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.	4
Lab Title	MIPS
Email Addresses of all Authors	kbtan@up.edu.ph
Date of Document Issue	04/20/2022
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 **Fourth and second to the last task** which will be featured in this laboratory report is on how to convert a letter in a string into its uppercase form. To convert a whole string into uppercase, we would make use of the “string_for_each” subroutine, and in that subroutine we would call “to_upper” for every character in that string. The task is actually quite simple as we only had to load that particular character, then check if it is a lowercase letter. If it is not a lowercase letter, then we skip “to_upper” and proceed to the next character in the string. However, if it was a lowercase letter then we simply had to subtract 32 to get its uppercase letter. Through this task, I learned how to convert lowercase letters into uppercase and vice-versa in MIPS using its ASCII value, and finally I also learned how to replace or overwrite values in the memory.

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 and in that process I also learned about the differences between an uppercase and a lowercase letter which is in their ASCII code. After that, the steps are quite simple as I just had to loop the “to_upper” subroutine in the “string_for_each” subroutine up until it detects a terminating NULL in that string in order to convert the whole string into uppercase letters.

TABLE OF CONTENTS

1 The Lab Problem

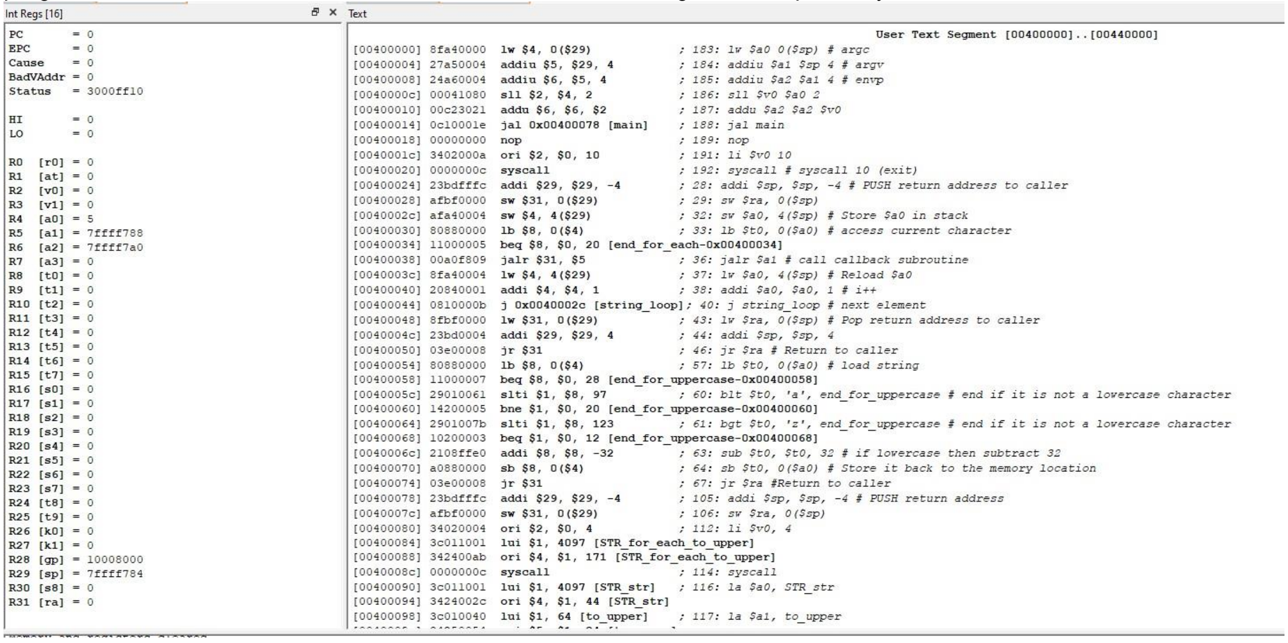
The problem to be solved in the fourth task of the laboratory activity is to create a subroutine wherein it takes the address of a letter in the string and convert it into uppercase letters. In this process, the uppercase letter will be placed in the same memory address, thus overwriting it. We will then be using the previous task's "string_for_each" to convert a whole string into uppercase letters.

