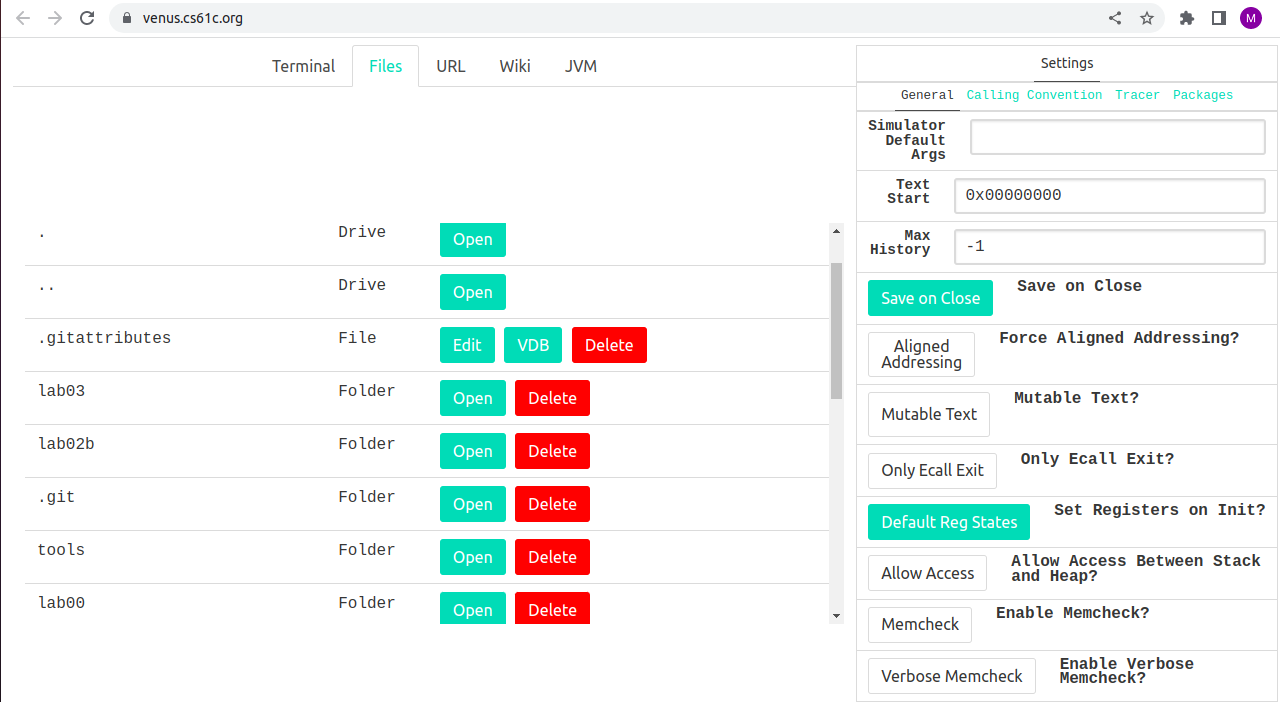
**Lab 3 RISC-V Assembly**

**Exercise 1: Connecting your files to Venus**

I have connected my files to venus you can see a screenshot below:



………………………………………………………………………………………………………

**Exercise 2: Familiarizing yourself with Venus**

**1):** What do the .data, .word, .text directives mean (i.e. what do you use them for)? Hint: think about the 4 sections of memory.

**.data:** The .data directive is used to declare the data segment, which is a section of memory that contains initialized data. This segment is usually read-write, meaning that the program can modify the data stored there.

**.word:** The .word directive is used to allocate space in memory for one or more 32-bit words (4 bytes) and initialize them with specific values. In the example code provided, the .word directive is used to define an array of four integers: 2, 4, 6, and 8.

**.text:** The .text directive is used to declare the code segment, which is a section of memory that contains the program's executable code. This segment is usually read-only, meaning that the program cannot modify the instructions stored there.

