COE 181.1 Lab Report

Lab Number: Lab 4

Lab Title: FLOW CONTROL

Student Name: Janna Joyce E. Jumao-as

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

Student Phone Number: 09309321199

Objectives:

- Get familiar with MIPS Jump and Branch instructions
- Learn about pseudo instructions in MIPS
- Learn how to translate high-level flow control constructs (if-then-else, for loop, while loop) to MIPS code

Procedures:

Task 1: *Write a MIPS program that asks the user to enter an integer and then displays the number of 1's in the binary representation of that integer. For example, if the user enters 9, then the program should display 2.*

- Display a message asking the user to enter an integer and then read that integer from input.
- Set a register to zero to hold the count of 1 bits.
- Use a loop to check each bit of the integer.
- **"and"** the integer with 1 to check if the least significant bit is 1. If true, increment the counter. If the bit is zero, skip increment
- Shift the integer right by one bit to process the next bit in the next loop iteration.
- Print the counter, which now holds the total count of 1s in the binary representation.

Task 2: *Write a program that asks the user to enter two integers: n1 and n2 and prints the sum of all numbers from n1 to n2. For example, if the user enters n1=3 and n2=7, then the program should display the sum as 25.*

- Ask the user to enter two integers n1 and n2.
- Ensure that n1 is less than or equal to n2. If not, display an error or exit.
- Use a loop to add all numbers from n1 to n2.
- Print the final result after the loop.

Task 3: *Write a program that asks the user to enter an integer and then display the hexadecimal representation of that integer.*

- Ask the user to enter an integer.
- Use syscall code 34, to print the integer in hexadecimal.
- Limit the string input to 20 characters.

Task 4: *Write a program that asks the user to enter a positive integer number n and then prints the nth number in the Fibonacci sequence.*

- Ask the user to enter a positive integer n.
- Check if the input is positive (n > 0). If n< 0, an error message is displayed, and the program exits.
- Loop to calculate nth Fibonacci number. Iteratively calculates Fibonacci numbers from 0 to n using registers $t1 (for fib0) and $t2 (for fib1), following the provided algorithm.

- Display the nth Fibonacci number.

Results and Analysis:

Task 1:

```
19
20      # Initialize counter for 1's count
21      li $t1, 0                 # $t1 holds the count of 1's
22
23  count_ones:
24      # Check if the least significant bit is 1
25      andi $t2, $t0, 1          # $t2 = $t0 AND 1 (extract least significant bit)
26      beq $t2, $zero, skip      # If the bit is 0, skip increment
27
28      # Increment the counter if LSB is 1
29      addi $t1, $t1, 1          # Increment count of 1's
30
31  skip:
32      # Shift right to process the next bit
33      srl $t0, $t0, 1           # Shift $t0 right by 1 bit
34      bne $t0, $zero, count_ones # Repeat until $t0 becomes 0
35
36      # Display result message
37      li $v0, 4
38      la $a0, result
39      syscall
40
41      # Print the count of 1's
42      move $a0, $t1             # Move count to $a0 for printing
43      li $v0, 1                 # Print integer syscall
44      syscall
```

For instance, I enter an integer 9.

In first iteration, $t0 = 1001 in binary (decimal 9). Then, the andi $t2, $t0, 1 check if the least significant bit (LSB) is 1. Since it is, $t1 is incremented to 1. Next, the srl $t0, $t0, 1 shift $t0 right by 1 bit, making $t0 = 100 (decimal 4).

In second iteration, $t0 = 100 in binary (decimal 4). Then, the andi $t2, $t0, 1 check if the LSB is 1. This time, it is 0, so $t1 remains 1. Next, the srl $t0, $t0, 1 shift $t0 right by 1 bit, making $t0 = 10 (decimal 2).

In third iteration, $t0 = 10 in binary (decimal 2). Then, the andi $t2, $t0, 1 check if the LSB is 1. It's 0, so $t1 remains 1. Next, the srl $t0, $t0, 1 shift $t0 right by 1 bit, making $t0 = 1 (decimal 1).

In fourth iteration, $t0 = 1 in binary (decimal 1). Then, the andi $t2, $t0, 1 check if the LSB is 1. Since it is, $t1 is incremented to 2. Next, the srl $t0, $t0, 1 shift $t0 right by 1 bit, making $t0 = 0.

Lastly, the loop exits when $t0 becomes 0. Hence, the program displays the value of $t1, which is 2, indicating that 9 has two 1s in its binary representation (1001).