2 The Assembly Source Code

```
##############################################################################
#
#  DESCRIPTION: Transforms a lower case character [a-z] to upper case [A-Z].
#
#        INPUT: $a0 - address of a character
#
##############################################################################

to_upper:
    #### Write your solution here ####
    lb   $t0, 0($a0)                            # load string
    beqz $t0, end_for_uppercase                 # check for null character

    blt  $t0, 'a', end_for_uppercase            # end if it is not a lowercase
character
    bgt  $t0, 'z', end_for_uppercase            # end if it is not a lowercase
character

    sub  $t0, $t0, 32                           # if lowercase then subtract 32
    sb   $t0, 0($a0)                            # Store it back to the memory location

end_for_uppercase:
    jr   $ra                                    #Return to caller
```

3 Screenshots

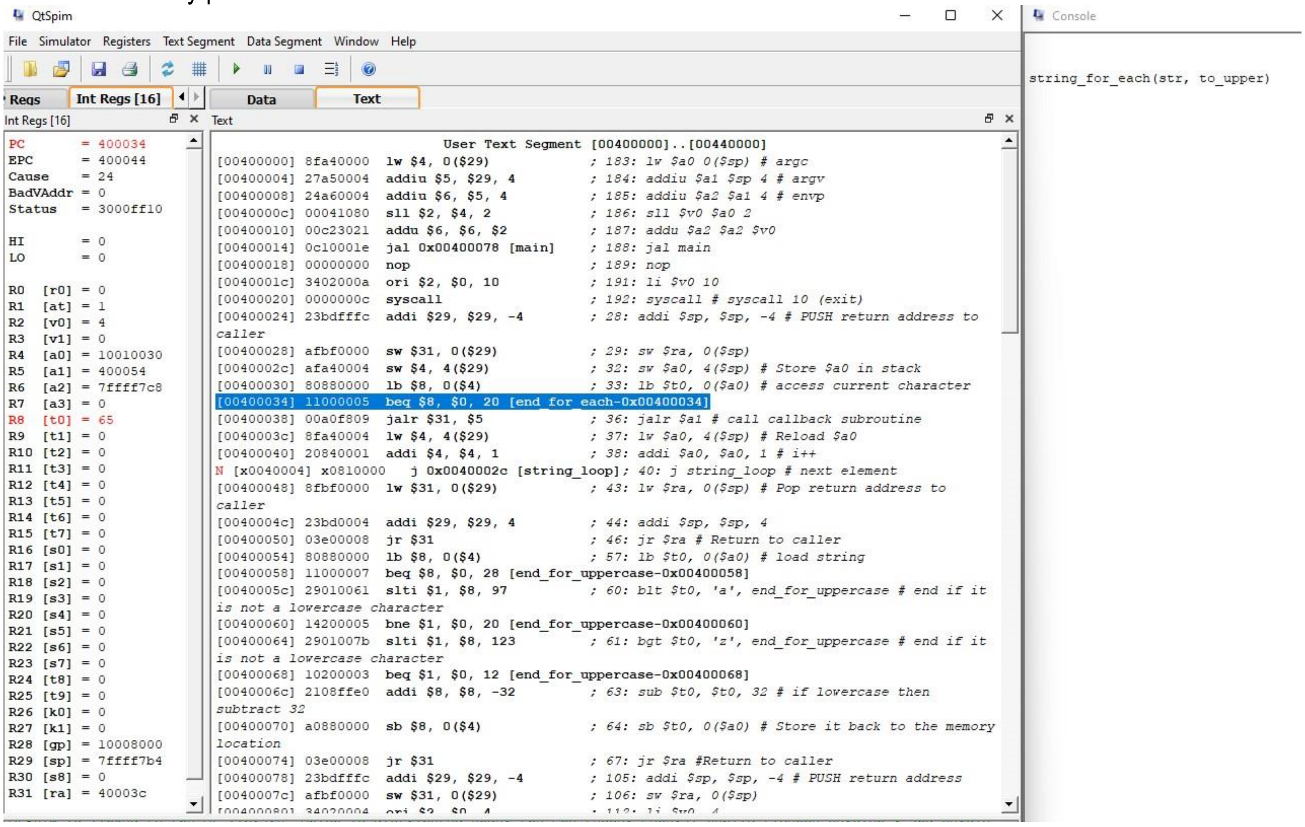
A. **Start of Execution** – Show the current state of the registers, code/program/text segment and data and stack segments in the memory after the program is loaded. Show the initial values of the registers. Show that your program and data has been loaded in the code and the data segments respectively.



```
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] = 5
R5  [a1] = 7ffff788
R6  [a2] = 7ffff7a0
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] = 7ffff784
