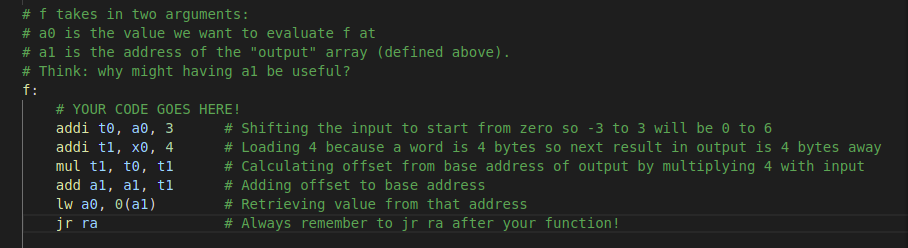
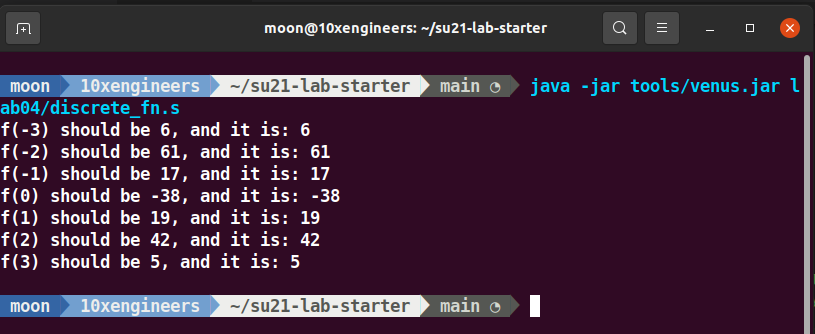
**Lab 4: RISC-V Functions, Pointers**

**Exercise 1: Write a function without branches**

Here is my code of function f without using branches and jumps:



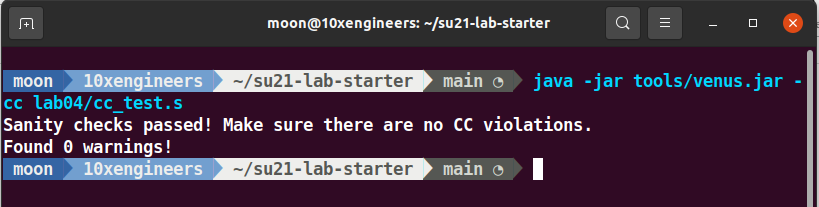
Its output is:



……………………………………………………………………………………………………....

**Exercise 2: Calling Convention Checker**

I have fixed all the errors. Output of the code after resolving errors and running code is:



**Action Items**

1. **What caused the errors in simple\_fn, naive\_pow, and inc\_arr that were reported by the Venus CC checker?**

* **simple\_fn:** The error was “Usage of unset register t0” on the line: mv a0, t0. It means that t0 register was previously unset in the code and does not contain a valid value. So moving the value of t0 to a0 may result in unpredictable behavior of the program. Also mv a0, t0 was unnecessarily used and it had no impact on the functionality of the function. So simply commenting or removing that line removed the warning that was shown by Venus CC checker.
* **naive\_pow:** The error was **“**Setting of a saved register (s0) which has not been saved!” on the line: li s0, 1. Here naïve\_pow is acting as a callee function of the main (i.e caller function). In RISCV convention, it is assumed that callee will not modify the saved registers. If callee had to then it must have to push those registers in stack first, then modify those saved registers then at the end retrieve back the value from the stack of the saved registers. But here s0 was not push to stack and retrieve back. By preserving the value of s0 using stack, the error is removed
* **inc\_arr:** Samecase was with inc\_arr as of naïve\_pow. Here s0 and s1 were not keep preserved by using stack. So both of these registers are then saved to stack then at the end their value is retrieved which results in the removal of the error.

1. **In RISC-V, we call functions by jumping to them and storing the return address in the ra register. Does calling convention apply to the jumps to the naive\_pow\_loop or naive\_pow\_end labels?**

j offset is a pseudo instruction of jal x0, offset. While calling naïve\_pow\_loop or naïve\_pow\_end label we are storing the return address to x0 register means we are discarding the return address because there is no need of return address. So calling conventions do not apply here. The calling convention does not apply to jumps within the same function, like the ones in this code.

1. **Why do we need to store ra in the prologue for inc\_arr, but not in any other function?**

In inc\_arr, the instruction jal helper\_fn is used, while jal is not used in any other functions. This means that main is calling inc\_arr (label that is not in main) and then inc\_arr is calling helper\_fn(label that is not in inc\_arr). So we need to save the return address of inc\_arr before entering to helper\_fn orherwise we will lost the return address of inc\_arr because the return address of helper\_fn will overwrite it.

