CMSC 133

Introduction to Computer Organization, Architecture and Assembly Language

Laboratory Report

Names of all Authors and Student IDs	Kienth Matthew B. Tan
Lab No.	1
Task/Step No.	5
Lab Title	MIPS
Email Addresses of all Authors	kbtan@up.edu.ph
Date of Document Issue	05/04/2022
Document Version Number	1.0
Address of Organisation Preparing this report:	Department of Computer Science University of the Philippines Cebu Gorordo Avenue, Lahug, Cebu City Tel: (032) 232 8185
Course Name	CMSC 133 Introduction to Computer Organization, Architecture and Assembly Language
Course Units	3

Summary.

The Laboratory activity is all about putting into action what we learned in our previous lessons and video lectures. It aims to help us learn the basics of MIPS assembly language from basic programs while task by task progressing into more complex problems.

The **Fifth and final task** which will be featured in this laboratory report is on how to reverse a string. To reverse a string, we would have to split the string into two halves, and after that we simply have to load characters from the first and last half of the string and then store it to their counterparts register. We split it into two halves so we can determine when the loop will end. By incrementing “left” or “i” in the first half of the string and decrementing “right” or “j” in the last half of the string, we will be able to access every character within the string. Another additional step in this laboratory activity is to manually write the code in main for the subroutine “reverse_string” compared to the previous laboratory tasks where the main codes were already written. As such we learned to use the instructions “li” or load immediate, “la” or load address, and “syscall” or system call.

In approaching my method to solving this task, I first created a C program that would solve the laboratory problem. We first separated the string into two halves, and through the code we swap the left and right side characters of the string through loading the byte and then storing it. We then proceed to the next element which is the left side character through incrementing it by 1 and to the right side character by decrementing it by 1 until “left” or “i” becomes greater than or equal to “$\frac{length}{2}$”. We check this using “bge” or branch if greater than or equal.

TABLE OF CONTENTS

1 The Lab Problem

The problem to be solved in the fifth and final task is to write a subroutine that reverses a given string called “reverse_string”. When reversing the string, it should modify or overwrite the original memory space of the string. Given the string “MIPS”, it should return “SPIM”. Another additional task in this activity is to manually write the MIPS code in main for the subroutine “reverse_string”.