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] 0c10001e  jal 0x00400078 [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] 80880000  lb $8, 0($4)            ; 57: lb $t0, 0($a0) # load string
[00400058] 11000007  beq $8, $0, 28 [end_for_uppercase-0x00400058]
[0040005c] 29010061  slti $1, $8, 97         ; 60: blt $t0, 'a', end_for_uppercase # end if it is not a lowercase character
[00400060] 14200005  bne $1, $0, 20 [end_for_uppercase-0x00400060]
[00400064] 2901007b  slti $1, $8, 123        ; 61: bgt $t0, 'z', end_for_uppercase # end if it is not a lowercase character
[00400068] 10200003  beq $1, $0, 12 [end_for_uppercase-0x00400068]
[0040006c] 2108ffe0  addi $8, $8, -32        ; 63: sub $t0, $t0, 32 # if lowercase then subtract 32
[00400070] a0880000  sb $8, 0($4)            ; 64: sb $t0, 0($a0) # Store it back to the memory location
[00400074] 03e00008  jr $31                  ; 67: jr $ra #Return to caller
[00400078] 23bdfffc  addi $29, $29, -4       ; 105: addi $sp, $sp, -4 # PUSH return address
[0040007c] afbf0000  sw $31, 0($29)          ; 106: sw $ra, 0($sp)
[00400080] 34020004  ori $2, $0, 4           ; 112: li $v0, 4
[00400084] 3c011001  lui $1, 4097 [STR_for_each_to_upper]
[00400088] 342400ab  ori $4, $1, 171 [STR_for_each_to_upper]
[0040008c] 0000000c  syscall                 ; 114: syscall
[00400090] 3c011001  lui $1, 4097 [STR_str]  ; 116: la $a0, STR_str
[00400094] 3424002c  ori $4, $1, 44 [STR_str]
[00400098] 3c010040  lui $1, 64 [to_upper]   ; 117: la $a1, to_upper
```

