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.	3
Lab Title	MIPS
Email Addresses of all Authors	kbtan@up.edu.ph
Date of Document Issue	04/06/2022
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 **third task** which will be featured in this laboratory report is on how to call a subroutine within another subroutine, which in this case would be for every character in a string, we would call that subroutine. The task is actually quite simple as we simply had to loop through all the characters in that string until we detect a NULL and for every character we call the subroutine. Through this task, I learned through practice on how to use a very useful instruction which is “jalr” or jump-and-link-register. The task also taught me how to specifically use stack pointers through the instructions “sw” and lw”.

In approaching my method to solving this task, I first created a C program that would solve the laboratory problem. I then translated most of my C code into MIPS, I said most as some additional lines of code are needed for the function to work such as the use of sw and lw for stacks and lb for loading that specific character. After that, the steps are quite simple as I just had to loop the code up until it detects a terminating NULL, and for each loop we call the callback subroutine using jalr.

TABLE OF CONTENTS

1 The Lab Problem

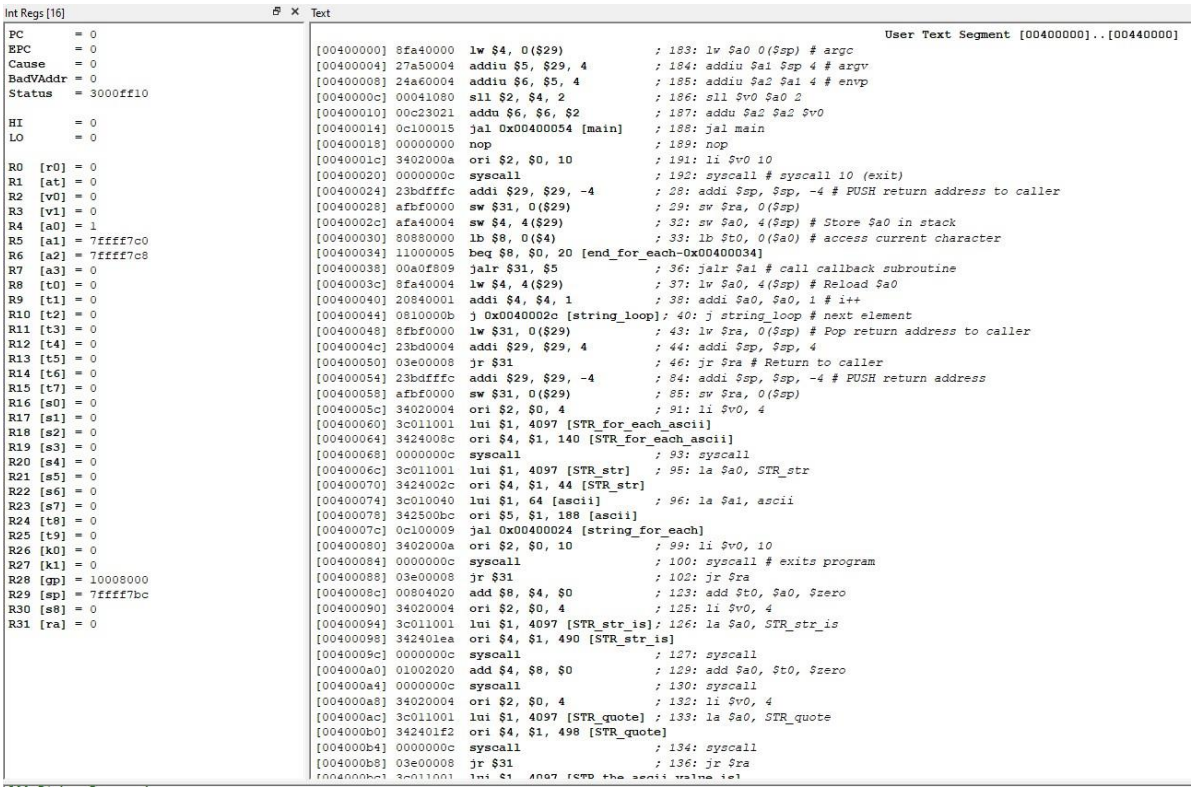
The problem to be solved in this third task of the laboratory activity is to create a subroutine wherein it takes the memory address of a certain string and another memory address for another subroutine. It should loop through all the characters in that string until it detects a NULL and for every letter in that string, it then calls that subroutine which in this case would be “print_test_string”. This subroutine that is being called would return the ASCII value of that character.

