COE 181. 1 Lab Report

Lab Number: Lab 2

Lab Title: INTRODUCTION TO MIPS ASSEMBLY PROGRAMMING

Student Name: Janna Joyce E. Jumao-as

Student Email: jannajoyce.jumao-as@g.msuiit.edu.ph

Student Phone Number: 09309321199

Objectives:

- Learn about the MIPS assembly language
- Write simple MIPS programs
- Use system calls for simple input and output

Procedures:

Task 1:

- Modified the program shown in Figure 2.4.
- Created a program that will ask the user to enter an integer and double the input integer.
- Implemented the "**add"** instruction for doubling the input integer.

Task 2:

- Modify the program shown in Task 1.
- Created a program that after performing the calculations, the program asks the user if they want to repeat the execution.
- Implemented **"beq"** instruction to check if the input character is "y". If "y", the program will restart.

Task 3:

- Created a program that prompts the user to enter their name and the prints "Hello [Name]".
- Implemented load immediate and load address instruction.
- Limit the string input to 20 characters.

Task 4:

- Created a program that prompts the user to enter three integers (a, b, c).
- Store the three integers in different registers.
- Implemented add, addi, and sub instruction to calculate s = (a + b) – (c+10).

Task 5:

- Created a program that prompts the user to input two integers.
- The program should output equal if the two integers are equal. Otherwise, it should output not equal.
- Implemented "**beq**" and "**bne**" instruction.

Results and Analysis:

Task 1:

```
str3:           .asciiz         "\nThe double is: "
```

```
#doubling the input number
        li      $v0, 4
        la      $a0, str3
        syscall
        add     $s1, $s0, $s0   #double the input number using add instruction
        li      $v0, 1
        move    $a0, $s1
        syscall
```

- ***add $s1, $s0, $s0:*** *Adds the input value $s0 to itself,* ***effectively doubling it****, and stores the result in $s1.*

Task 2:

```
str4:           .asciiz         "\nRepeat [y/n]?: "
```

```
#repeat the program(?)

        li      $v0, 4
        la      $a0, str4
        syscall

        li      $v0, 12         #use service code 12 to read a character
        syscall

        li      $t0, 'y'
        beq     $t0, $v0, main  #compare the input and the t0, if equal back to main label.


        li      $v0, 10
        syscall
```

- ***li $v0, 12: Loads system call code 12****, which is used to read a single character input from the user.*
- ***beq $t0, $v0, main:*** *Compares the value in $t0 ('y') with the user input in $v0. If the input character is 'y', the program branches back to the main label, effectively restarting the program from the beginning.*

Task 3:

lab2task1and2.asm | lab2task3.asm

```
.data
str1:           .asciiz         "\nEnter your name:"
str2:           .asciiz         "Hello "
str3:           .space          20

.globl          main
.text
main:
        li      $v0, 4          #service code for print string
        la      $a0, str1       #load address of str1 into $a0
        syscall                 #print str1 string
        li      $v0, 8
        la      $a0, str3
        li      $a1, 20
        syscall
        li      $v0, 4          #service code for print string
        la      $a0, str2       #load address of str1 into $a0
        syscall                 #print str1 string
        li      $v0, 4          #service code for print string
        la      $a0, str3       #load address of str1 into $a0
        syscall                 #print str1 string

li $v0, 10
syscall
```

- *This program prompts the user to enter their name, then greets them by printing "Hello [Name]."*
- ***la $a0, str2****: Loads the address of str2 ("Hello ") into $a0.*
- ***la $a0, str3****: Loads the address of str3 (the user's name) into $a0.*

