**Module: R2: Intro to RISC-V Assembly**

**Section:** RISC-V ISA **Task:** RISC-V Instruction Formats

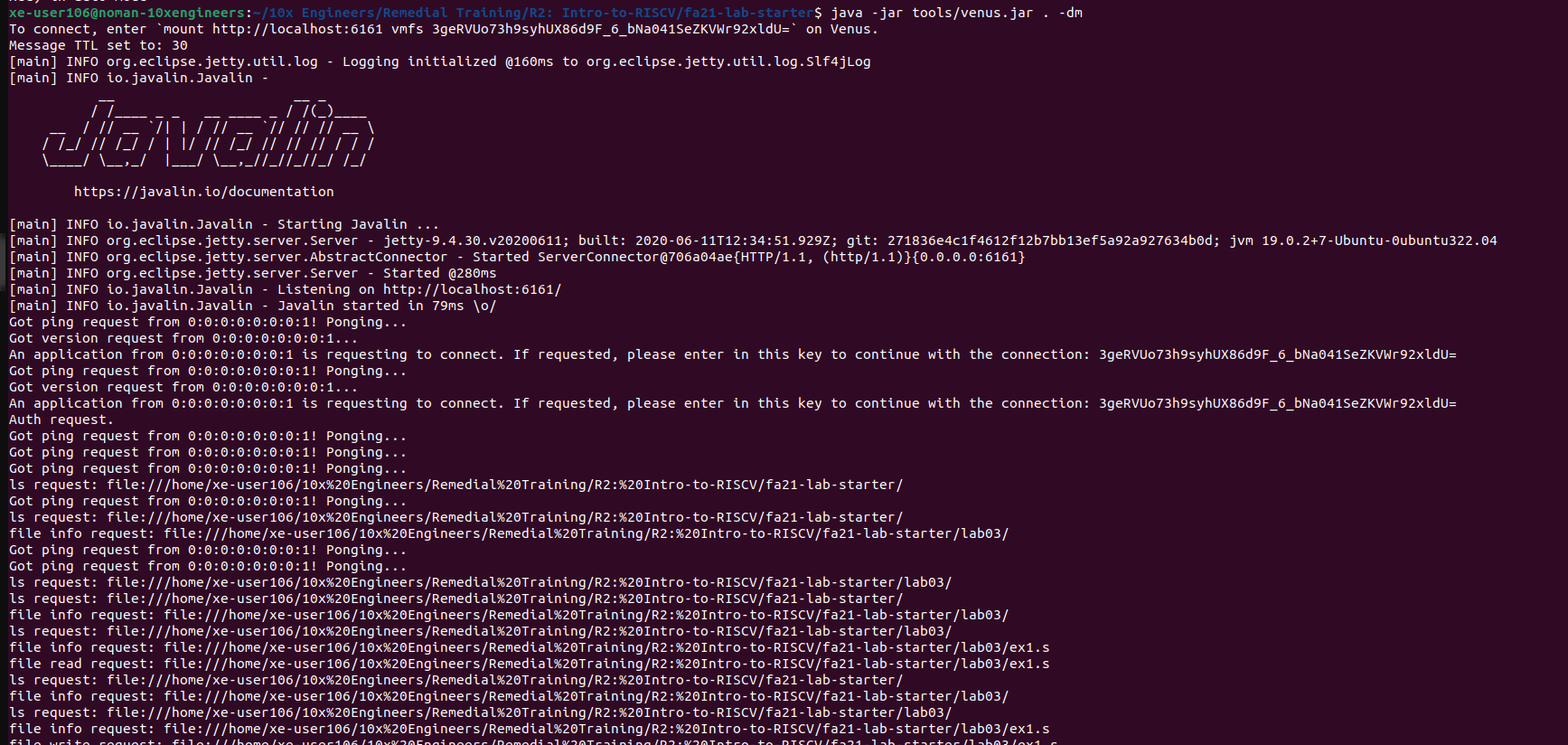
**LAB 03 -** <https://github.com/ImNomanCR7/fa21-lab-starter.git>