2 The Assembly Source Code

```
###############################################################################
#
#  DESCRIPTION: For each of the characters in a string (from left to right),
#       call a callback subroutine.
#
#       The callback suboutine will be called with the address of
#           the character as the input parameter ($a0).
#
#        INPUT: $a0 - address to a NUL terminated string.
#
#       $a1 - address to a callback subroutine.
#
###############################################################################
string_for_each:

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

string_loop:
    sw      $a0, 4($sp)           # Store $a0 in stack
    lb      $t0, 0($a0)           # access current character
    beqz    $t0, end_for_each     # check for null character

    jalr    $a1                   # call callback subroutine
    lw      $a0, 4($sp)           # Reload $a0
    addi    $a0, $a0, 1           # i++

    j       string_loop           # next element

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

    jr      $ra                   # Return to caller

###############################################################################
```

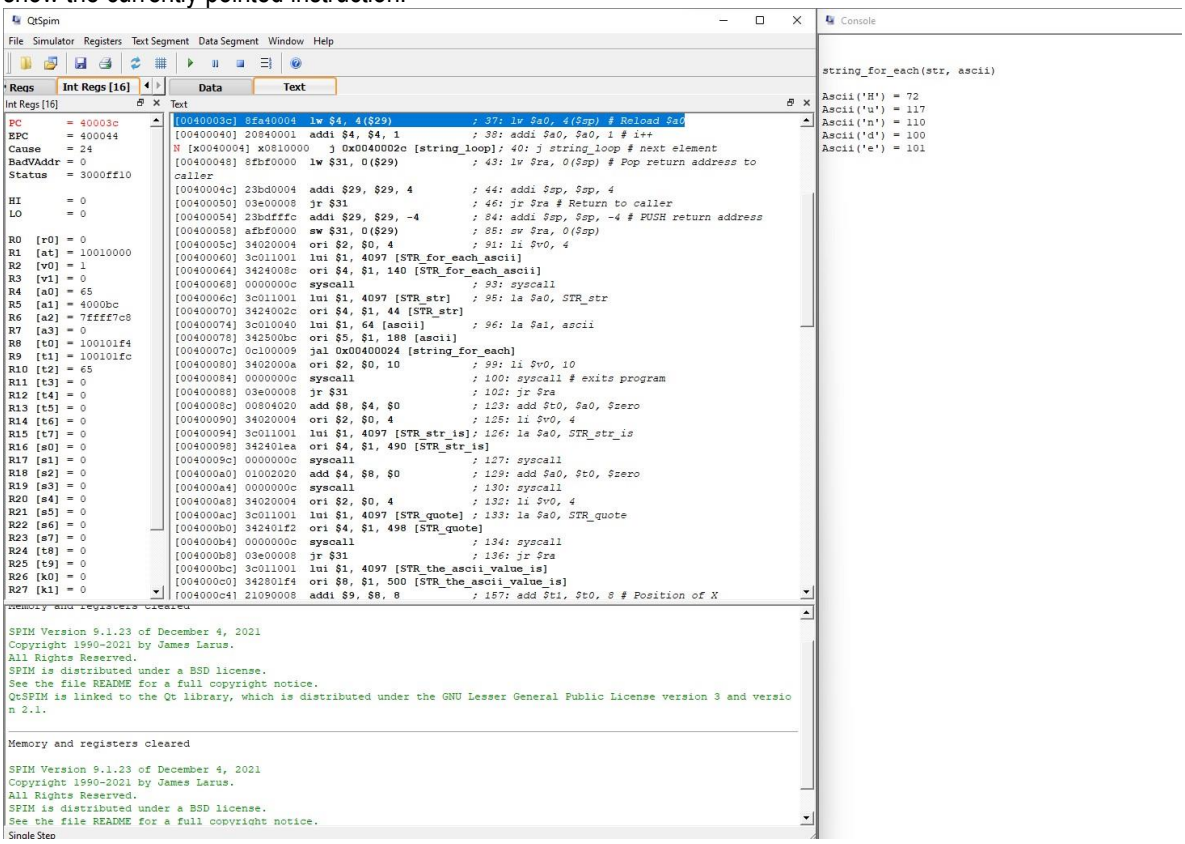
3 Screenshots

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.



