

# 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.	2
Lab Title	MIPS
Email Addresses of all Authors	kbtan@up.edu.ph
Date of Document Issue	03/30/2022
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 **second task** which will be featured in this laboratory report is on how to find the length of a string. The task is actually quite simple as we simply had to loop through the string to get the length. Through this task, I learned through practice on how to use numerous other instructions such as lb, li, and beqz. The task also taught me how to access the next character in a string.

In approaching my method to solving this task, I first created a C program that would solve the laboratory problem. Things would then be much simpler after that as I only had to translate my C code into MIPS assembly code. I then tried to apply what I learned on how to indicate when the loop ends. 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 add the length by 1.

## TABLE OF CONTENTS

1	THE LAB PROBLEM.....	4
2	THE ASSEMBLY SOURCE CODE.....	4
3	SCREENSHOTS .....	5
4	LEARNING & INSIGHTS.....	6
5	COMPARISON TO A HIGH-LEVEL IMPLEMENTATION .....	6
6	CONCLUSION.....	7

## 1 The Lab Problem

The problem to be solved in this second task of the laboratory activity is to create a subroutine wherein it has to take the memory address of a string and then return the number of characters in that given string. This means that through an iteration or loop, it keeps count of the number of characters in the string until it detects an exit code or a terminating NULL at the end of the string. The length of the string would then be returned to the caller to be printed in the console.

