 addi \$13, \$13, 1 ; 48: addi \$t5, \$t5, 1 # t5 = t5 + 1

[00400094] 21c80001 addi \$14, \$14, 1 ; 49: addi \$t6, \$t6, 1 # t6 = t6 + 1

[00400098] 08100020 j 0x00400080 [reverse_loop]; 50: j reverse_loop

[0040009c] 34020004 ori \$2, \$0, 4 ; 54: li \$v0, 4

Copyright 1990-2021 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.

Memory and registers cleared

SPIM Version 9.1.23 of December 4, 2021

Copyright 1990-2021 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

Console

Enter the string :|

After execution

Console

Enter the string :Hi I am Pranav

Reverse string :

vanarP ma I iH

Registers after execution of the code

FP Regs

Int Regs [10]

Int Regs [10]

PC = 4194496

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] = 268501120

R5 [a1] = 128

R6 [a2] = 2147480968

R7 [a3] = 0

R8 [t0] = 268500992

R9 [t1] = 268500992

R10 [t2] = 268500992

R11 [t3] = 15

R12 [t4] = 72

R13 [t5] = 268501135

R14 [t6] = 15

R15 [t7] = 0

R16 [s0] = 0

R17 [s1] = 0

R18 [s2] = 0

R19 [s3] = 0

R20 [s4] = 0

R21 [s5] = 0

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

The screenshot shows the QtSpim MIPS simulator interface. The 'Text' window displays the assembly code for the User Text Segment [00400000]..[00440000]. The code includes instructions for loading arguments, initializing registers, printing the integer result, and printing the string 'dot product'. The code is as follows:

```

[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $p, 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 ; 193: li $t0, 0 # initialize loop counter
[00400028] 34090000 ori $9, $0, 0 ; 194: li $t1, 0 # index of element in the vector
[0040002c] 34020004 ori $2, $0, 4 ; 195: li $v0, 4 # print string syscall
[00400030] 3c011001 lui $1, 4097 [string_1] ; 196: la $a0, string_1 # load string address
[00400034] 34240028 ori $4, $1, 40 [string_1] ; 197: syscall # call operating system to perform the operation
[00400038] 0000000c syscall ; 198: li $v0, 1 # print the integer
[0040003c] 34020001 ori $2, $0, 1 ; 199: move $a0, $t1 # move value in $a0 from $t1
[00400040] 00092021 addu $4, $0, $9 ; 200: syscall # call operating system to perform the operation
[00400044] 0000000c syscall ; 201: li $v0, 4 # print string
[00400048] 34020004 ori $2, $0, 4 ; 202: la $a0, string_3 # load string address
[0040004c] 3c011001 lui $1, 4097 [string_3] ; 203: syscall # call operating system to perform the operation
[00400050] 34240080 ori $4, $1, 128 [string_3] ; 204: li $v0, 5 # read int syscall
[00400054] 0000000c syscall ; 205: syscall # call operating system to perform the operation
[00400058] 34020005 ori $2, $0, 5 ; 206: addi $t1, $t1, 1 # increment the vector index
[0040005c] 0000000c syscall ; 207: sw $v0, vector_1($t0) # store input in vector_1
[00400060] 21290001 addi $9, $9, 1 ; 208: addi $t0, $t0, 4 # increment loop counter by 4 bytes
[00400064] 3c011001 lui $1, 4097 ; 209: bne $t0, 20, input_vector_1 # repeat until all values are entered by branch if not equal command
[00400068] 00280821 addu $1, $1, $8 ; 210: li $t0, 0 # initialize loop counter
[0040006c] ac220000 sw $2, 0($1) ; 211:
[00400070] 21080004 addi $8, $8, 4 ; 212:
[00400074] 34010014 ori $1, $8, 20 ; 213:
[00400078] 1428ffed bne $1, $8, -76 [input_vector_1-0x00400078] ; 214:
[0040007c] 34080000 ori $8, $0, 0 ; 215:

```

The console window at the bottom shows the following output:

```

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.
Attempt to execute non-instruction at 0x00400024

Memory and registers cleared

SPIM Version 9.1.23 of December 4, 2021
Copyright 1990-2021 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 images


```
Console
Enter the value in 1st vector at the index 0 :|
```

```
Console
Enter the value in 1st vector at the index 0 :5
Enter the value in 1st vector at the index 1 :45
Enter the value in 1st vector at the index 2 :7
Enter the value in 1st vector at the index 3 :6
Enter the value in 1st vector at the index 4 :
```