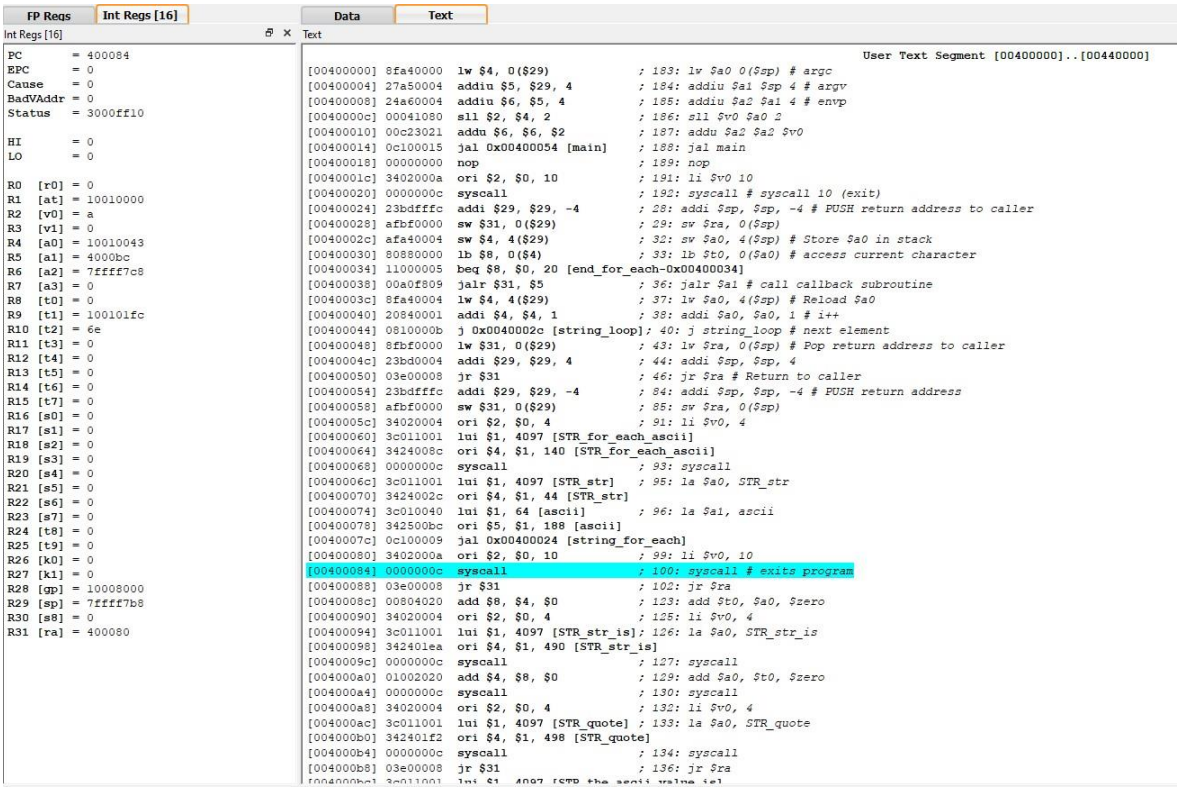
```
Int Regs [16]
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff7c0
R6  [a2] = 7ffff7c8
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] = 7ffff7bc
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] 0c100015  jal 0x00400054 [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] 23bdfffc  addi $29, $29, -4        ; 28: addi $sp, $sp, -4 # PUSH return address to caller
[00400028] afbf0000  sw $31, 0($29)           ; 29: sw $ra, 0($sp)
[0040002c] afa40004  sw $4, 4($29)            ; 32: sw $a0, 4($sp) # Store $a0 in stack
[00400030] 80880000  lb $8, 0($4)             ; 33: lb $t0, 0($a0) # access current character
[00400034] 11000005  beq $8, $0, 20 [end_for_each-0x00400034]
[00400038] 00a0f809  jalr $31, $5             ; 36: jalr $a1 # call callback subroutine
[0040003c] 8fa40004  lw $4, 4($29)            ; 37: lw $a0, 4($sp) # Reload $a0
[00400040] 20840001  addi $4, $4, 1           ; 38: addi $a0, $a0, 1 # i++
[00400044] 0810000b  j 0x0040002c [string_loop]; 40: j string_loop # next element
[00400048] 8fbf0000  lw $31, 0($29)           ; 43: lw $ra, 0($sp) # Pop return address to caller
[0040004c] 23bd0004  addi $29, $29, 4         ; 44: addi $sp, $sp, 4
[00400050] 03e00008  jr $31                   ; 46: jr $ra # Return to caller
[00400054] 23bdfffc  addi $29, $29, -4        ; 84: addi $sp, $sp, -4 # PUSH return address
[00400058] afbf0000  sw $31, 0($29)           ; 85: sw $ra, 0($sp)
[0040005c] 34020004  ori $2, $0, 4            ; 91: li $v0, 4
[00400060] 3c011001  lui $1, 4097 [STR_for_each_ascii]
[00400064] 3424008c  ori $4, $1, 140 [STR_for_each_ascii]
[00400068] 0000000c  syscall                  ; 93: syscall
[0040006c] 3c011001  lui $1, 4097 [STR_str]   ; 95: la $a0, STR_str
[00400070] 3424002c  ori $4, $1, 44 [STR_str]
[00400074] 3c010040  lui $1, 64 [ascii]       ; 96: la $a1, ascii
[00400078] 342500bc  ori $5, $1, 188 [ascii]
[0040007c] 0c100009  jal 0x00400024 [string_for_each]
[00400080] 3402000a  ori $2, $0, 10           ; 99: li $v0, 10
[00400084] 0000000c  syscall                  ; 100: syscall # exits program
[00400088] 03e00008  jr $31                   ; 102: jr $ra
[0040008c] 00804020  add $8, $4, $0           ; 123: add $t0, $a0, $zero
[00400090] 34020004  ori $2, $0, 4            ; 125: li $v0, 4
[00400094] 3c011001  lui $1, 4097 [STR_str_is]; 126: la $a0, STR_str_is
[00400098] 342401ea  ori $4, $1, 490 [STR_str_is]
[0040009c] 0000000c  syscall                  ; 127: syscall
[004000a0] 01002020  add $4, $8, $0           ; 129: add $a0, $t0, $zero
[004000a4] 0000000c  syscall                  ; 130: syscall
[004000a8] 34020004  ori $2, $0, 4            ; 132: li $v0, 4
[004000ac] 3c011001  lui $1, 4097 [STR_quote] ; 133: la $a0, STR_quote
[004000b0] 342401f2  ori $4, $1, 498 [STR_quote]
[004000b4] 0000000c  syscall                  ; 134: syscall
[004000b8] 03e00008  jr $31                   ; 136: jr $ra
```

