**COE 181. 1 Lab Report**

**Lab Number:** Lab 3

**Lab Title:** INTEGER ARITHMETIC

**Student Name:** Janna Joyce E. Jumao-as

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

**Student Phone Number:** 09309321199

**Objectives:**

* Get familiar with the basic MIPS integer arithmetic and logic instructions including:
* Integer addition and subtraction instructions
* Bitwise logic instructions
* Shift instructions
* Learn some useful applications of these instructions.

**Procedures:**

**Task 1:**

* Ask the user to enter the integer A and integer B.
* Multiply the value of B by 2 to get 2B.
* Add the value of A to 2B. Then, subtract 5 from the result of the addition.
* Display the result of A + 2B – 5.
* End the program.

**Task 2:**

* Set the value of $s1 to 0x12345678 and $s2 to 0xffff9a00.
* Compute bitwise AND of $s1 and $s2, then store the result in $s3.
* Compute the bitwise OR of $s1 and $s2, and store the result in $s4.
* Compute the bitwise XOR of $s1 and $s2, and store the result in $s5.
* Compute the bitwise NOR of $s1 and $s2, and store the result in $s6.
* After the execution, check the contents of registers $s3, $s4, $s5, and $s6.

**Task 3:**

* Load the value of $s1 with 0x87654321.
* Execute the **‘sll’** instruction to shift the bits in $s1 left by 16 positions and store the result in $s2.
* Execute the **‘srl’** instruction to shift the bits in $s1 right by 8 positions and store the result in $s3.
* Execute the **‘sra’** instruction to shift the bits in $s1 right by 12 positions and store the result in $s4.
* After the execution, check the contents of registers $s2, $s3, and $s4.

**Task 4:**

* Ask the user to enter a single alphabetic character and read the character input from the user.
* Determine whether the input character is uppercase or lowercase. For uppercase characters ('A' to 'Z') have ASCII values from 65 to 90. For lowercase characters ('a' to 'z') have ASCII values from 97 to 122.
* If the character is uppercase, convert it to lowercase by adding 32 to its ASCII value (e.g., char + 32).
* If the character is lowercase, convert it to uppercase by subtracting 32 from its ASCII value (e.g., char - 32).
* Display the newly converted character back to the user, showing the change in case.
* End the program.

**Task 5:**

* Ask the user to enter an integer number.
* Ask the user to enter a bit position (between 0 and 31).
* Check if the entered bit position is within the valid range (0 to 31). f the bit position is invalid, display an error message and exit the program or prompt for a valid input.
* Perform a right shift operation **‘>>’** on the integer by the specified bit position.
* Use a bitwise AND operation **‘&’** with 1 to isolate the value of the bit (e.g., (integer >> bit\_position) & 1).
* Output the value of the extracted bit to the user.
* End the program.

**Task 6:**

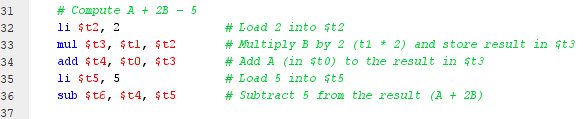
* Ask the user to enter a signed integer number.
* Since the multiplication involves a floating-point number (24.5), convert the integer input to a floating-point representation.
* Perform the multiplication of the converted floating-point number by 24.5.
* Output the value of the multiplication.
* End the program.

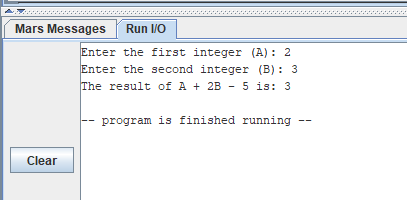
**Task 7:**

* Ask the user to enter an unsigned integer number.
* Create a variable to hold the result after the bit swapping.
* Use a loop or a series of operations to iterate over the bits of the input number.
* Identify odd and even bits. Even bits can be extracted using a bitwise AND with a mask (e.g., 0xAAAAAAAA for positions 1, 3, 5, ...). Odd bits can be extracted using a bitwise AND with a mask (e.g., 0x55555555 for positions 0, 2, 4, ...).
* For even bits, shift the even bits to the right by one position to move them to odd positions. For odd bits, shift the odd bits to the left by one position to move them to even positions.
* Use a bitwise OR operation to combine the shifted even and odd bits into the result variable.
* Display the result.
* End of program.