Task 4:

lab2task1and2.asm | lab2task3.asm | labtask4.asm

```
.data
str1:           .asciiz         "\nEnter the value of a:"
str2:           .asciiz         "\nEnter the value of b:"
str3:           .asciiz         "\nEnter the value of c:"
str4:           .asciiz         "\nThe value of s is "

.globl          main
.text
main:
        li      $v0, 4          #service code for print string
        la      $a0, str1       #load address of str1 into $a0
        syscall                 #print str1 string
        li      $v0, 5          #service code for read integer
        syscall                 #read integer input into $v0
        move    $s0, $v0        #save input value in $s0

        li      $v0, 4          #service code for print string
        la      $a0, str2       #load address of str1 into $a0
        syscall                 #print str1 string
        li      $v0, 5          #service code for read integer
        syscall                 #read integer input into $v0
        move    $s1, $v0        #save input value in $s0

        li      $v0, 4          #service code for print string
        la      $a0, str3       #load address of str1 into $a0
        syscall                 #print str1 string
        li      $v0, 5          #service code for read integer
        syscall                 #read integer input into $v0
        move    $s2, $v0        #save input value in $s0

        add $s3, $s0, $s1
        addi $s4, $s2, 101
        sub $s5, $s3, $s4
```

```

        li      $v0, 4          #service code for print string
        la      $a0, str4       #load address of str1 into $a0
        syscall                 #print str1 string
        li      $v0, 1          #service code to print integer
        move    $a0, $s5        #copy input value
        syscall                 #print integer

        li $v0, 10
syscall
```

- *a:*

```
ages   Run I/O

Enter the value of a:5

Enter the value of b:10

Enter the value of c:-30

The value of s is -56
-- program is finished running --
```

- *b:*

```
30
31          add $s3, $s0, $s1
32          addi $s4, $s2, 101
33          sub $s5, $s3, $s4
34
```

For add instruction, the registers are $s3 (value of a), $s0 (value of b), and $s1(result of a + b).

For addi instruction, the registers are $s2 (value of c) and $s4 (result of c + 101).

For sub instruction, the registers are $s5 (result of (a + b) – (c + 101), $s3 (value of a + b), and $s4 (value of c + 101).

The sub $s5, $s3, $s4 will compute the s.

- *c:*

0x00400054	0x02119820	add $19,$16,$17	31:	add $s3, $s0, $s1
0x00400058	0x22540065	addi $20,$18,101	32:	addi $s4, $s2, 101
0x0040005c	0x0274a822	sub $21,$19,$20	33:	sub $s5, $s3, $s4

The address of sub $s5, $s3, $s4 is 0x0040005c.

- *d:*

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	0
$a0	4	-56
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	5
$s1	17	10
$s2	18	-30
$s3	19	15
$s4	20	71
$s5	21	-56
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194436
hi		0
lo		0

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0xffffffc8
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000005
$s1	17	0x0000000a
$s2	18	0xffffffe2
$s3	19	0x0000000f
$s4	20	0x00000047
$s5	21	0xffffffc8
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400084
hi		0x00000000
lo		0x00000000

- *Since the value of s will be stored at $s5, the $s5 in decimal is -56 and in hexadecimal is 0xffffffc8.*

Task 5:

lab2task1and2.asm | lab2task3.asm | labtask4.asm | lab2task5.asm

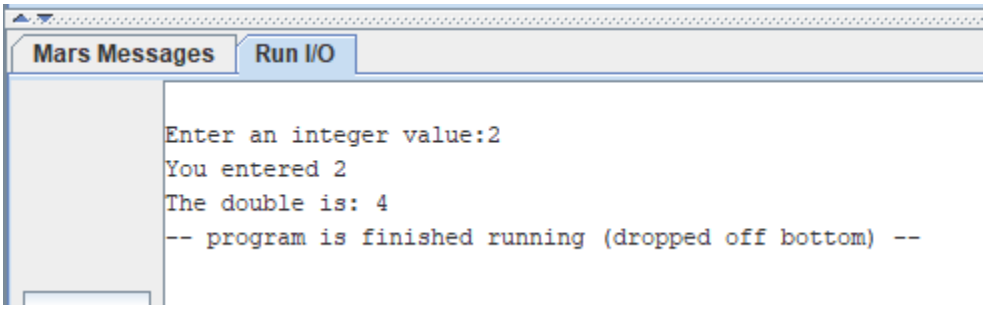
```
.data
str1:           .asciiz         "Enter first integer:"
str2:           .asciiz         "Enter second integer:"
str3:           .asciiz         "equal"
str4:           .asciiz         "not equal"

.globl          main
.text
main:
        li      $v0, 4          #service code for print string
        la      $a0, str1       #load address of str1 into $a0
        syscall                 #print str1 string
        li      $v0, 5          #service code for read integer
        syscall                 #read integer input into $v0
        move    $s0, $v0        #save input value in $s0

        li      $v0, 4          #service code for print string
        la      $a0, str2       #load address of str1 into $a0
        syscall                 #print str1 string
        li      $v0, 5          #service code for read integer
        syscall                 #read integer input into $v0
        move    $s1, $v0        #save input value in $s0

        beq $s0, $s1, equal
        bne $s0, $s1, not_equal


equal:
        li      $v0, 4          #service code for print string
        la      $a0, str3       #load address of str1 into $a0
        syscall                 #print str1 string
        li $v0, 10
        syscall

not_equal:
        li      $v0, 4          #service code for print string
        la      $a0, str4       #load address of str1 into $a0
        syscall                 #print str1 string
        li $v0, 10
        syscall
```