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

Int Regs [16]
PC       = 40003c
EPC      = 400044
Cause    = 24
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = 1
R3  [v1] = 0
R4  [a0] = 65
R5  [a1] = 4000bc
R6  [a2] = 7ffff7c8
R7  [a3] = 0
R8  [t0] = 100101f4
R9  [t1] = 100101fc
R10 [t2] = 65
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

Text
[0040003c] 8fa40004  lw $4, 4($29)            ; 37: lw $a0, 4($sp) # Reload $a0
[00400040] 20840001  addi $4, $4, 1           ; 38: addi $a0, $a0, 1 # i++
N [x0040004] x0810000   j 0x0040002c [string_loop]; 40: j string_loop # next element
[00400048] 8fbf0000  lw $31, 0($29)           ; 43: lw $ra, 0($sp) # Pop return address to
caller
[0040004c] 23bd0004  addi $29, $29, 4         ; 44: addi $sp, $sp, 4
[00400050] 03e00008  jr $31                   ; 46: jr $ra # Return to caller
[00400054] 23bdfffc  addi $29, $29, -4        ; 84: addi $sp, $sp, -4 # PUSH return address
[00400058] afbf0000  sw $31, 0($29)           ; 85: sw $ra, 0($sp)
[0040005c] 34020004  ori $2, $0, 4            ; 91: li $v0, 4
[00400060] 3c011001  lui $1, 4097 [STR_for_each_ascii]
[00400064] 3424008c  ori $4, $1, 140 [STR_for_each_ascii]
[00400068] 0000000c  syscall                  ; 93: syscall
[0040006c] 3c011001  lui $1, 4097 [STR_str]   ; 95: la $a0, STR_str
[00400070] 3424002c  ori $4, $1, 44 [STR_str]
[00400074] 3c010040  lui $1, 64 [ascii]       ; 96: la $a1, ascii
[00400078] 342500bc  ori $5, $1, 188 [ascii]
[0040007c] 0c100009  jal 0x00400024 [string_for_each]
[00400080] 3402000a  ori $2, $0, 10           ; 99: li $v0, 10
[00400084] 0000000c  syscall                  ; 100: syscall # exits program
[00400088] 03e00008  jr $31                   ; 102: jr $ra
[0040008c] 00804020  add $8, $4, $0           ; 123: add $t0, $a0, $zero
[00400090] 34020004  ori $2, $0, 4            ; 125: li $v0, 4
[00400094] 3c011001  lui $1, 4097 [STR_str_is]; 126: la $a0, STR_str_is
[00400098] 342401ea  ori $4, $1, 490 [STR_str_is]
[0040009c] 0000000c  syscall                  ; 127: syscall
[004000a0] 01002020  add $4, $8, $0           ; 129: add $a0, $t0, $zero
[004000a4] 0000000c  syscall                  ; 130: syscall
[004000a8] 34020004  ori $2, $0, 4            ; 132: li $v0, 4
[004000ac] 3c011001  lui $1, 4097 [STR_quote] ; 133: la $a0, STR_quote
[004000b0] 342401f2  ori $4, $1, 498 [STR_quote]
[004000b4] 0000000c  syscall                  ; 134: syscall
[004000b8] 03e00008  jr $31                   ; 136: jr $ra
[004000bc] 3c011001  lui $1, 4097 [STR_the_ascii_value_is]
[004000c0] 342801f4  ori $8, $1, 500 [STR_the_ascii_value_is]
[004000c4] 21090008  addi $9, $8, 8           ; 157: add $t1, $t0, 8 # Position of X

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 versio
n 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.
Single Step

Console

string_for_each(str, ascii)

Ascii('H') = 72
Ascii('u') = 117
Ascii('n') = 110
Ascii('d') = 100
Ascii('e') = 101
```

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.



FP Regs | Int Regs [16]

Int Regs [16]

```
PC       = 400084