**2):** Run the program to completion. What number did the program output? What does this number represent?

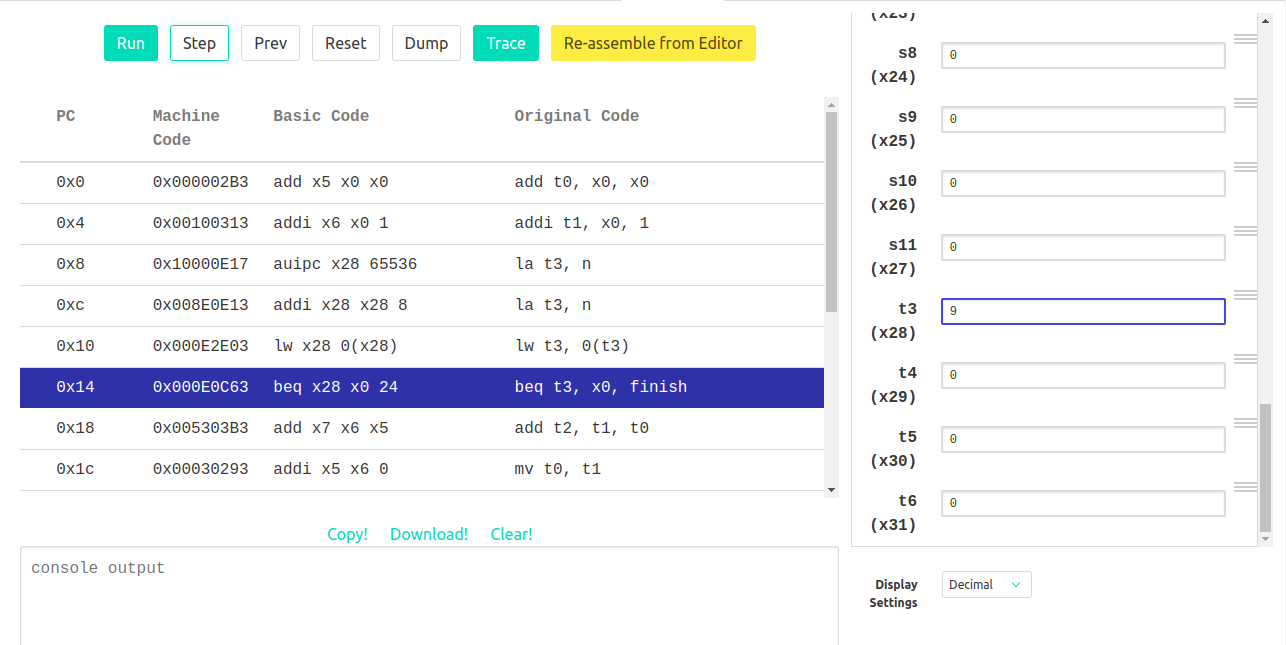
The program prints **34**. This is because the loop runs 9 times as the value of n is 9 and prints the 9th number of Fibonacci sequence i.e 34.

**3):** At what address is n stored in memory? Hint: Look at the contents of the registers

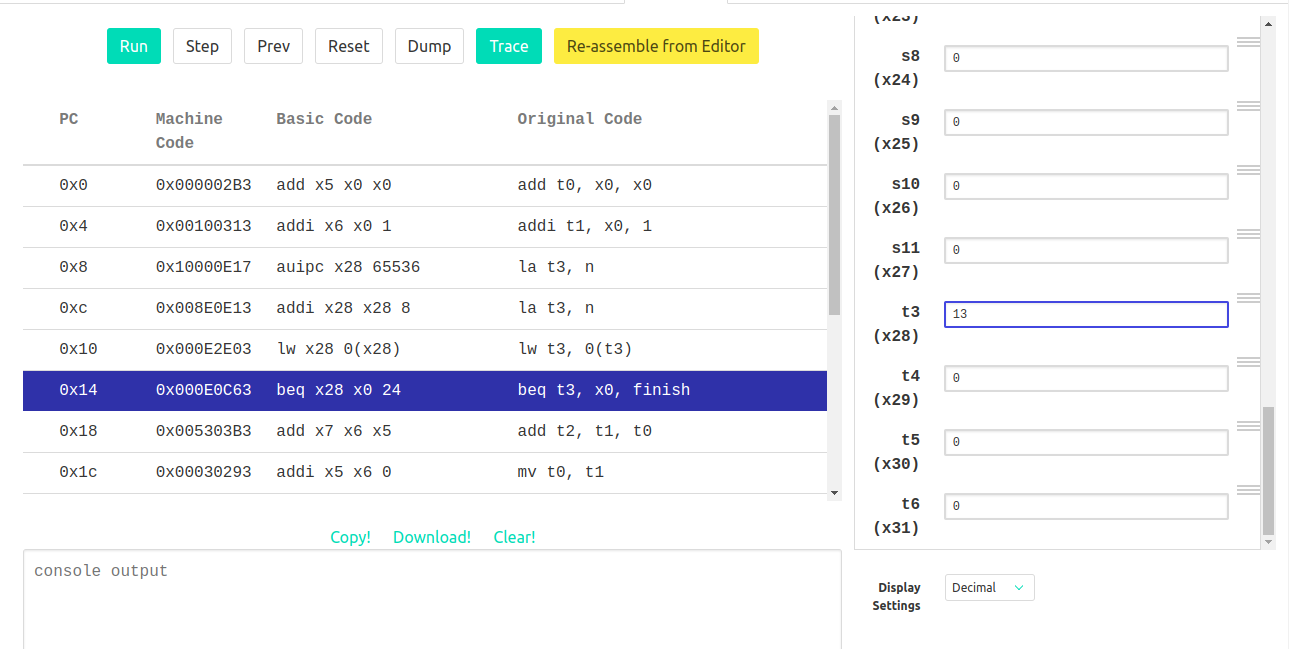
n is 9 that is stored at memory address 0x10000010