Task 2:

```
31
32      # Check if n1 <= n2
33      ble $t0, $t1, calculate_sum  # If n1 <= n2, continue to sum calculation
34
35      # Error if n1 > n2
36      li $v0, 4
37      la $a0, error
38      syscall
39      j exit                      # Jump to program exit
40
41  calculate_sum:
42      # Initialize sum
43      move $t2, $t0               # $t2 is the running number (starts from n1)
44      li $t3, 0                   # $t3 will hold the total sum
45
46  sum_loop:
47      # Add current number to sum
48      add $t3, $t3, $t2           # $t3 = $t3 + $t2
49
50      # Increment number and check if it reached n2
51      addi $t2, $t2, 1            # $t2 = $t2 + 1
52      ble $t2, $t1, sum_loop      # Continue loop if $t2 <= $t1
53
54      # Print result message
55      li $v0, 4
56      la $a0, result
57      syscall
58
59      # Print the sum
60      move $a0, $t3               # Move the sum to $a0 for printing
61      li $v0, 1                   # Syscall to print integer
62      syscall
```

- **ble $t0, $t1, calculate_sum:** This instruction checks if n1 (in $t0) is less than or equal to n2 (in $t1). If true, it jumps to the calculate_sum label to proceed with the sum calculation.
- **add $t3, $t3, $t2:** Adds the current value of $t2 (the running number) to $t3 (the total sum).
- **addi $t2, $t2, 1:** Increments $t2 by 1 to prepare for the next iteration.
- **ble $t2, $t1, sum_loop:** Checks if $t2 is still less than or equal to n2. If true, it continues looping to add the next integer.

For instance, n1 = 3 and n2 = 7. On each iteration, the program adds the current value of t2 (starting from 3) to t3, which stores the sum.

- Iteration 1: 3 (total: 3)
- Iteration 2: 3+4=7 (total: 7)
- Iteration 3: 7+5=12 (total: 12)
- Iteration 4: 12+6=18 (total: 18)

- Iteration 5: 18+7=25 (total: 25)

Hence, the program prints 25, which is the sum of numbers from 3 to 7.

Task 3:

```
    # Print the integer in hexadecimal
    move $a0, $t0               # move the integer to $a0 for printing
    li $v0, 34                  # syscall for print_hex
    syscall
```

- **li $v0, 34:** This instruction loads the immediate value 34 into register $v0. In MIPS, the value stored in $v0 determines which system call to execute. Here, 34 corresponds to the syscall for printing an integer in hexadecimal format.

Task 4:

```
    # Initialize Fibonacci values: Fib0 = 0, Fib1 = 1
    li $t1, 0                   # fib0 = 0
    li $t2, 1                   # fib1 = 1

    # Check if n == 0, then result is fib0 (0)
    beq $t0, 0, fib0

    # Loop to calculate Fibonacci up to the nth term
    li $t3, 2                   # i = 2
fibonacci_loop:
    # Exit loop if i > n
    bgt $t3, $t0, fib1

    # Execute the algorithm steps
    move $t4, $t1               # temp = fib0
    move $t1, $t2               # fib0 = fib1
    add $t2, $t4, $t2           # fib1 = temp + fib1

    # Increment i
    addi $t3, $t3, 1            # i++
    j fibonacci_loop            # repeat the loop

```

- **li $t1, 0:** fib0 = 0.
- **li $t2, 1:** fib1 = 1.
- **beq $t0, 0, print_fib0:** This instruction checks if the user input (stored in $t0) is 0. If it is, the program branches to the label print_fib0, where it will print the first Fibonacci number (which is 0). This handles the edge case for Fibonacci of 0.
- **li $t3, 2:** This instruction initializes the loop counter $t3 to 2, as the Fibonacci calculation starts from the third number (since Fib0 and Fib1 are already defined).

- **bgt $t3, $t0, print_fib1:** This instruction checks if the loop counter $t3 (which represents the current Fibonacci position) is greater than the input value $t0. If true, it branches to print_fib1, where it will output the second Fibonacci number (1).
- **move $t4, $t1:** temp = fib0
- **move $t1, $t2:** fib0 = fib1
- **add $t2, $t4, $t2:** fib1 = temp + fib1
- **addi $t3, $t3, 1:** This instruction increments the loop counter $t3 by 1, moving to the next Fibonacci number in the sequence.
- **j fibonacci_loop**: This instruction jumps back to the fibonacci_loop label, repeating the process until the loop condition is no longer satisfied.

For instance, I enter an integer 7.

The loop starts at i=2 and continues until it reaches n=7.

Iteration 1 (i=2):

- temp=0 (fib0)
- Update fib0=1 (fib1)
- Update fib1=0+1=1 (new fib1)

Iteration 2 (i=3):

- temp=1 (fib0)
- Update fib0=1 (fib1)
- Update fib1=1+1=2

Iteration 3 (i=4):

- temp=1
- Update fib0=2
- Update fib1=1+2=3

Iteration 4 (i=5):

- temp=2
- Update fib0=3
- Update fib1=2+3=5

Iteration 5 (i=6):