- *This program compares two integers entered by the user and outputs whether they are equal or not equal.*

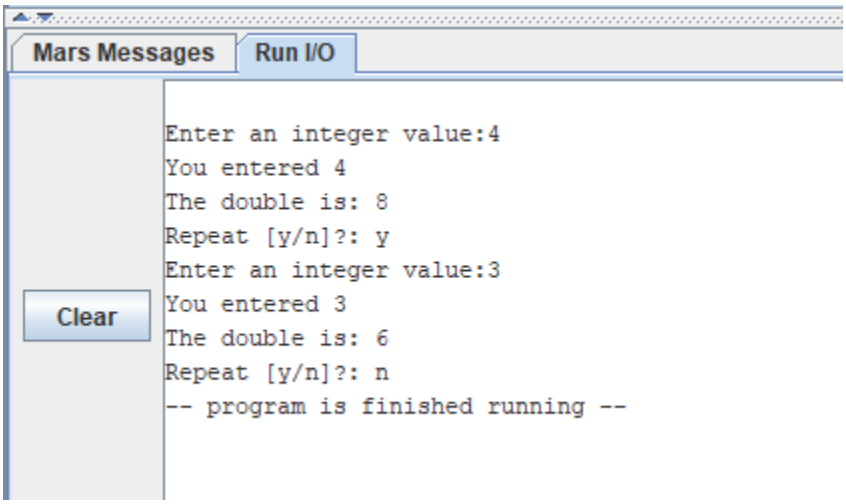
- ***beq $s0, $s1, equal:*** *If the first and second integers are equal, branch to the equal label.*
- ***bne $s0, $s1, not_equal:*** *If the integers are not equal, branch to the not_equal label.*

Screenshots (Output):

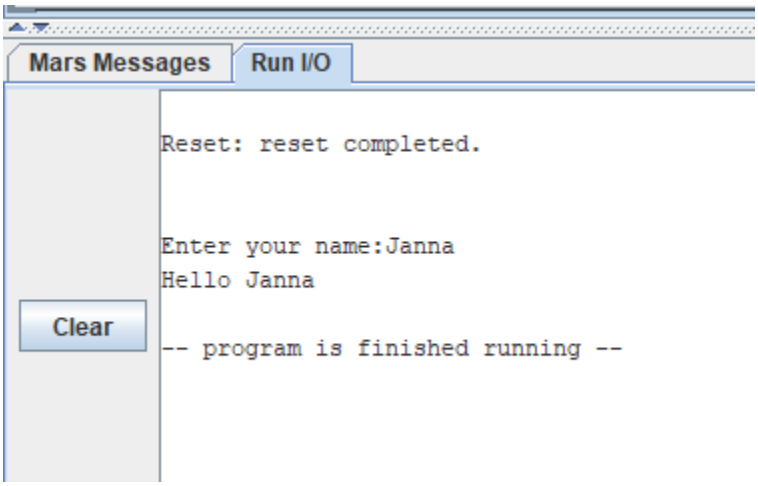
Task 1:



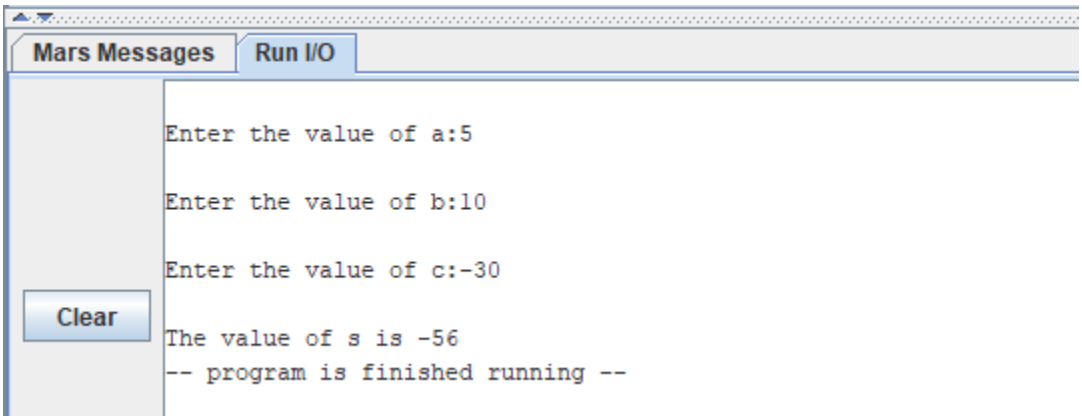
Task 2:



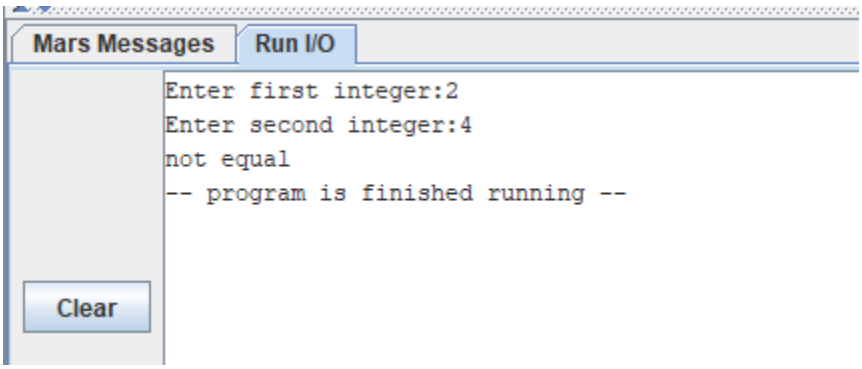
Task 3:

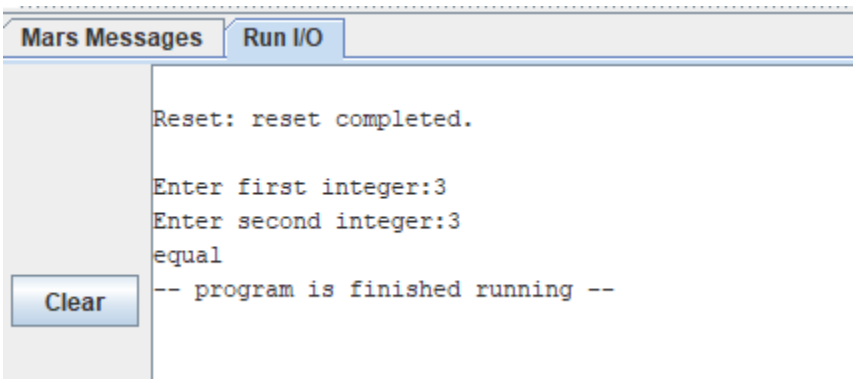


Task 4:



Task 5:





Conclusion:

- This activity helped the student gain familiarity with assembler directives, MIPS instructions, registers, and their format and syntax.
- It provided insight into implementing basic arithmetic and control instructions in MIPS assembly language.
- The student also learned how the MARS simulator handles syscall exceptions and provides system services to programs.

- Furthermore, the activity underscored the significance of branch instructions, especially for comparing values and controlling program flow.

Additional Notes/Observations:

- Registers like $s0 and $s1 are limited, so it must always ensure that save or restore values in registers if they need to be reused across different parts of the code.