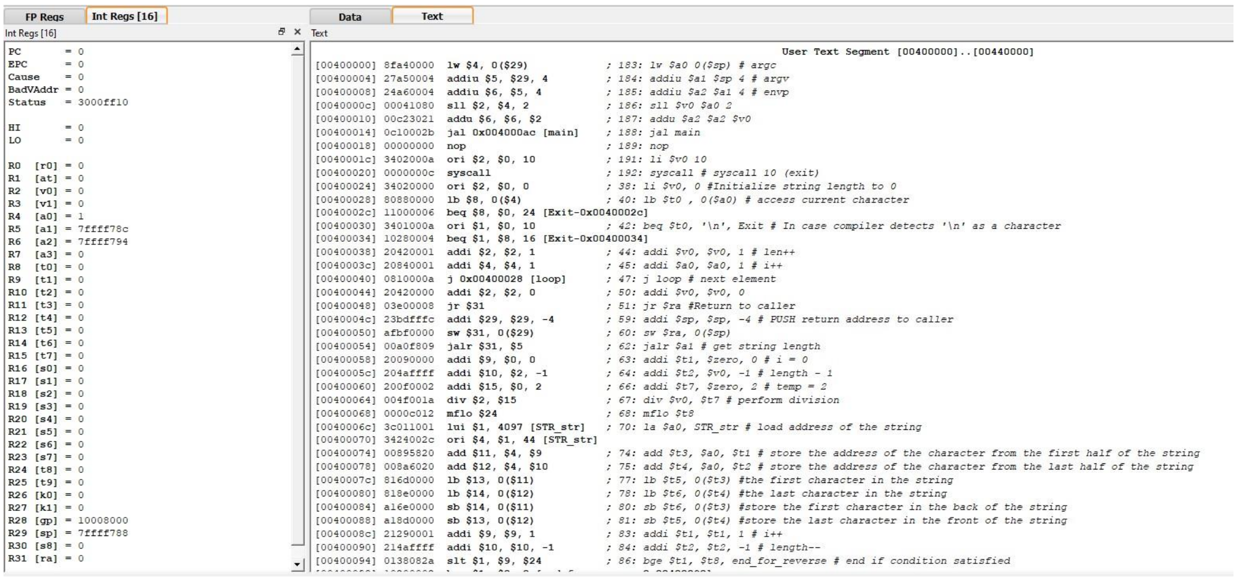
2 The Assembly Source Code

```
reverse_string:
    addi    $sp, $sp, -4        # PUSH return address to caller
    sw      $ra, 0($sp)

    jalr    $a1                 # get string length
    addi    $t1, $zero, 0       # left = 0
    addi    $t2, $v0, -1        # right = length - 1

    addi    $t7, $zero, 2       # divisor = 2
    div     $v0, $t7            # perform division
    mflo    $t8

    la $a0, STR_str             # load address of the string

reverse_loop:

    add     $t3, $a0, $t1       # store the address of the character from the first
half of the string
    add     $t4, $a0, $t2       # store the address of the character from the last
half of the string

    lb      $t5, 0($t3)         #the first character in the string
    lb      $t6, 0($t4)         #the last character in the string

    sb      $t6, 0($t3)         #store the first character in the back of the
string
    sb      $t5, 0($t4)         #store the last character in the front of the
string

    addi    $t1, $t1, 1         # left++
    addi    $t2, $t2, -1        # right--

    bge     $t1, $t8, end_for_reverse   # end if condition satisfied
    j       reverse_loop                # next element

end_for_reverse:
    lw      $ra, 0($sp)         #Pop return address to caller
    addi    $sp, $sp, 4

    jr      $ra                 # Return to caller
```

```
STR_reversed:
    .asciiz "\nstring_reversed(str): "
newline:
    .asciiz "\n"

    .text
    .globl main

main:
    addi    $sp, $sp, -4    # PUSH return address
    sw      $ra, 0($sp)

    la      $a0, STR_str
    jal     print_test_string   #prints string to be reversed

    li      $v0, 4
    la      $a0, STR_reversed   #prints output caption
    syscall

    la      $a0, STR_str        #load address to null terminated string
    la      $a1, string_length  #load address to subroutine string_length

    jal     reverse_string

    la      $a0, STR_str
    jal     print_test_string   #print output


    lw      $ra, 0($sp) # POP return address
    addi    $sp, $sp, 4

    jr      $ra
```