## 2 The Assembly Source Code

```
#####
#
# DESCRIPTION: Gives the length of a string.
#
#     INPUT: $a0 - address to a NUL terminated string.
#
#     OUTPUT: $v0 - length of the string (NUL excluded).
#
#     EXAMPLE:  string_length("abcdef") == 6.
#
#####
string_length:
    ##### Write your solution here #####
    li $v0, 0           #Initialize string length to 0
loop:
    lb  $t0, 0($a0)      # access current character
    beqz $t0, Exit       # check for null character
    beq $t0, '\n', Exit  # In case compiler detects '\n' as a character

    addi $v0, $v0, 1     # len++
    addi $a0, $a0, 1     # i++

    j    loop           # next element

Exit:
    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	Text	User Text Segment [00400000]..[00440000]
PC = 0	00400000	8fa40000 lw \$4, 0(\$29)	: 183: lw \$a0 0(\$sp) # argc
EPC = 0	00400004	27a50004 addiu \$5, \$29, 4	: 184: addiu \$a1 \$sp 4 # argv
Cause = 0	00400008	24a60004 addiu \$6, \$5, 4	: 185: addiu \$a2 \$a1 4 # envp
BadVAddr = 0	0040000c	00041080 sll \$2, \$4, 2	: 186: sll \$v0 \$a0 2
Status = 3000fff10	00400010	00c23021 addu \$6, \$6, \$2	: 187: addu \$a2 \$a2 \$v0
HI = 0	00400014	0c10001b jal 0x0040006c [main]	: 188: jal main
LO = 0	00400018	00000000 nop	: 189: nop
R0 [r0] = 0	0040001c	3402000a ori \$2, \$0, 10	: 191: li \$v0 10
R1 [at] = 0	00400020	0000000c syscall	: 192: syscall # syscall 10 (exit)
R2 [v0] = 0	00400024	20020000 addi \$2, \$0, 0	: 40: addi \$v0, \$zero, 0 # Initialize Sum to zero.
R3 [v1] = 0	00400028	00004020 add \$8, \$0, \$0	: 41: add \$t0, \$zero, \$zero # Initialize array index i to zero.
R4 [a0] = 1	0040002c	03e00008 jr \$31	: 57: jr \$ra # Return to caller.
R5 [a1] = 7ffff758	00400030	34020000 ori \$2, \$0, 0	: 71: li \$v0, 0 # Initialize string length to 0
R6 [a2] = 7ffff760	00400034	80880000 lb \$8, 0(\$4)	: 73: lb \$t0, 0(\$a0) # access current character
R7 [a3] = 0	00400038	11000006 beq \$8, \$0, 24 [Exit-0x00400038]	: 75: beq \$t0, '\n', Exit # In case compiler detects '\n' as a character
R8 [t0] = 0	0040003c	3401000a ori \$1, \$0, 10	: 76: beq \$t0, '\n', Exit # In case compiler detects '\n' as a character
R9 [t1] = 0	00400040	10280004 beq \$1, \$8, 16 [Exit-0x00400040]	: 77: beq \$t0, '\n', Exit # In case compiler detects '\n' as a character
R10 [t2] = 0	00400044	20420001 addi \$2, \$2, 1	: 77: addi \$v0, \$v0, 1 # len++
R11 [t3] = 0	00400048	20840001 addi \$4, \$4, 1	: 78: addi \$a0, \$a0, 1 # i++
R12 [t4] = 0	0040004c	0810000d j 0x00400034 [loop]	: 80: j loop # next element
R13 [t5] = 0	00400050	03e00008 jr \$31	: 83: jr \$ra # Return to caller
R14 [t6] = 0	00400054	23bdfcfc addi \$29, \$29, -4	: 100: addi \$sp, \$sp, -4 # PUSH return address to caller
R15 [t7] = 0	00400058	afbf0000 sw \$31, 0(\$29)	: 101: sw \$ra, 0(\$sp)
R16 [s0] = 0	0040005c	8fbf0000 lw \$31, 0(\$29)	: 105: lw \$ra, 0(\$sp) # Pop return address to caller
R17 [s1] = 0	00400060	23bd0004 addi \$29, \$29, 4	: 106: addi \$sp, \$sp, 4
R18 [s2] = 0	00400064	03e00008 jr \$31	: 108: jr \$ra
R19 [s3] = 0	00400068	03e00008 jr \$31	: 121: jr \$ra
R20 [s4] = 0	0040006c	23bdfcfc addi \$29, \$29, -4	: 166: addi \$sp, \$sp, -4 # PUSH return address
R21 [s5] = 0	00400070	afbf0000 sw \$31, 0(\$29)	: 167: sw \$ra, 0(\$sp)
R22 [s6] = 0	00400074	3402000a ori \$2, \$0, 4	: 173: li \$v0, 4