**Results and Analysis:**

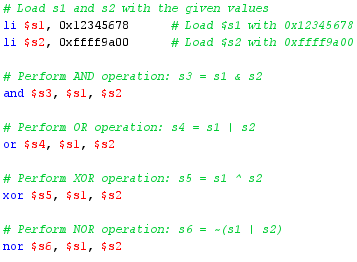
***Task 1:***

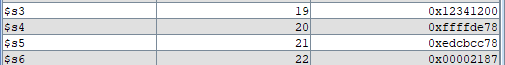
******

**

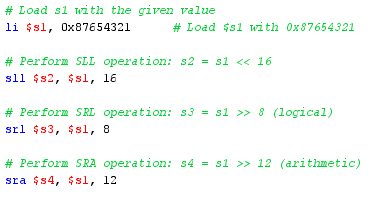
* *To compute the expression A + 2B − 5, I utilized a combination of* ***multiplication****,* ***addition****, and* ***subtraction*** *instructions. First, compute 2B then add A to 2B. Finally, deduct 5 from the result of A + 2B.*

***Task 2:***

******

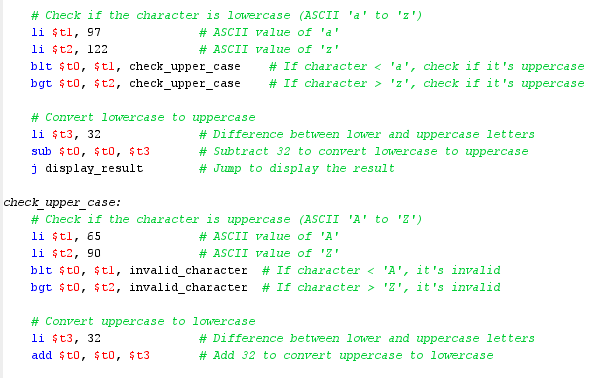
**

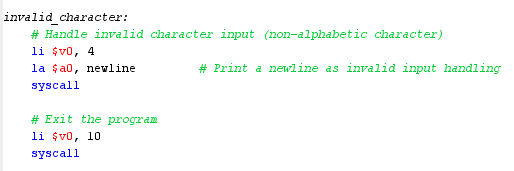
***Task 3:***

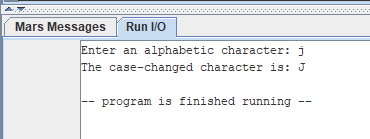
******

**

***Task 4:***

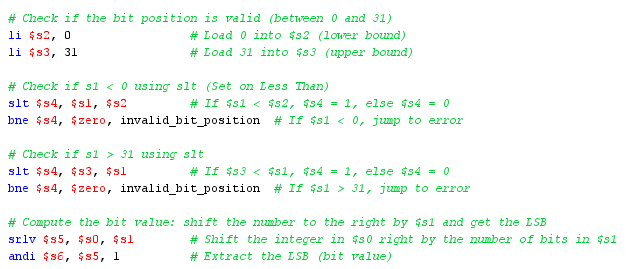
******

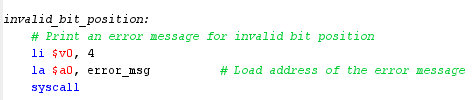
******

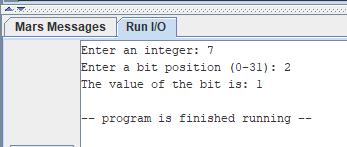
**

* *For character case conversion, I utilized branch instructions (blt and bgt) to determine whether a character is lowercase or uppercase.*
* *I used* ***‘blt’*** *to check if the character's ASCII value is less than 97 (the start of lowercase letters). If true, it’s not lowercase.*
* *I used* ***‘bgt’*** *to check if it’s greater than 122 (the end of lowercase letters). If true, it’s not lowercase.*
* *If the character is lowercase, I converted it to uppercase by subtracting 32 from its ASCII value.*
* *If it’s uppercase (ASCII value between 65 and 90), I converted it to lowercase by adding 32.*