3 Screenshots

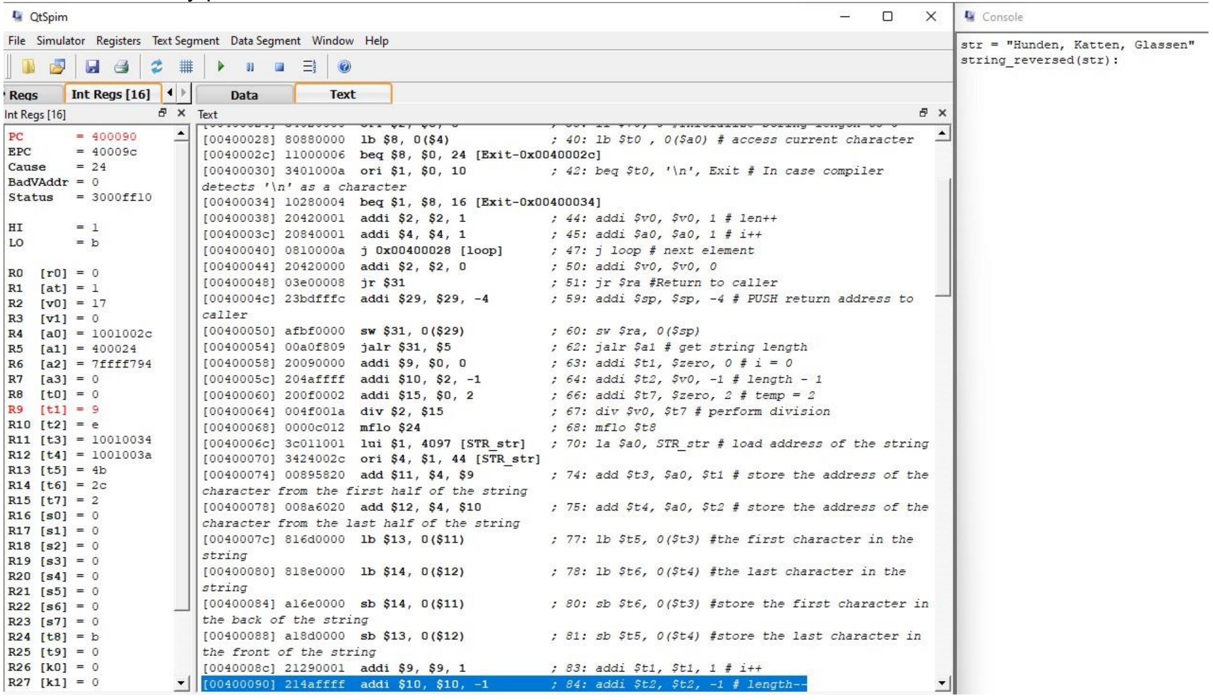
A. **Start of Execution** – Show the current state of the registers, code/program/text segment and data and stack segments in the memory after the program is loaded. Show the initial values of the registers. Show that your program and data has been loaded in the code and the data segments respectively.



```
Int Regs [16]
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff78c
R6  [a2] = 7ffff794
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] = 10008000
R29 [sp] = 7ffff788
R30 [s8] = 0
R31 [ra] = 0
```

```
Text
                                                  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] 0c10002b  jal 0x004000ac [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] 34020000  ori $2, $0, 0            ; 38: li $v0, 0 #Initialize string length to 0
[00400028] 80880000  lb $8, 0($4)             ; 40: lb $t0 , 0($a0) # access current character
[0040002c] 11000006  beq $8, $0, 24 [Exit-0x0040002c]
[00400030] 3401000a  ori $1, $0, 10           ; 42: beq $t0, '\n', Exit # In case compiler detects '\n' as a character
[00400034] 10280004  beq $1, $8, 16 [Exit-0x00400034]
[00400038] 20420001  addi $2, $2, 1           ; 44: addi $v0, $v0, 1 # len++
[0040003c] 20840001  addi $4, $4, 1           ; 45: addi $a0, $a0, 1 # i++
[00400040] 0810000a  j 0x00400028 [loop]      ; 47: j loop # next element
[00400044] 20420000  addi $2, $2, 0           ; 50: addi $v0, $v0, 0
[00400048] 03e00008  jr $31                   ; 51: jr $ra #Return to caller
[0040004c] 23bdfffc  addi $29, $29, -4        ; 59: addi $sp, $sp, -4 # PUSH return address to caller
[00400050] afbf0000  sw $31, 0($29)           ; 60: sw $ra, 0($sp)
[00400054] 00a0f809  jalr $31, $5             ; 62: jalr $a1 # get string length
[00400058] 20090000  addi $9, $0, 0           ; 63: addi $t1, $zero, 0 # i = 0
[0040005c] 204affff  addi $10, $2, -1         ; 64: addi $t2, $v0, -1 # length - 1
[00400060] 200f0002  addi $15, $0, 2          ; 66: addi $t7, $zero, 2 # temp = 2
[00400064] 004f001a  div $2, $15              ; 67: div $v0, $t7 # perform division
[00400068] 0000c012  mflo $24                 ; 68: mflo $t8
[0040006c] 3c011001  lui $1, 4097 [STR_str]   ; 70: la $a0, STR_str # load address of the string
[00400070] 3424002c  ori $4, $1, 44 [STR_str]
[00400074] 00895820  add $11, $4, $9          ; 74: add $t3, $a0, $t1 # store the address of the character from the first half of the string
[00400078] 008a6020  add $12, $4, $10         ; 75: add $t4, $a0, $t2 # store the address of the character from the last half of the string
[0040007c] 816d0000  lb $13, 0($11)           ; 77: lb $t5, 0($t3) #the first character in the string
[00400080] 818e0000  lb $14, 0($12)           ; 78: lb $t6, 0($t4) #the last character in the string
[00400084] a16e0000  sb $14, 0($11)           ; 80: sb $t6, 0($t3) #store the first character in the back of the string
[00400088] a18d0000  sb $13, 0($12)           ; 81: sb $t5, 0($t4) #store the last character in the front of the string
[0040008c] 21290001  addi $9, $9, 1           ; 83: addi $t1, $t1, 1 # i++
[00400090] 214affff  addi $10, $10, -1        ; 84: addi $t2, $t2, -1 # length--
[00400094] 0138082a  slt $1, $9, $24          ; 86: bge $t1, $t8, end_for_reverse # end if condition satisfied
```

B. **Middle of Execution** – Show the current state of the registers and data and stack segments mid-execution. Also show the currently pointed instruction.