Note that there is no such situation in any other function where jal is used means we are jumping outside of that function.

1. **Why wasn't the calling convention error in helper\_fn reported by the CC checker? (Hint: it's mentioned above in the exercise instructions.)**

Because it will only look for bugs in functions that are exported with the .globl directive. In the code provided helper\_fn is not exported by .globl directive while simple\_fn, naïve\_pow and inc\_arr are exported with .globl directive.

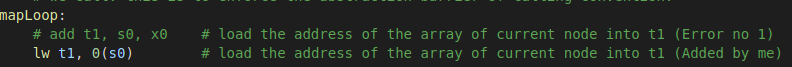
……………………………………………………………………………………………………....

**Exercise 3: Debugging**

**Find the six mistakes inside the map function in megalistmanips.s. Read all of the commented lines under the map function in megalistmanips.s and make sure that the lines do what the comments say**

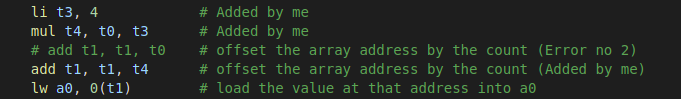
**Mistake 1:**

We have to load the address of the array of current node to t1. But in the provided code we were just moving the value of s0 to t1. By using lw, I have load the address of s0 to t1 not value.

****

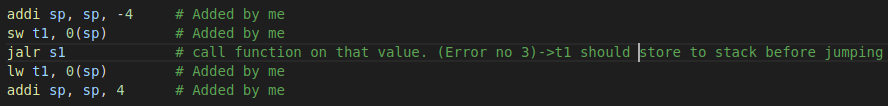
**Mistake 2:**

As stated in the comment that add statement is calculating the offset. But it was calculating offset by 1 byte. While the memory is word addressable so we have to add a offset of 4 bytes. Corrected version of the code is shown below:

****

**Mistake 3:**

It is the responsibility of caller function to preserve the temporary registers that it has used before calling the callee. While in the code provided mapLoop is using t1 but it is not preserving t1 before calling mystery. And the mystery function changed the value of t1. So we have to push t1 to stack before jumping and after return from that function we have to restore the value of t1. The corrected patch of code is shown below:

****

**Mistake 4:**

There was a syntax error which was infact reported by venus. la statement requires the label name next but here its syntax was wrong. Also there is no need to use la we had to load the address of s0 not the label so lw will be used.

****

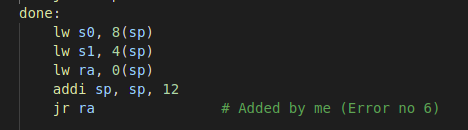
**Mistake 5:**

The comment says that we had to put address of function back to a1. Note that s1 already contains the address of function. Using lw will load the value that is stored at the address stored in s1. While we want only the value stored in s1. So, add statement will be used as:

****

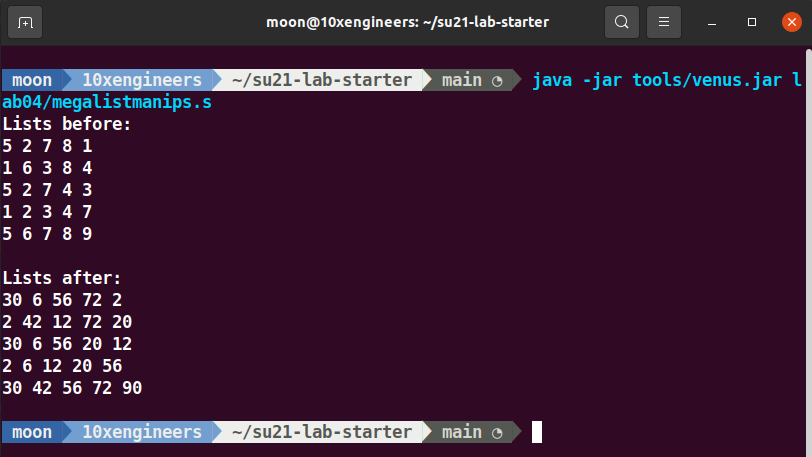
**Mistake 6:**

jr ra is not used to return back from where we had jump. So adding jr ra, the code is shown below:

****

**Output of the Program:**

After resolving all the errors, the output of the program is shown as:



……………………………………………………………………………………………………....

**Exercise 4: Finding and solving the bugs**

* Find the bugs in four of the five accumulators
* For each broken accumulator, write a test that fails on the broken one, but passes the correct implementation.