```
Console
Enter the value in 1st vector at the index 0 :5
Enter the value in 1st vector at the index 1 :45
Enter the value in 1st vector at the index 2 :7
Enter the value in 1st vector at the index 3 :6
Enter the value in 1st vector at the index 4 :45
Enter the value in 2nd vector at the index 0 :23
Enter the value in 2nd vector at the index 1 :0
Enter the value in 2nd vector at the index 2 :234
Enter the value in 2nd vector at the index 3 :23
Enter the value in 2nd vector at the index 4 :334
The dot product is :16921
```

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

Registers after execution of the code

nt Regs [10]		
PC	=	4194600
EPC	=	0
Cause	=	0
BadVAddr	=	0
Status	=	805371664
HI	=	0
LO	=	15030
R0	[r0]	= 0
R1	[at]	= 268500992
R2	[v0]	= 10
R3	[v1]	= 0
R4	[a0]	= 16921
R5	[a1]	= 2147480960
R6	[a2]	= 2147480968
R7	[a3]	= 0
R8	[t0]	= 20
R9	[t1]	= 16921
R10	[t2]	= 45
R11	[t3]	= 334
R12	[t4]	= 15030
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

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?

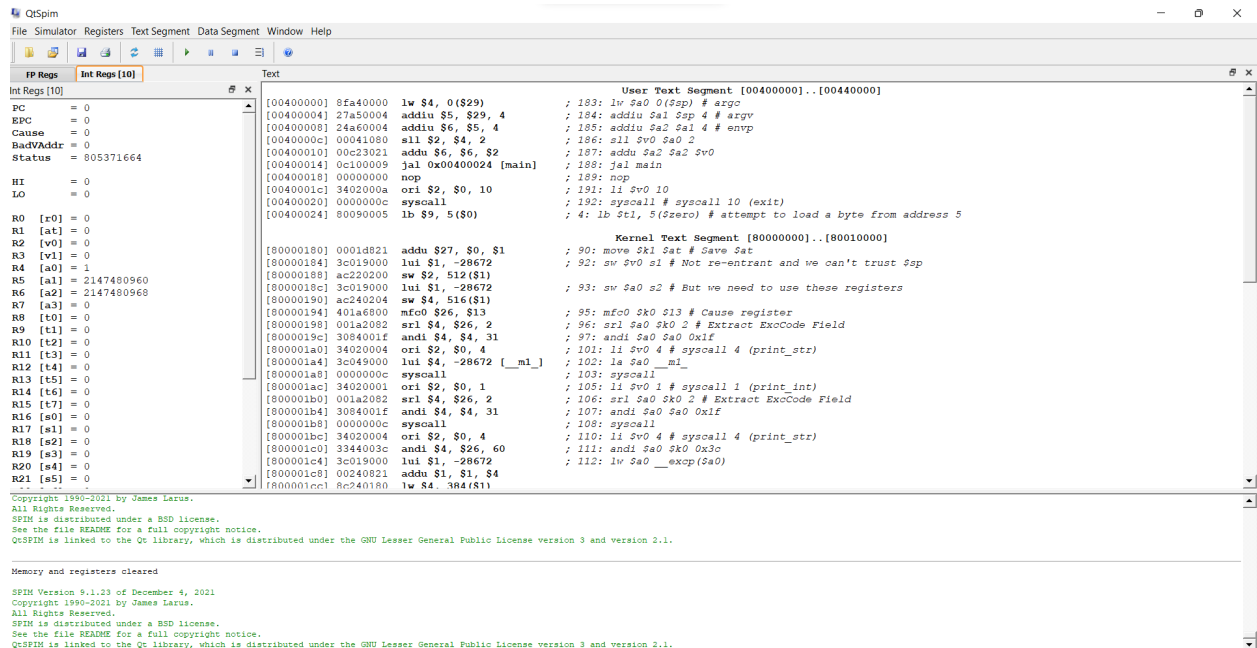
```
.text # tells assembler to switch to the text segment or succeeding lines
contains instructions
```

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

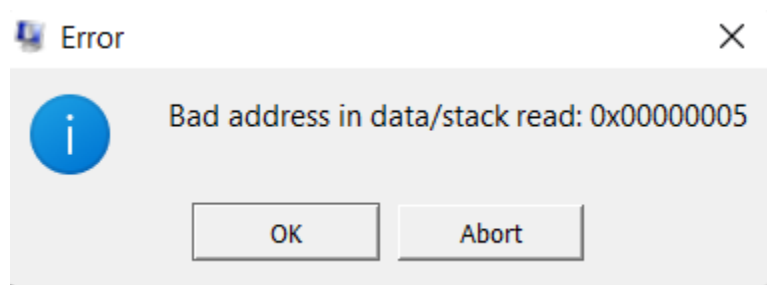
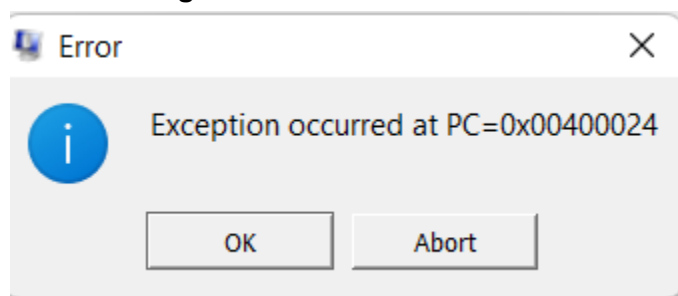
Brief overview of the code section

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

After loading the third file



Error messages



Register values

FP Regs

Int Regs [10]

Int Regs [10]

PC = -2147483264

EPC = 4194340

Cause = 28

BadVAddr = 5

Status = 805371666

HI = 0

LO = 0

R0 [r0] = 0

R1 [at] = 0

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

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.