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = a
R3  [v1] = 0
R4  [a0] = 10010043
R5  [a1] = 4000bc
R6  [a2] = 7ffff7c8
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 100101fc
R10 [t2] = 6e
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] = 7ffff7b8
R30 [s8] = 0
R31 [ra] = 400080
```

Data | Text

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] 0c100015  jal 0x00400054 [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] 23bdfffc  addi $29, $29, -4        ; 28: addi $sp, $sp, -4 # PUSH return address to caller
[00400028] afbf0000  sw $31, 0($29)           ; 29: sw $ra, 0($sp)
[0040002c] afa40004  sw $4, 4($29)            ; 32: sw $a0, 4($sp) # Store $a0 in stack
[00400030] 80880000  lb $8, 0($4)             ; 33: lb $t0, 0($a0) # access current character
[00400034] 11000005  beq $8, $0, 20 [end_for_each-0x00400034]
[00400038] 00a0f809  jalr $31, $5             ; 36: jalr $a1 # call callback subroutine
[0040003c] 8fa40004  lw $4, 4($29)            ; 37: lw $a0, 4($sp) # Reload $a0
[00400040] 20840001  addi $4, $4, 1           ; 38: addi $a0, $a0, 1 # i++
[00400044] 0810000b  j 0x0040002c [string_loop]; 40: j string_loop # next element
[00400048] 8fbf0000  lw $31, 0($29)           ; 43: lw $ra, 0($sp) # Pop return address to caller
[0040004c] 23bd0004  addi $29, $29, 4         ; 44: addi $sp, $sp, 4
[00400050] 03e00008  jr $31                   ; 46: jr $ra # Return to caller
[00400054] 23bdfffc  addi $29, $29, -4        ; 84: addi $sp, $sp, -4 # PUSH return address
[00400058] afbf0000  sw $31, 0($29)           ; 85: sw $ra, 0($sp)
[0040005c] 34020004  ori $2, $0, 4            ; 91: li $v0, 4
[00400060] 3c011001  lui $1, 4097 [STR_for_each_ascii]
[00400064] 3424008c  ori $4, $1, 140 [STR_for_each_ascii]
[00400068] 0000000c  syscall                  ; 93: syscall
[0040006c] 3c011001  lui $1, 4097 [STR_str]   ; 95: la $a0, STR_str
[00400070] 3424002c  ori $4, $1, 44 [STR_str]
[00400074] 3c010040  lui $1, 64 [ascii]       ; 96: la $a1, ascii
[00400078] 342500bc  ori $5, $1, 188 [ascii]
[0040007c] 0c100009  jal 0x00400024 [string_for_each]
[00400080] 3402000a  ori $2, $0, 10           ; 99: li $v0, 10
[00400084] 0000000c  syscall                  ; 100: syscall # exits program
[00400088] 03e00008  jr $31                   ; 102: jr $ra
[0040008c] 00804020  add $8, $4, $0           ; 123: add $t0, $a0, $zero
[00400090] 34020004  ori $2, $0, 4            ; 125: li $v0, 4
[00400094] 3c011001  lui $1, 4097 [STR_str_is]; 126: la $a0, STR_str_is
[00400098] 342401ea  ori $4, $1, 490 [STR_str_is]
[0040009c] 0000000c  syscall                  ; 127: syscall
[004000a0] 01002020  add $4, $8, $0           ; 129: add $a0, $t0, $zero
[004000a4] 0000000c  syscall                  ; 130: syscall
[004000a8] 34020004  ori $2, $0, 4            ; 132: li $v0, 4
[004000ac] 3c011001  lui $1, 4097 [STR_quote] ; 133: la $a0, STR_quote
[004000b0] 342401f2  ori $4, $1, 498 [STR_quote]
[004000b4] 0000000c  syscall                  ; 134: syscall
[004000b8] 03e00008  jr $31                   ; 136: jr $ra
```