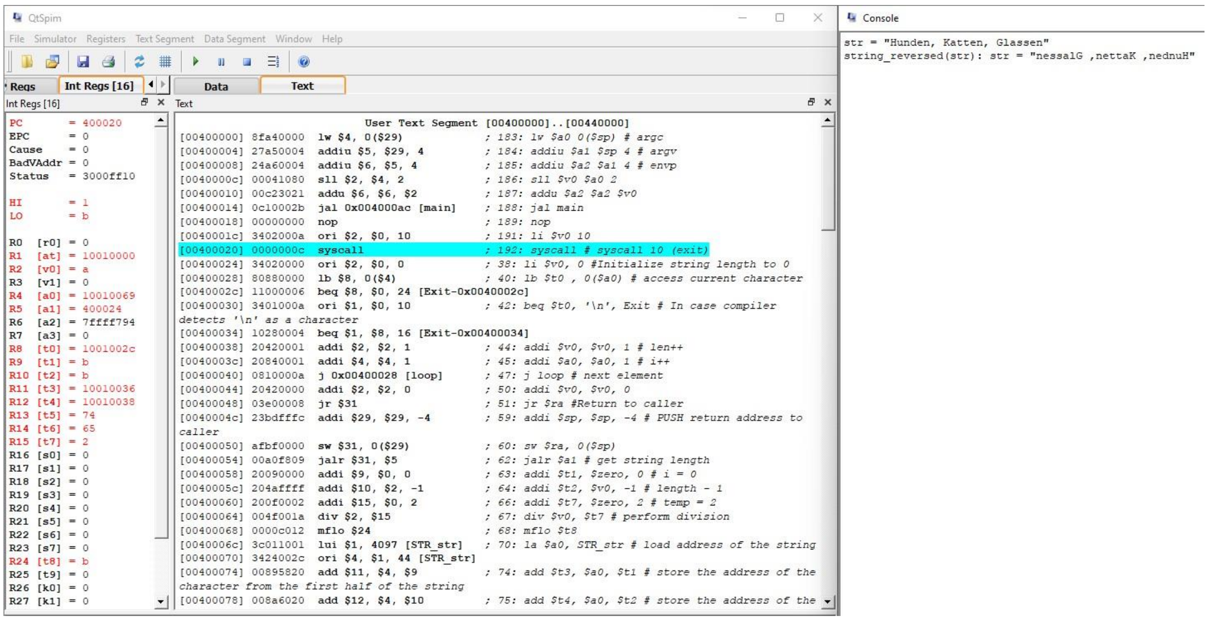
QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Regs
Int Regs [16]
Data
Text
Console

```
Int Regs [16]
PC       = 400090
EPC      = 40009c
Cause    = 24
BadVAddr = 0
Status   = 3000ff10

HI       = 1
LO       = b

R0  [r0] = 0
R1  [at] = 1
R2  [v0] = 17
R3  [v1] = 0
R4  [a0] = 1001002c
R5  [a1] = 400024
R6  [a2] = 7ffff794
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 9
R10 [t2] = e
R11 [t3] = 10010034
R12 [t4] = 1001003a
R13 [t5] = 4b
R14 [t6] = 2c
R15 [t7] = 2
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] = b
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