**4):** Without actually editing the code (i.e. without going into the "Editor" tab), have the program calculate the 13th fib number (0-indexed) by manually modifying the value of a register. You may find it helpful to first step through the code. If you prefer to look at decimal values, change the "Display Settings" option at the bottom.

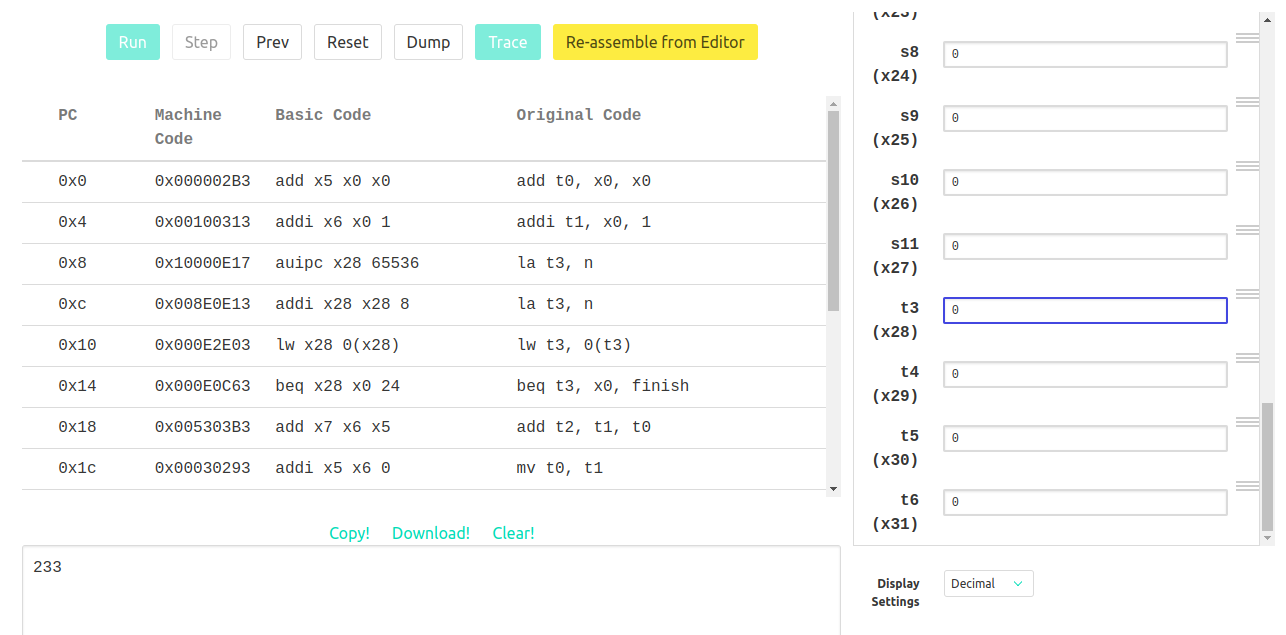
I continue stepping until t3 gets the value 9 as you can see in the figure below



Then I manually changed the value of t3 to 13 as shown:



Then running the code, I get 233 which is the 13th number of Fibonacci sequence.



………………………………………………………………………………………………………

**Exercise 3: Translating from C to RISC-V**

Find and identify the following components of this assembly file, and be able to explain how they work.

● The register representing the variable k: **t0**

The purpose of t0 in this code is to act as the loop counter and keep track of the current index being processed in the source array.

● The register representing the variable sum: **s0**

Before the loop starts, s0 is initialized to zero with the instruction addi s0, x0, 0. Then, during each iteration of the loop, the result of fun(source[k]) is stored in dest[k], and added to sum with the instruction add s0, s0, t2. After the loop ends, the final value of sum is stored in the a0 register with the instruction add a0, x0, s0.