R23 [s7] = 0	00400078	3c011001 lui \$1, 4097 [STR_sum_of_fibonacci_a]	
R24 [t8] = 0	0040007c	34240047 ori \$4, \$1, 71 [STR_sum_of_fibonacci_a]	
R25 [t9] = 0	00400080	0000000c syscall	: 175: syscall
R26 [k0] = 0	00400084	3c011001 lui \$1, 4097	: 177: lw \$a0, ARRAY_SIZE
R27 [k1] = 0	00400088	8c240000 lw \$4, 0(\$1)	
R28 [gp] = 10008000	0040008c	34020001 ori \$2, \$0, 1	: 178: li \$v0, 1
R29 [sp] = 7ffff754	00400090	0000000c syscall	: 179: syscall
R30 [s8] = 0	00400094	3402000a ori \$2, \$0, 4	: 181: li \$v0, 4
R31 [ra] = 0	00400098	3c011001 lui \$1, 4097 [STR_sum_of_fibonacci_b]	
	0040009c	34240057 ori \$4, \$1, 87 [STR_sum_of_fibonacci_b]	
	004000a0	0000000c syscall	: 183: syscall
	004000a4	3c011001 lui \$1, 4097 [FIBONACCI_ARRAY]	: 185: la \$a0, FIBONACCI_ARRAY
	004000a8	34240004 ori \$4, \$1, 4 [FIBONACCI_ARRAY]	
	004000ac	3c011001 lui \$1, 4097	: 186: lw \$a1, ARRAY_SIZE
	004000b0	8c250000 lw \$5, 0(\$1)	

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

Int Regs [16]	PC	Text	User Text Segment [00400000]..[00440000]
PC = 400048	00400000	8fa40000 lw \$4, 0(\$29)	: 183: lw \$a0 0(\$sp) # argc
EPC = 4000f8	00400004	27a50004 addiu \$5, \$29, 4	: 184: addiu \$a1 \$sp 4 # argv
Cause = 24	00400008	24a60004 addiu \$6, \$5, 4	: 185: addiu \$a2 \$a1 4 # envp
BadVAddr = 0	0040000c	00041080 sll \$2, \$4, 2	: 186: sll \$v0 \$a0 2
Status = 3000fff10	00400010	00c23021 addu \$6, \$6, \$2	: 187: addu \$a2 \$a2 \$v0
HI = 0	00400014	0c10001b jal 0x0040006c [main]	: 188: jal main
LO = 0	00400018	00000000 nop	: 189: nop
R0 [r0] = 0	0040001c	3402000a ori \$2, \$0, 10	: 191: li \$v0 10
R1 [at] = a	00400020	0000000c syscall	: 192: syscall # syscall 10 (exit)
R2 [v0] = 8	00400024	20020000 addi \$2, \$0, 0	: 40: addi \$v0, \$zero, 0 # Initialize Sum to zero.
R3 [v1] = 0	00400028	00004020 add \$8, \$0, \$0	: 41: add \$t0, \$zero, \$zero # Initialize array index i to zero.
R4 [a0] = 10010033	0040002c	03e00008 jr \$31	: 57: jr \$ra # Return to caller.
R5 [a1] = a	00400030	34020000 ori \$2, \$0, 0	: 71: li \$v0, 0 # Initialize string length to 0
R6 [a2] = 7ffff760	00400034	80880000 lb \$8, 0(\$4)	: 73: lb \$t0, 0(\$a0) # access current character
R7 [a3] = 0	00400038	11000006 beq \$8, \$0, 24 [Exit-0x00400038]	: 75: beq \$t0, '\n', Exit # In case compiler detects '\n' as a character
R8 [t0] = 20	0040003c	3401000a ori \$1, \$0, 10	: 76: beq \$t0, '\n', Exit # In case compiler detects '\n' as a character
R9 [t1] = 0	00400040	10280004 beq \$1, \$8, 16 [Exit-0x00400040]	: 77: beq \$t0, '\n', Exit # In case compiler detects '\n' as a character
R10 [t2] = 0	00400044	20420001 addi \$2, \$2, 1	: 77: addi \$v0, \$v0, 1 # len++
R11 [t3] = 0	00400048	20840001 addi \$4, \$4, 1	: 78: addi \$a0, \$a0, 1 # i++
R12 [t4] = 0	0040004c	0810000d j 0x00400034 [loop]	: 80: j loop # next element
R13 [t5] = 0	00400050	03e00008 jr \$31	: 83: jr \$ra # Return to caller
R14 [t6] = 0	00400054	23bdfcfc addi \$29, \$29, -4	: 100: addi \$sp, \$sp, -4 # PUSH return address to caller
R15 [t7] = 0	00400058	afbf0000 sw \$31, 0(\$29)	: 101: sw \$ra, 0(\$sp)
R16 [s0] = 0	0040005c	8fbf0000 lw \$31, 0(\$29)	: 105: lw \$ra, 0(\$sp) # Pop return address to caller
R17 [s1] = 0	00400060	23bd0004 addi \$29, \$29, 4	: 106: addi \$sp, \$sp, 4
R18 [s2] = 0	00400064	03e00008 jr \$31	: 108: jr \$ra