**Corner testcase:**

a0 = {0,1,2,3,4,5,6,7,0} is the corner test case which will be only satisfied by the correct accumulator function and failed by all other functions.

1. **accumulatorone:**
2. **Error:** Not passed corner case (if 0 appeared at the start of the array the test failed). Also it is the duty of callee to push the saved\_registers to stack if callee had to use them and then restore those registers to their original value. But that condition is not tested.
3. **Test that failed:** Adding the following code will test that condition.

#-----------Test Condition accumulatorone-----------

# This condition will only failed by accumulatorone while passed by all other accumulators

la a0 inputarray

li s0, 10 # Added by me

jal accumulatorone

li t0, 10 # Added by me

bne t0, s0, Fail # Added by me

li t0 28

beq a0 t0 Pass

This checks that if the value of s0 before and after calling the accumulator function is same or altered. **Note that:** only accumulatorone fails this condition while all other satisfies because they are not using s0 without preserving it.

1. **accumulatortwo:**
2. **Error:** Not passed corner case (if 0 appeared at the start of the array the test failed). Also proper stack management practice is not used. sp is first incremented and at the end sp is decremented. As our stack is LIFO, so we should first decrement the stack and then increment it. The code is working fine but if some other code modifies sp (following the correct practice) or writes to the memory location s0 then it is possible that we get the wrong result in s0.
3. **Test that failed:** I have added the following lines of code to check whether there is first decrement and then increment in stack or not.

# -----------Test Condition accumulatortwo-----------

# This condition will only failed by accumulatortwo while passed by all other accumulators

addi sp, sp -8 # Added by me

la a0 inputarray

li t4, 15 # Added by me

li t5, 30 # Added by me

sw t4, 0(sp) # Added by me

sw t5, 4(sp) # Added by me

jal accumulatortwo

lw t6, 0(sp) # Added by me

lw t0, 4(sp) # Added by me

addi sp, sp, 8 # Added by me

bne t4, t6, Fail # Added by me

bne t5, t0, Fail # Added by me

li t0 28

beq a0 t0 Pass

**Note that:** only accumulatortwo fails this condition while all other satisfies because others are following the LIFO management in their functions properly.

1. **accumulatorthree:**
2. **Error:** Not passed corner case (if 0 appeared at the start of the array the test failed).
3. **Test that failed:** I have added another array named inputarray\_other to test this function and this function does not give the correct result.

.data

inputarray\_other: .word 0,1,2,3,4,5,6,7,0

.text

# -----------Test Condition accumulatorthree----------

# This condition will be failed by accumulatorthree

la a0 inputarray\_other

jal accumulatorthree

li t0 28

beq a0 t0 Pass

1. **accumulatorfour:**
2. **Error**: Not passed corner case (if 0 appeared at the start of the array the test failed). Also the problem with accumulatorfour is that it is using t2 without initializing it to some value. So it is possible that “add t2 t2 t1” may result in a garbage value as t2 is not initialized so its value may be taken as garbage.
3. **Test that failed:** To test this we can initialize t2 to some value (i.e 10 in this code). Thus the result will be greater than the expected result by a value of 10 because t2 was not initialized in the code.

la a0 inputarray

li t2 10 # Added by me

jal accumulatorfour

li t0 28

beq a0 t0 Pass

**Note that:** only accumulatorfour fails this test while all other result in correct output as they are not using t2 uninitialzed.

1. **accumulatorfive:**
2. **Error**: There is no error in this function.
3. **Test that failed:** If all the above tests are combined. It will pass that test too. Combining all test yield:

# -----------Combining all Test Cases----------------

# This combine condition will only passed by accumulatorfive while failed by all other accumulators

la a0 inputarray\_other

li s0, 10 # Added by me (for accumulatorone)

addi sp, sp -8 # Added by me (for accumulatortwo)

li t4, 15 # Added by me (for accumulatortwo)

li t5, 30 # Added by me (for accumulatortwo)

sw t4, 0(sp) # Added by me (for accumulatortwo)

sw t5, 4(sp) # Added by me (for accumulatortwo)

li t2 10 # Added by me (for accumulatorfour)

jal accumulatorfive

li t0, 10 # Added by me (for accumulatorone)

bne t0, s0, Fail # Added by me (for accumulatorone)

lw t6, 0(sp) # Added by me (for accumulatortwo)

lw t0, 4(sp) # Added by me (for accumulatortwo)

addi sp, sp, 8 # Added by me (for accumulatortwo)

bne t4, t6, Fail # Added by me (for accumulatortwo)

bne t5, t0, Fail # Added by me (for accumulatortwo)

li t0 28

beq a0 t0 Pass

**Note:** accumulatorfive only passes this test.

……………………………………………………………………………………………………....