● The registers acting as pointers to the source and dest arrays: **s1 for source, s2 for destination**

Index k is used to calculate the offset into the source and dest arrays for the current element being processed. The offset is calculated by multiplying k by 4 with the instruction slli s3, t0, 2, and then adding the result to the base address of the corresponding array using the add instructions.

● The assembly code for the loop found in the C code:

In the loop firstly we shift the value of t0 to the left by 2 bits in order to calculate byte offset. Then it loads the value of the current element in the source array to register t2. After this it checks if the current element is equal to 0. If so, exits the loop. Then sets the argument of the fun function to the current element in the source array. Also it reserves 8 bytes of memory on the stack by adjusting the value of the stack pointer and stores the value of t0 (the loop counter) on the stack so that it can be restored later. It also stores the value of t2 (the current element of the source array) on the stack so that it can be restored later. Then jump to the function fun. After that restore t0 and t2 from stack. Then stores the result of the operation on the current element to the corresponding element in the dest array. And then increments the loop.

● How the pointers are manipulated in the assembly code:

Pointers are manipulated by the following instructions:

**la s1, source:** This loads the address of the source array into register s1, effectively making s1 a pointer to the beginning of the array.

**la s2, dest:** This loads the address of the dest array into register s2, making s2 a pointer to the beginning of the array.

**slli s3, t0, 2:** This left-shifts the value of t0 by 2 bits (equivalent to multiplying t0 by 4), and stores the result in s3. This is used to calculate the offset for accessing elements in the arrays.

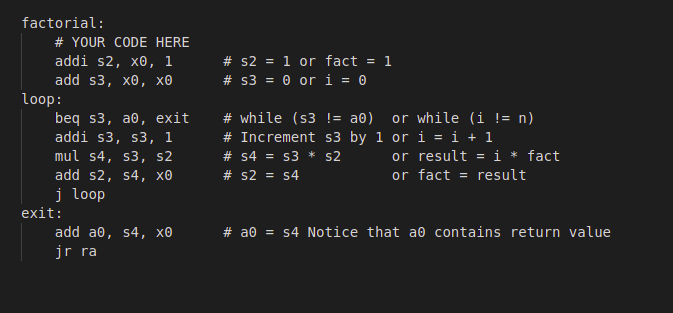
**add t1, s1, s3:** This adds the value of s3 to the value of s1, resulting in a pointer to the k-th element of the source array.

**add t3, s2, s3:** This adds the value of s3 to the value of s2, resulting in a pointer to the k-th element of the destination array.