- temp=3
- Update fib0=5
- Update fib1=3+5=8

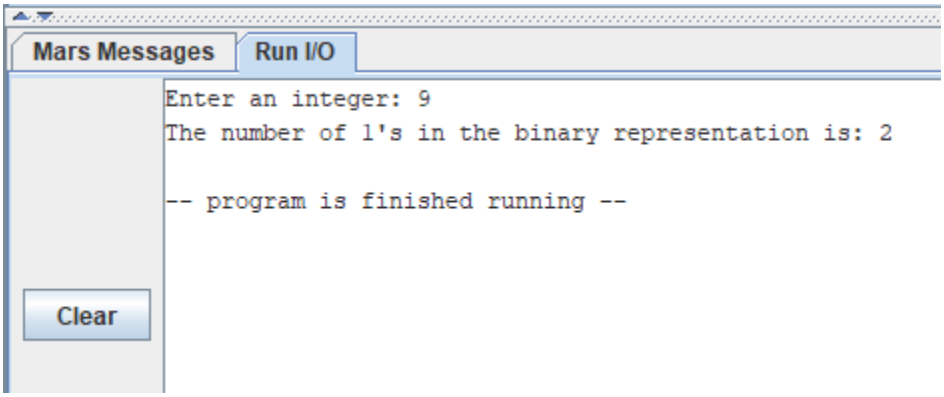
Iteration 6 (i=7):

- temp=5
- Update fib0=8
- Update fib1=5+8=13

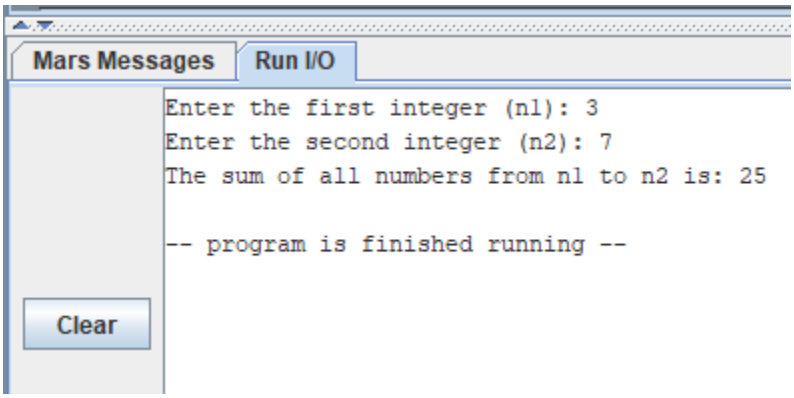
The program prints the 7th Fibonacci number, which is stored in fib1=13. Thus, the result for the input **7** is **13**.

Screenshots (Output):

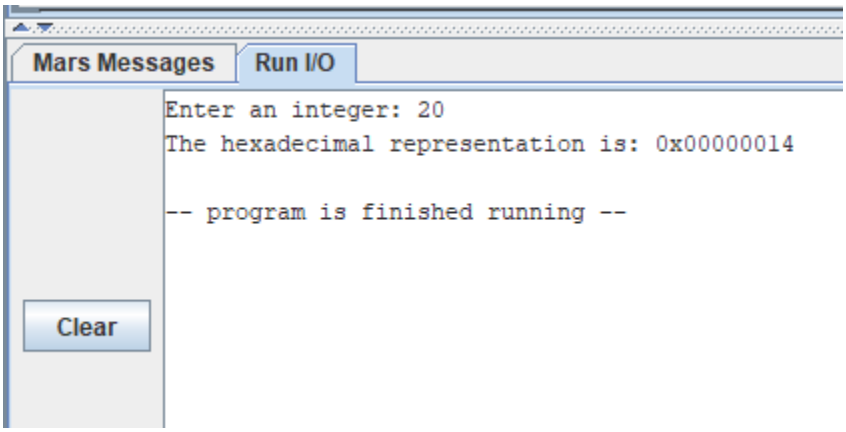
Task 1:



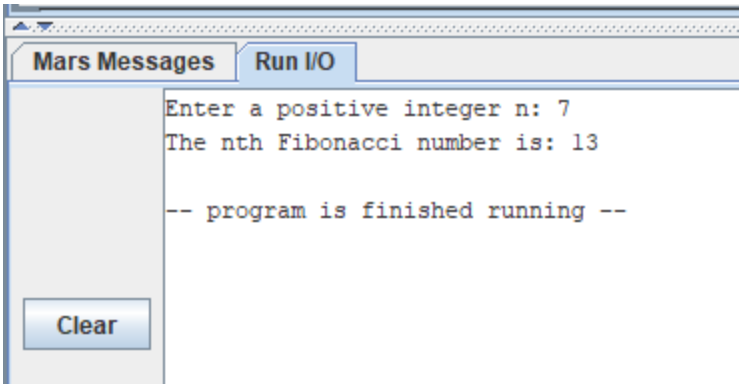
Task 2:



Task 3:



Task 4:



Conclusion:

- This activity helped me to gain a better understanding of using jump instruction, branch instructions, and pseudo-instructions
- It practiced me how to translate high-level language constructs into MIPS and I learn how to manage loops such as initialization, condition checks, and iteration steps.
- This activity provided hands-on experience performing bitwise operations, counting bits, and managing binary representations of integers.

Additional Notes/Observations:

- The Fibonacci sequence problem is the most challenging for me because we know that in assembly there are no high-level constructs like array indexing or function calls which can be difficult to maintain updates of two variables while iterating up to a specified count.
- Using bgt or bne to exit loops at a certain condition, help to structure program flow even without high-level syntax.