```
Text
[00400028] 80880000  lb $8, 0($4)             ; 40: lb $t0 , 0($a0) # access current character
[0040002c] 11000006  beq $8, $0, 24 [Exit-0x0040002c]
[00400030] 3401000a  ori $1, $0, 10           ; 42: beq $t0, '\n', Exit # In case compiler
detects '\n' as a character
[00400034] 10280004  beq $1, $8, 16 [Exit-0x00400034]
[00400038] 20420001  addi $2, $2, 1           ; 44: addi $v0, $v0, 1 # len++
[0040003c] 20840001  addi $4, $4, 1           ; 45: addi $a0, $a0, 1 # i++
[00400040] 0810000a  j 0x00400028 [loop]      ; 47: j loop # next element
[00400044] 20420000  addi $2, $2, 0           ; 50: addi $v0, $v0, 0
[00400048] 03e00008  jr $31                   ; 51: jr $ra #Return to caller
[0040004c] 23bdfffc  addi $29, $29, -4        ; 59: addi $sp, $sp, -4 # PUSH return address to
caller
[00400050] afbf0000  sw $31, 0($29)           ; 60: sw $ra, 0($sp)
[00400054] 00a0f809  jalr $31, $5             ; 62: jalr $a1 # get string length
[00400058] 20090000  addi $9, $0, 0           ; 63: addi $t1, $zero, 0 # i = 0
[0040005c] 204affff  addi $10, $2, -1         ; 64: addi $t2, $v0, -1 # length - 1
[00400060] 200f0002  addi $15, $0, 2          ; 66: addi $t7, $zero, 2 # temp = 2
[00400064] 004f001a  div $2, $15              ; 67: div $v0, $t7 # perform division
[00400068] 0000c012  mflo $24                 ; 68: mflo $t8
[0040006c] 3c011001  lui $1, 4097 [STR_str]   ; 70: la $a0, STR_str # load address of the string
[00400070] 3424002c  ori $4, $1, 44 [STR_str]
[00400074] 00895820  add $11, $4, $9          ; 74: add $t3, $a0, $t1 # store the address of the
character from the first half of the string
[00400078] 008a6020  add $12, $4, $10         ; 75: add $t4, $a0, $t2 # store the address of the
character from the last half of the string
[0040007c] 816d0000  lb $13, 0($11)           ; 77: lb $t5, 0($t3) #the first character in the
string
[00400080] 818e0000  lb $14, 0($12)           ; 78: lb $t6, 0($t4) #the last character in the
string
[00400084] a16e0000  sb $14, 0($11)           ; 80: sb $t6, 0($t3) #store the first character in
the back of the string
[00400088] a18d0000  sb $13, 0($12)           ; 81: sb $t5, 0($t4) #store the last character in
the front of the string
[0040008c] 21290001  addi $9, $9, 1           ; 83: addi $t1, $t1, 1 # i++
[00400090] 214affff  addi $10, $10, -1        ; 84: addi $t2, $t2, -1 # length--
```

```
str = "Hunden, Katten, Glassen"
string_reversed(str):
```

C. **End of Execution** – Show the current state of registers and data and stack segments after the execution of the last instruction of your program. Also show the currently pointed instruction after the execution of the last instruction.


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Regs
Int Regs [16]
Data
Text
Int Regs [16]
Text
Console

```
PC       = 400020
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 1
LO       = b

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = a
R3  [v1] = 0
R4  [a0] = 10010069
R5  [a1] = 400024
R6  [a2] = 7ffff794
R7  [a3] = 0
R8  [t0] = 1001002c
R9  [t1] = b
R10 [t2] = b
R11 [t3] = 10010036
R12 [t4] = 10010038
R13 [t5] = 74
R14 [t6] = 65
R15 [t7] = 2
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] = b
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 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] 0c10002b  jal 0x004000ac [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] 34020000  ori $2, $0, 0             ; 38: li $v0, 0 #Initialize string length to 0
[00400028] 80880000  lb $8, 0($4)              ; 40: lb $t0 , 0($a0) # access current character
[0040002c] 11000006  beq $8, $0, 24 [Exit-0x0040002c]
[00400030] 3401000a  ori $1, $0, 10            ; 42: beq $t0, '\n', Exit # In case compiler
detects '\n' as a character
[00400034] 10280004  beq $1, $8, 16 [Exit-0x00400034]
[00400038] 20420001  addi $2, $2, 1            ; 44: addi $v0, $v0, 1 # len++
[0040003c] 20840001  addi $4, $4, 1            ; 45: addi $a0, $a0, 1 # i++
[00400040] 0810000a  j 0x00400028 [loop]       ; 47: j loop # next element
[00400044] 20420000  addi $2, $2, 0            ; 50: addi $v0, $v0, 0
[00400048] 03e00008  jr $31                    ; 51: jr $ra #Return to caller
[0040004c] 23bdfffc  addi $29, $29, -4         ; 59: addi $sp, $sp, -4 # PUSH return address to
caller
[00400050] afbf0000  sw $31, 0($29)            ; 60: sw $ra, 0($sp)
[00400054] 00a0f809  jalr $31, $5              ; 62: jalr $a1 # get string length
[00400058] 20090000  addi $9, $0, 0            ; 63: addi $t1, $zero, 0 # i = 0
[0040005c] 204affff  addi $10, $2, -1          ; 64: addi $t2, $v0, -1 # length - 1
[00400060] 200f0002  addi $15, $0, 2           ; 66: addi $t7, $zero, 2 # temp = 2
[00400064] 004f001a  div $2, $15               ; 67: div $v0, $t7 # perform division
[00400068] 0000c012  mflo $24                  ; 68: mflo $t8
[0040006c] 3c011001  lui $1, 4097 [STR_str]    ; 70: la $a0, STR_str # load address of the string
[00400070] 3424002c  ori $4, $1, 44 [STR_str]
[00400074] 00895820  add $11, $4, $9           ; 74: add $t3, $a0, $t1 # store the address of the
character from the first half of the string
[00400078] 008a6020  add $12, $4, $10          ; 75: add $t4, $a0, $t2 # store the address of the
```