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

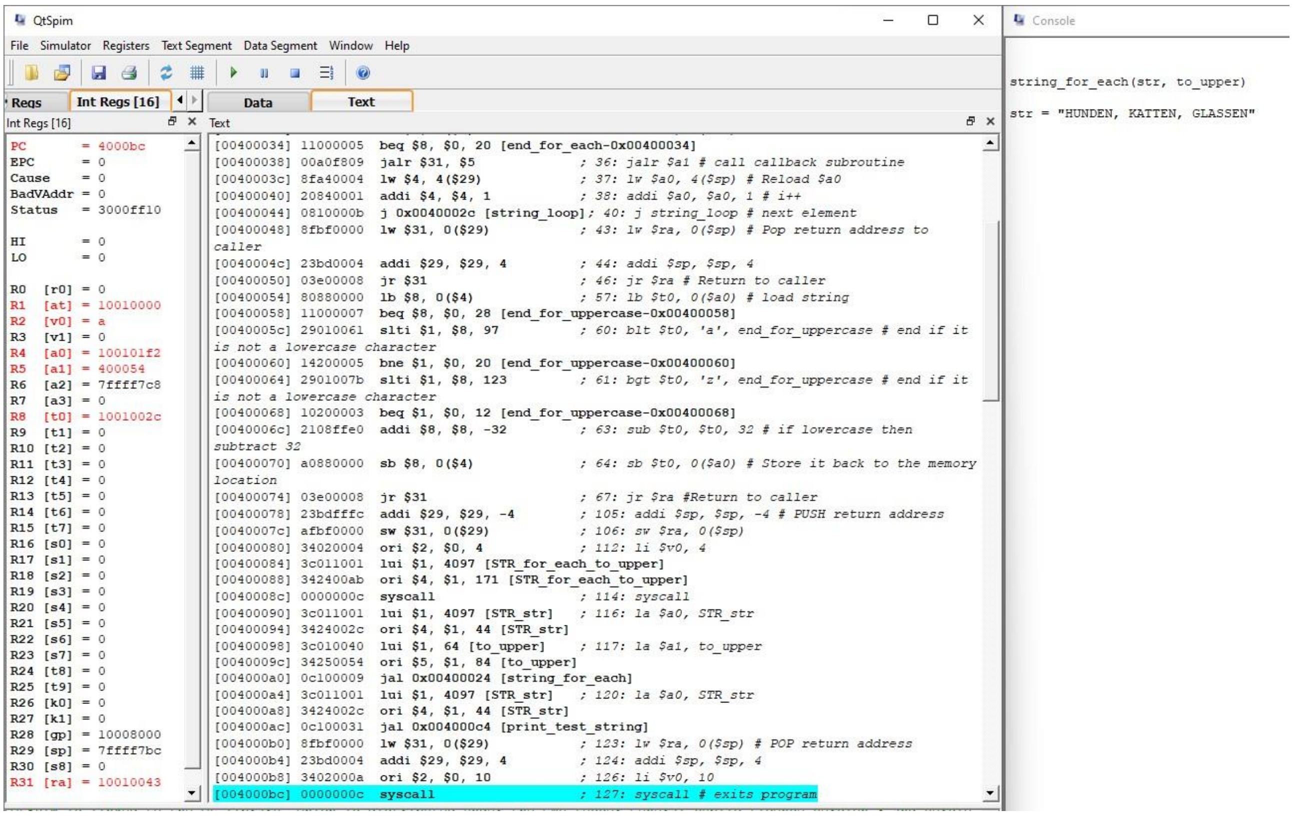

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Regs
Int Regs [16]
Data
Text
Console

```
Int Regs [16]

PC       = 400034
EPC      = 400044
Cause    = 24
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 1
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 10010030
R5  [a1] = 400054
R6  [a2] = 7ffff7c8
R7  [a3] = 0
R8  [t0] = 65
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] = 7ffff7b4
R30 [s8] = 0
R31 [ra] = 40003c
```

```
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] 0c10001e  jal 0x00400078 [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++
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] 80880000  lb $8, 0($4)            ; 57: lb $t0, 0($a0) # load string
[00400058] 11000007  beq $8, $0, 28 [end_for_uppercase-0x00400058]
[0040005c] 29010061  slti $1, $8, 97         ; 60: blt $t0, 'a', end_for_uppercase # end if it
is not a lowercase character
[00400060] 14200005  bne $1, $0, 20 [end_for_uppercase-0x00400060]
[00400064] 2901007b  slti $1, $8, 123        ; 61: bgt $t0, 'z', end_for_uppercase # end if it
is not a lowercase character
[00400068] 10200003  beq $1, $0, 12 [end_for_uppercase-0x00400068]
[0040006c] 2108ffe0  addi $8, $8, -32        ; 63: sub $t0, $t0, 32 # if lowercase then
subtract 32
[00400070] a0880000  sb $8, 0($4)            ; 64: sb $t0, 0($a0) # Store it back to the memory
location
[00400074] 03e00008  jr $31                  ; 67: jr $ra #Return to caller
[00400078] 23bdfffc  addi $29, $29, -4       ; 105: addi $sp, $sp, -4 # PUSH return address
[0040007c] afbf0000  sw $31, 0($29)          ; 106: sw $ra, 0($sp)
```

```
string_for_each(str, to_upper)
```