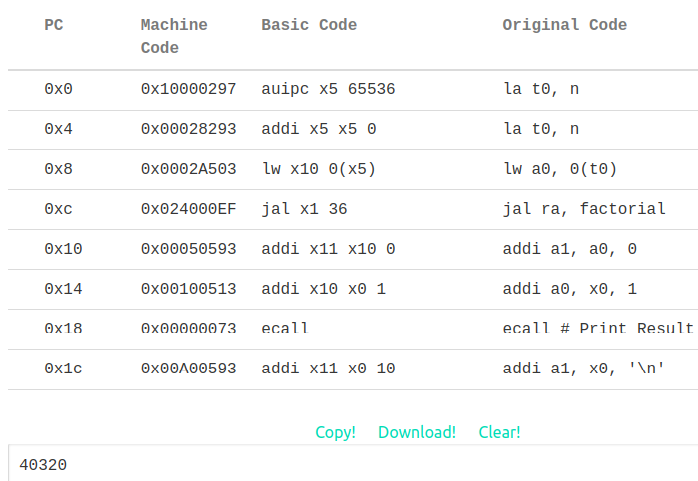
………………………………………………………………………………………………………

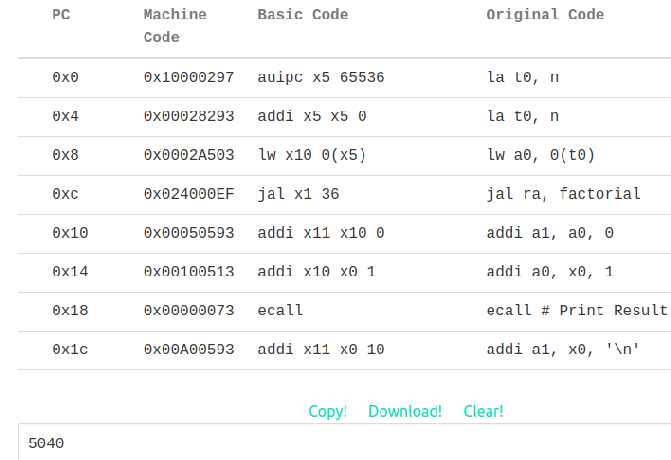
**Exercise 4: Factorial**

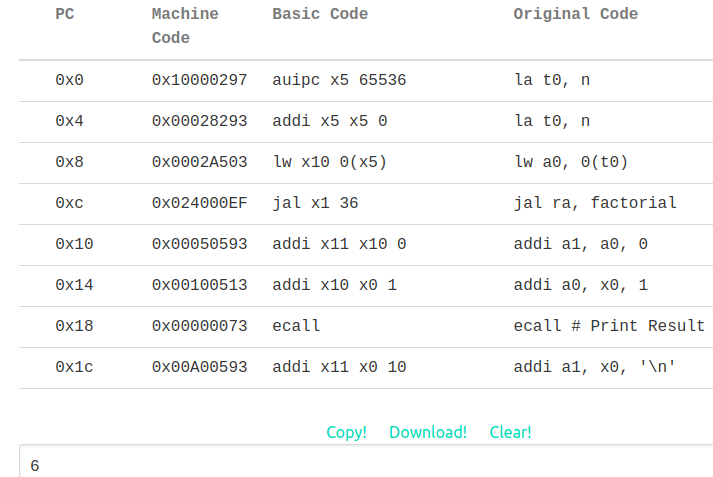
Below is my assembly code for factorial function and each line is explained by a comment following it.



Output of this code with n = 8; n=7; n=3 using the venus online version is:







Output of this code with n = 8; n=7; n=3 by testing locally with venus is:



………………………………………………………………………………………………………

**Exercise 5: RISC-V function calling with map**

I have filled all the ten markers and I have also answer the (why) reason in the comments where asked. Complete code is not shown below. Only the ten markers are shown below

# Load the address of the "square" function into a1 (hint: check out "la" on the green sheet)

### YOUR CODE HERE ###

la a1, square

# Load the address of the "decrement" function into a1 (should be very similar to before)

### YOUR CODE HERE ###

la a1, decrement

map:

# Prologue: Make space on the stack and back-up registers

### YOUR CODE HERE ###

addi sp, sp, -12

sw s0, 0(sp)

sw s1, 4(sp)

sw ra, 8(sp)

# Load the value of the current node into a0

# THINK: Why a0?

# --because a0 is the register where function arguments are typically passed and we want the value of current node to pass as an argument

### YOUR CODE HERE ###

lw a0, 0(s0)

# Call the function in question on that value. DO NOT use a label (be prepared to answer why).

# Hint: Where do we keep track of the function to call? Recall the parameters of "map".

# -- because the function to be called is not known at compile time. Instead, it is passed as a parameter to the map function and stored in the a1 register.

### YOUR CODE HERE ###

jalr ra, a1, 0

# Store the returned value back into the node

# Where can you assume the returned value is? -- It is in a0

### YOUR CODE HERE ###

sw a0, 0(s0)

# Load the address of the next node into a0

# The address of the next node is an attribute of the current node.

# Think about how structs are organized in memory.

### YOUR CODE HERE ###

lw a0, 4(s0)

# Put the address of the function back into a1 to prepare for the recursion

# THINK: why a1? What about a0?

# -- because a0 is already being used to hold the value of the current node that we want to pass as an argument to the function.

### YOUR CODE HERE ###

add a1, x0, s1

# lw a1, 0(s1)

# Recurse

### YOUR CODE HERE ###

jal ra, map

done:

# Epilogue: Restore register values and free space from the stack

### YOUR CODE HERE ###

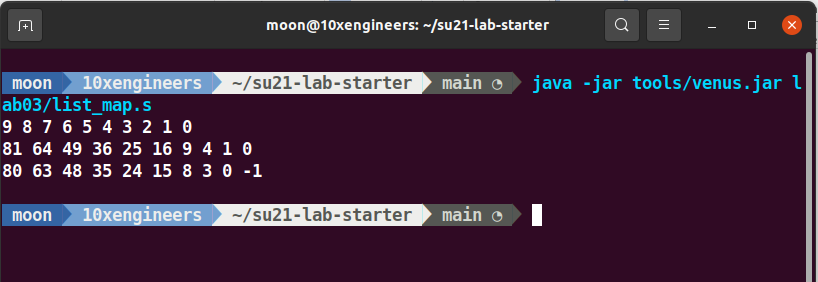
lw s0, 0(sp)

lw s1, 4(sp)

lw ra, 8(sp)

addi sp, sp, 12

Output of the code is:



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