R19 [s3] = 0	00400068	03e00008 jr \$31	: 121: jr \$ra
R20 [s4] = 0	0040006c	23bdfcfc addi \$29, \$29, -4	: 166: addi \$sp, \$sp, -4 # PUSH return address
R21 [s5] = 0	00400070	afbf0000 sw \$31, 0(\$29)	: 167: sw \$ra, 0(\$sp)
R22 [s6] = 0	00400074	3402000a ori \$2, \$0, 4	: 173: li \$v0, 4
R23 [s7] = 0	00400078	3c011001 lui \$1, 4097 [STR_sum_of_fibonacci_a]	
R24 [t8] = 0	0040007c	34240047 ori \$4, \$1, 71 [STR_sum_of_fibonacci_a]	
R25 [t9] = 0	00400080	0000000c syscall	: 175: syscall
R26 [k0] = 0	00400084	3c011001 lui \$1, 4097	: 177: lw \$a0, ARRAY_SIZE
R27 [k1] = 0	00400088	8c240000 lw \$4, 0(\$1)	
R28 [gp] = 0	0040008c	34020001 ori \$2, \$0, 1	: 178: li \$v0, 1
R29 [sp] = 7ffff750	00400090	0000000c syscall	: 179: syscall
R30 [s8] = 0	00400094	3402000a ori \$2, \$0, 4	: 181: li \$v0, 4
R31 [ra] = 4000fc	00400098	3c011001 lui \$1, 4097 [STR_sum_of_fibonacci_b]	
	0040009c	34240057 ori \$4, \$1, 87 [STR_sum_of_fibonacci_b]	
	004000a0	0000000c syscall	: 183: syscall
	004000a4	3c011001 lui \$1, 4097 [FIBONACCI_ARRAY]	: 185: la \$a0, FIBONACCI_ARRAY
	004000a8	34240004 ori \$4, \$1, 4 [FIBONACCI_ARRAY]	
	004000ac	3c011001 lui \$1, 4097	: 186: lw \$a1, ARRAY_SIZE
	004000b0	8c250000 lw \$5, 0(\$1)	
	004000b4	0c100009 jal 0x00400024 [integer_array_sum]	
	004000b8	add \$4, \$2, \$0	: 190: add \$a0, \$v0, \$zero

- 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.

Int Regs [16]	Text
PC = 400020 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] = 100100d6 R5 [a1] = 400068 R6 [a2] = 7ffff760 R7 [a3] = 0 R8 [t0] = 1001002c 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 R22 [s6] = 0 R23 [s7] = 0 R24 [t8] = 0 R25 [t9] = 0 R26 [k0] = 0 R27 [k1] = 0 R28 [gp] = 10008000 R29 [sp] = 7ffff754 R30 [s8] = 0 R31 [ra] = 400018	<pre> [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] 0c10001b jal 0x0040006c [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] 20020000 addi \$2, \$0, 0 ; 40: addi \$v0, \$zero, 0 # Initialize Sum to zero. [00400028] 00004020 add \$8, \$0, \$0 ; 41: add \$t0, \$zero, \$zero # Initialize array index i to zero. [0040002c] 03e00008 jr \$31 ; 57: jr \$ra # Return to caller. [00400030] 34020000 ori \$2, \$0, 0 ; 71: li \$v0, 0 # Initialize string length to 0 [00400034] 80880000 lb \$8, 0(\$4) ; 73: lb \$t0, 0(\$a0) # access current character [00400038] 11000006 beq \$8, \$0, 24 [Exit-0x00400038] ; 75: beq \$t0, '\n', Exit # In case compiler detects '\n' as a character [0040003c] 3401000a ori \$1, \$0, 10 ; 76: ori \$t0, 10 [00400040] 10220004 beq \$1, \$8, 16 [Exit-0x00400040] ; 77: beq \$t0, \$t0, 16 # len++ [00400044] 20420001 addi \$2, \$2, 1 ; 78: addi \$a0, \$a0, 1 # i++ [00400048] 20840001 addi \$4, \$4, 1 ; 80: j loop # next element [0040004c] 0810000d j 0x00400034 [loop] ; 83: jr \$ra #Return to caller [00400050] 03e00008 jr \$31 ; 100: addi \$sp, \$sp, -4 # PUSH return address to caller [00400054] 23bdfffc addi \$29, \$29, -4 ; 101: sv \$ra, 0(\$sp) [00400058] afbf0000 sw \$31, 0(\$29) ; 105: lw \$ra, 0(\$sp) # Pop return address to caller [0040005c] 8fbf0000 lw \$31, 0(\$29) ; 106: addi \$sp, \$sp, 4 [00400060] 23bd0004 addi \$29, \$29, 4 ; 109: jr \$ra [00400064] 03e00008 jr \$31 ; 121: jr \$ra [00400068] 03e00008 jr \$31 ; 166: addi \$sp, \$sp, -4 # PUSH return address [0040006c] 23bdfffc addi \$29, \$29, -4 ; 167: sv \$ra, 0(\$sp) [00400074] afbf0000 sw \$31, 0(\$29) ; 173: li \$v0, 4 [00400078] 3c011001 lui \$1, 4097 [STR_sum_of_fibonacci_a] [0040007c] 34240047 ori \$4, \$1, 71 [STR_sum_of_fibonacci_a] [00400080] 0000000c syscall ; 175: syscall [00400084] 3c011001 lui \$1, 4097 ; 177: lw \$a0, ARRAY_SIZE [00400088] 8c240000 lw \$4, 0(\$1) ; 178: li \$v0, 1 [0040008c] 34020001 ori \$2, \$0, 1 ; 179: syscall [00400090] 0000000c syscall ; 181: li \$v0, 4 [00400094] 34020004 ori \$2, \$0, 4 ; 182: li \$v0, 4 [00400098] 3c011001 lui \$1, 4097 [STR_sum_of_fibonacci_b] [0040009c] 34240057 ori \$4, \$1, 87 [STR_sum_of_fibonacci_b] [004000a0] 0000000c syscall ; 193: syscall [004000a4] 3c011001 lui \$1, 4097 [FIBONACCI_ARRAY]; 195: la \$a0, FIBONACCI_ARRAY [004000a8] 8c240000 lw \$4, 0(\$1) # FIBONACCI_ARRAY </pre>

## 4 Learning & Insights

Through this laboratory activity, I learned how to deal with strings in MIPS assembly specifically in the case of how to end the loop when it comes to checking every character in the string, in this case it would be the terminating NULL at the end of the string depicted by “.asciiz” when we first initialize the string.

In looping through the characters of a string, we first had to find a way to access those characters. Through this task, I learned that instead of using “lw” or load word, we had to use “lb” or load byte as each character in a string is one byte compared to words which are 4 bytes. To access the next character, we simply had to increment the address by 1 in each loop.

I also learned how to use “li” or load immediate which I think is just a way to initialize values. There was also “beqz” which is different from “beq” since it only takes one address instead of two. It helps if you are trying to determine if that address equates to 0 or NULL.

## 5 Comparison to a High-Level Implementation

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

```

#include <stdio.h>

int string_length(char str[]){
    int length = 0;

    for (int i = 0; str[i] != '\0'; ++i){
        length += 1;
    }
    return length;
}

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

```

Similar to our assembly source code, the main acts as the caller and the function sum acts as the callee. In looping through a string, compared to C where we had to check if the "i" in that array of characters equates to "\0" which is treated as a NULL character, in MIPS we didn't have to explicitly state the NULL character as the string is already a NULL terminated string since it was initialized with .asciiz. To detect the NULL character in the string, we simply had to use "beqz" to check if the current character equated to 0 or NULL.

In terms of printing the output, it is also a bit different since in C we didn't have to do instructions such as add, load immediate, or syscall, as we simply had to write printf and enclosed within are the strings or values we want to print 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. 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 manipulation of string variables in MIPS and it exposed me to other instructions that are very useful when it comes to strings such as "beqz" for detecting NULL's and "lb" loading a character in a string.