```
str = "Hunden, Katten, Glassen"
string_reversed(str): str = "nessalG ,nettaK ,nednuH"
```

4 Learning & Insights

Through this laboratory activity, I learned how to deal with printing strings in the main function through the use of syscall. Depending on what type of data we want to print, the code that follows \$v0 would change. If it was an integer, \$a0 would contain the integer we want to print and the service code to print it would be "1". To print it, what would follow would be the MIPS code "li \$v0, 1" and then "syscall" at the next line. In our laboratory activity for example, \$a0 would contain the address of a null terminated string and the service code to print it would be "4". An example shown in the activity is in the case of printing the caption for the reversed string. The following code would be "li \$v0, 4", "la \$a0, STR_reversed", "syscall". We first load the service register number in the register \$v0 then we load our argument value which in this case would be "STR_reversed". Finally we use the syscall or system call instruction to retrieve the values.

In the case of the main activity of this task, another important thing I learned is as to how to use division in MIPS and swap the values of registers especially since in this task we have to reverse a given string. To do division in MIPS, we simply have to use the instruction "div" or division by overflow followed by the dividend and the divisor. See how we didn't assign a register as to where the value will be stored, this is because the quotient of the division will be stored in LO and the remainder is stored in HI. LO and HI from what I researched are special registers that store the result of multiplication and division. To access the values within LO and HI, we use the instructions "mfhi" and "mflo", in this case we used "mflo" since we needed to access the quotient of "$\frac{length}{2}$" inside LO and store it into a register. The swapping of values on the other hand is quite easy as we only had to load the first character from the first half of the string and also store the last character from the last half of the string, we then store it into the opposite's registers. Through this method we can reverse the string.

5 Comparison to a High-Level Implementation

Here is an implementation of the lab problem in the language C.

```c
#include <stdio.h>
#include <string.h>

int reverse_string(char str[]){
    char temp;
    int left, right, len;
    len = strlen(str);
    left = 0;
    right = len - 1;
    for (int i = left; i < right; i++){
        temp = str[i];
        str[i] = str[right];
        str[right] = temp;
        right--;
    }
    return 0;
}
int main(){
    char Str[] = "Hunden, Katten, Glassen";
    reverse_string(Str);
    printf("str = %s", Str);
    return 0;
}
```

The most prominent difference here to our MIPS code is at the point where we had to swap values in order to reverse the string. In the case of C, we swap characters by simply storing the first character in one temporary or “temp” variable then proceed with the swap. In MIPS however, we are not using one variable but two as we need to store the loaded values in different registers. In C it is something like “temp = a”, “a = b”, b = temp” and in MIPS it is by first loading the values “reg1 = a”, “reg2 = b”, and then storing it “reg2 = a”, “reg1 = b”.

Another obvious difference is the fact that we didn’t need to create another function to get the string length similar to how we did in MIPS as we can simply use the function “strlen()”.

6 Conclusion

Overall, this laboratory activity helped polish what I learned from the lecture videos provided, it enabled me to understand MIPS assembly by hand in regards to strings, the main function, and in additional new instructions such as “div”, “mflo”, “bge”, and in the main function which is “la”, “li”, and syscall. Similar to other programming languages, I realized that it is important to understand MIPS Assembly as it will assist me in future projects especially those that include the manipulation of strings and overwriting values in memory addresses where in this task we had to reverse a given string. We swap characters through registers using the instruction “lb” or load byte and “sb” or store byte, and when we print out the string, what comes out is its reverse.