4 Learning & Insights

Through this laboratory activity, I learned how to deal with calling subroutines within another subroutine. In this case I had to use an instruction that I haven't used before which is the "jalr" instruction or the jump-and-link-register instruction. It kind of acts as the combination between jal and jr where it transfer control to the address in a specified register which in this case would be the callback subroutine $a1 and it then stores the return address in the register file allowing us to return to the first subroutine after the callback subroutine finishes.

This particular task also helped me understand through practical implementation on how "sw" and "lw" works especially in this case where we had to access specific characters in that string for every loop as an input to the callback subroutine. We need to store the address of the string in the stack for every loop because if we don't do so then the only the first character of the string would return an ASCII value, this is because $a0 gets overwritten in the print_test_string subroutine and thus the second loop through the string will not work. We first store the address of the string, do the callback subroutine, and then load back the address of the string from the stack into $a0 in preparation for the next loop, we do this until we loop through all the characters in 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>

int print_test_string(char str[]){
    printf("Ascii('%c') = %d\n", str, str);
    return 0;
}

int string_for_each(char str[]){

    for (int i = 0; str[i] != '\0'; ++i){
        print_test_string(str[i]);
    }
    return 0;
}


int main(){
    char Str[] = "Hunden, Katten, Glassen";
    printf("str = %s\n", Str);
    string_for_each(Str);
    return 0;
}
```

Similar to our assembly source code, the main acts as the caller and the function sum acts as the callee. In this code, string_for_each acts as the main subroutine that is called by main and print_test_string is the callback subroutine within the called by the first subroutine. In terms of the subroutine string_for_each, compared to MIPS where we had to store and load the word in every loop, in C we didn't need to specifically do so. In calling the subroutine, we translated "print_test_string(str[i]) in the for-loop of the function string_for_each into "jalr $a1" in MIPS in which $a1 contains the address of print_test_string.

In terms of printing the string in its ASCII value in C, we simply had to identify what we needed to be printed out from that string, which in this case would be "%d" instead of "%c" as we didn't need the character but the value of that character to be printed out.

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, its ASCII value, and in how to use "jalr" for callback subroutines. 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 includes the specific use of addresses, memories and registers. This particular task helped in discovering how the program works around the use of subroutines within another subroutine which in this case I did through the use of "jalr" and in how to remedy cases wherein an address possibly gets overwritten through the use of "sw" and "lw".