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
Console

```
Int Regs [16]
PC       = 4000bc
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] = 100101f2
R5  [a1] = 400054
R6  [a2] = 7ffff7c8
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 [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] = 10010043
```

```
Text
[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] 80880000  lb $8, 0($4)             ; 57: lb $t0, 0($a0) # load string
[00400058] 11000007  beq $8, $0, 28 [end_for_uppercase-0x00400058]
[0040005c] 29010061  slti $1, $8, 97          ; 60: blt $t0, 'a', end_for_uppercase # end if it
is not a lowercase character
[00400060] 14200005  bne $1, $0, 20 [end_for_uppercase-0x00400060]
[00400064] 2901007b  slti $1, $8, 123         ; 61: bgt $t0, 'z', end_for_uppercase # end if it
is not a lowercase character
[00400068] 10200003  beq $1, $0, 12 [end_for_uppercase-0x00400068]
[0040006c] 2108ffe0  addi $8, $8, -32         ; 63: sub $t0, $t0, 32 # if lowercase then
subtract 32
[00400070] a0880000  sb $8, 0($4)             ; 64: sb $t0, 0($a0) # Store it back to the memory
location
[00400074] 03e00008  jr $31                   ; 67: jr $ra #Return to caller
[00400078] 23bdfffc  addi $29, $29, -4        ; 105: addi $sp, $sp, -4 # PUSH return address
[0040007c] afbf0000  sw $31, 0($29)           ; 106: sw $ra, 0($sp)
[00400080] 34020004  ori $2, $0, 4            ; 112: li $v0, 4
[00400084] 3c011001  lui $1, 4097 [STR_for_each_to_upper]
[00400088] 342400ab  ori $4, $1, 171 [STR_for_each_to_upper]
[0040008c] 0000000c  syscall                  ; 114: syscall
[00400090] 3c011001  lui $1, 4097 [STR_str]   ; 116: la $a0, STR_str
[00400094] 3424002c  ori $4, $1, 44 [STR_str]
[00400098] 3c010040  lui $1, 64 [to_upper]    ; 117: la $a1, to_upper
[0040009c] 34250054  ori $5, $1, 84 [to_upper]
[004000a0] 0c100009  jal 0x00400024 [string_for_each]
[004000a4] 3c011001  lui $1, 4097 [STR_str]   ; 120: la $a0, STR_str
[004000a8] 3424002c  ori $4, $1, 44 [STR_str]
[004000ac] 0c100031  jal 0x004000c4 [print_test_string]
[004000b0] 8fbf0000  lw $31, 0($29)           ; 123: lw $ra, 0($sp) # POP return address
[004000b4] 23bd0004  addi $29, $29, 4         ; 124: addi $sp, $sp, 4
[004000b8] 3402000a  ori $2, $0, 10           ; 126: li $v0, 10
[004000bc] 0000000c  syscall                  ; 127: syscall # exits program
```

```
string_for_each(str, to_upper)

str = "HUNDEN, KATTEN, GLASSEN"
```

4 Learning & Insights

Through this laboratory activity, I learned how to deal with the ASCII values of characters and through that we can convert lowercase letters into uppercase letters and vice-versa. What I realized is that all the lowercase letters are simply +32 of the ASCII value of their corresponding uppercase letters, and from that information we can subtract or add depending on what type of letter we want whether it be uppercase or lowercase. As an example, the ASCII value of the character “a” is 97 and to get the uppercase letter we simply have to subtract 32 from it to get 65 which is the ASCII value of the character “A”.

This particular task also helped me understand through practical implementation on how “blt” and “bgt” works which is simply a way to check if the value of the character in that string is “blt” or less than ‘a’ or if that value is “bgt” or greater than ‘z’, if any of these conditions return True then the character is not a lowercase letter and thus we branch to “end_for_uppercase” to skip converting it to uppercase and then we proceed to the next letter 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 to_upper(char str[]){
    for (int i = 0; str[i] != '\0'; i++){
        if (str[i] >= 'a' && str[i] <= 'z'){
            str[i] = str[i] - 32;
        }
    }
    return 0;
}


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

The most prominent difference here to our MIPS code is that I didn't need to use the function "string_for_each" to convert a whole string into uppercase letters as it would only make the code a bit more complicated compared to simply doing the whole operation in one function. If I wanted to make it much simpler, we could also use the function "strupr()" to convert a whole string into uppercase letters but if we did that, we wouldn't be able to see how we really convert lowercase letters to uppercase and vice-versa through subtraction or addition. In C, instead of using "blt" or "bgt" for comparing values to decide whether we skip converting the letter which is the case if it was not a lowercase letter as we didn't need to convert an uppercase letter when it is already uppercase, we simply had to use the operations ">=" to check if the character value is greater than 'a' and "<=" to check if the character value is lower than 'z'. If both are true, we then convert it to uppercase by subtracting 32. We continue converting each character in the string up until we detect a NULL value which indicates the end of the string.

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 convert lowercase letters into uppercase and vice-versa. 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 replace the character in its memory address by its uppercase form so when we print the whole string, what comes out is the uppercase of the string.