***Task 5:***

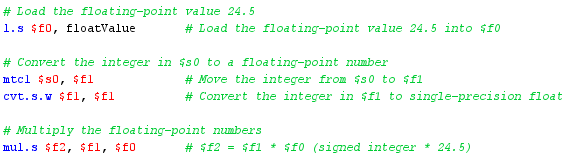
******

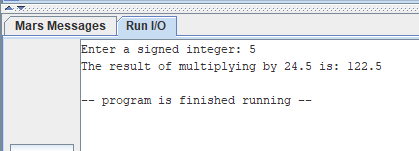
******

******

* *To ensure the bit position was valid, I utilized the* ***‘slt’*** *(set on less than) instruction to verify that it fell within the range of 0 to 31.*
* *For example, if the user inputs 7 and specifies bit position 2, I first validated that the bit position was within the range of 0 to 31 using the slt instruction. After validating the bit position, I shift the input number (7) to the right by the specified number of positions (2) using the* ***srlv*** *(Shift Right Logical Variable) instruction. To determine the value of the bit at position* ***2****, I then use the* ***andi*** *(AND Immediate) instruction with the value 1. In this case, the LSB is 1, indicating that the bit at position 2 in the original number is also 1.*

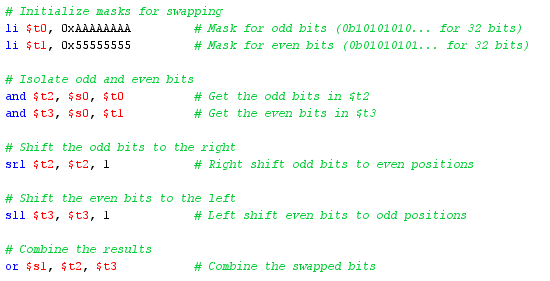
***Task 6:***

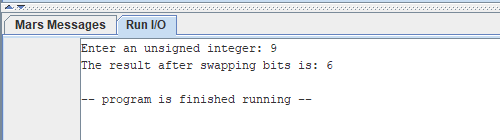
******

******

* *In my program, I utilized the* ***mul.s*** *instruction to perform multiplication with floating-point numbers. The* ***mtc1*** *instruction was used to transfer an integer value to a floating-point register, while* ***cvt.s.w*** *converted the integer to single-precision floating-point format. This combination enabled accurate arithmetic operations involving floating-point calculations.*

***Task 7:***

******

******

* *To swap bits in my program, I used masking along with* ***‘srl’*** *(shift right logical) and* ***‘sll’*** *(shift left logical) instructions to isolate and reposition the bits. For instance, when the user inputs the unsigned integer 9, the program performs the necessary bit manipulations, resulting in the output of 6. This approach effectively exchanges the bits at odd and even positions.*

**Conclusion:**

* I learned how to manipulate registers, perform arithmetic and logical operations, and work with floating-point numbers.
* This activity helped meto manipulate individual bits in a number.
* I also learned how to convert data types (e.g., integers to floating-point) and understand the implications of these conversions in terms of precision and representation.
* I became familiar with basic MIPS integer arithmetic and logic instructions, including:
* **Integer Addition and Subtraction**
* **Bitwise Logic Instructions:** Understanding these instructions allowed me to perform operations like AND, OR, XOR, and NOR.
* **Shift Instructions:** I learned how to effectively use shift instructions to modify bit positions within integers.

**Additional Notes/Observations:**

* When dealing with floating-point arithmetic, I observe precision and rounding issues. To solve this, I used appropriate floating-point instructions for operations such as mul.s, mtc1 and cvt.s.w.