**RISC-V Instruction Formats**

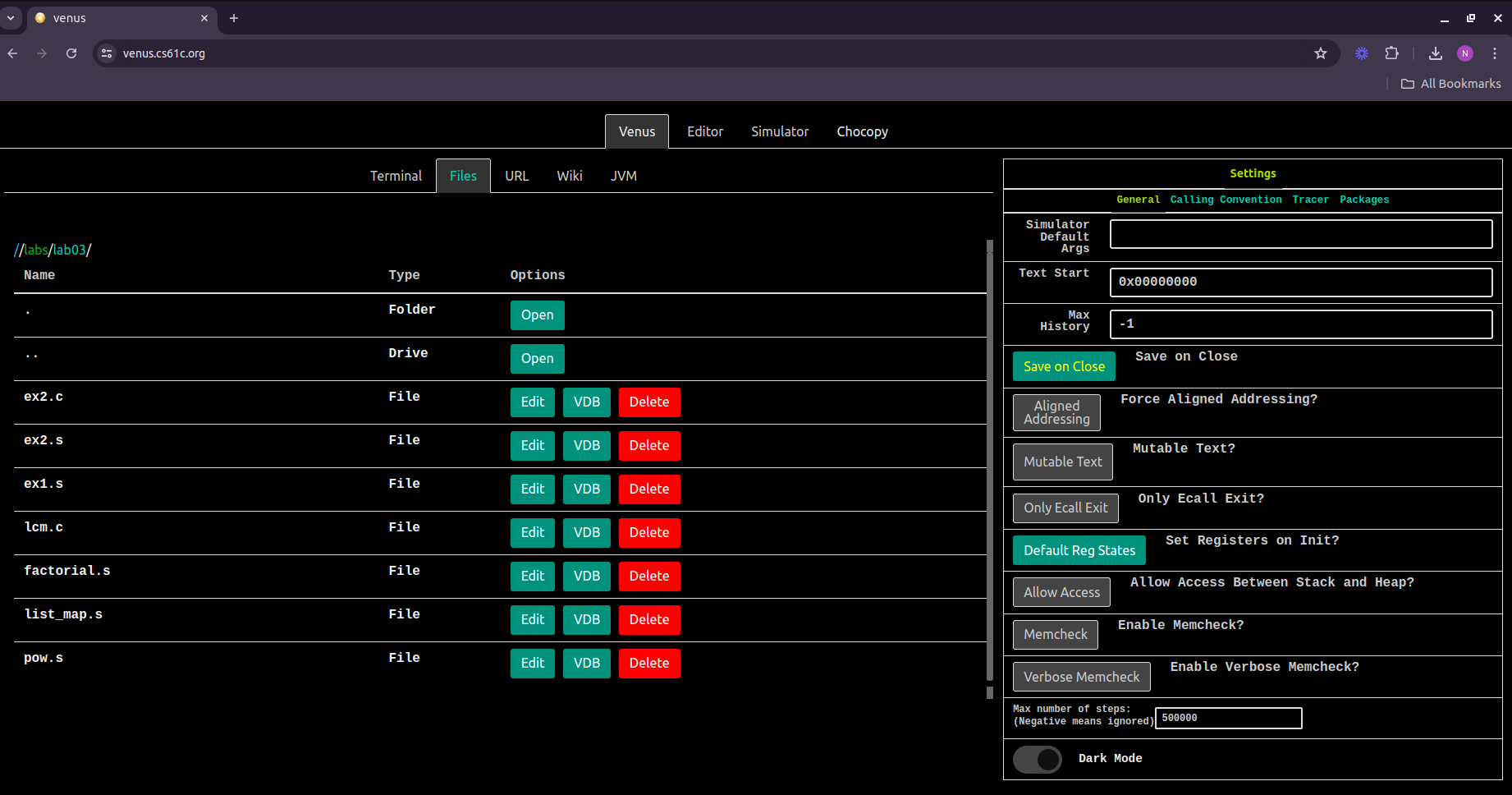
**Exercise 1: Connecting Files to Venus**

* + **Code Snippet:**

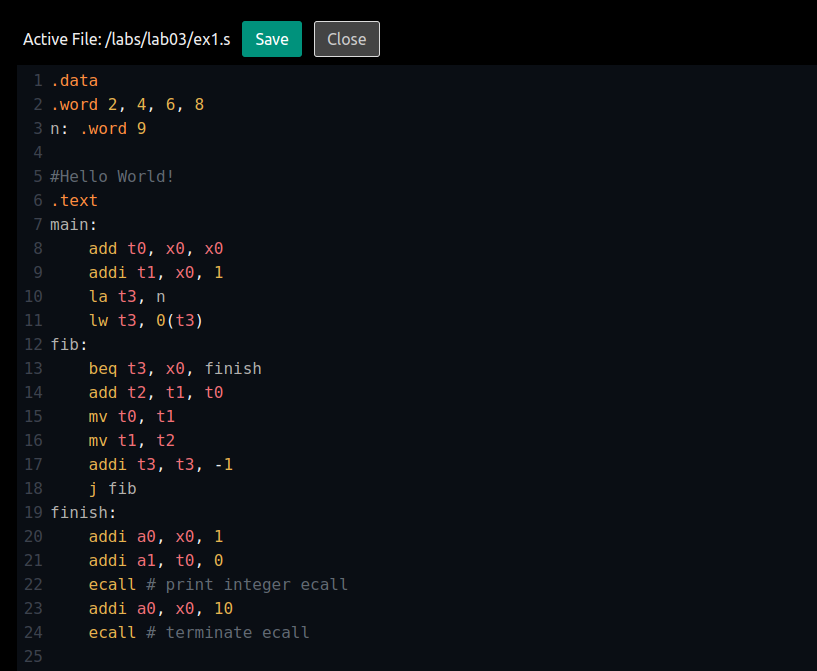
venus .jar tools/venus.jar . -dm



**Mounted Local Labs**



**Made an Edit to ex1.s:**

****

**Exercise 2: Familiarizing with Venus**

**‭i - Directives:‬**

I‭n RISC-V assembly language programming, directives like "**‬‭.data‬‭**", "**‬‭.word‬‭**", and "**‬‭.text‬‭**"‬

‭are used to organize and specify different sections of memory within a program. The‬

‭"**‬‭.data‬‭**" directive defines the data section, where ‬‭variables and constants are initialized.‬

‭The "**‬‭.word‬‭**" directive, typically within the "**.data**"‬‭section, allocates space for 32-bit‬

‭words of data. The "‬‭**.text‬‭**" directive defines the text ‬‭section, containing executable code‬

‭such as instructions and function definitions.

‬

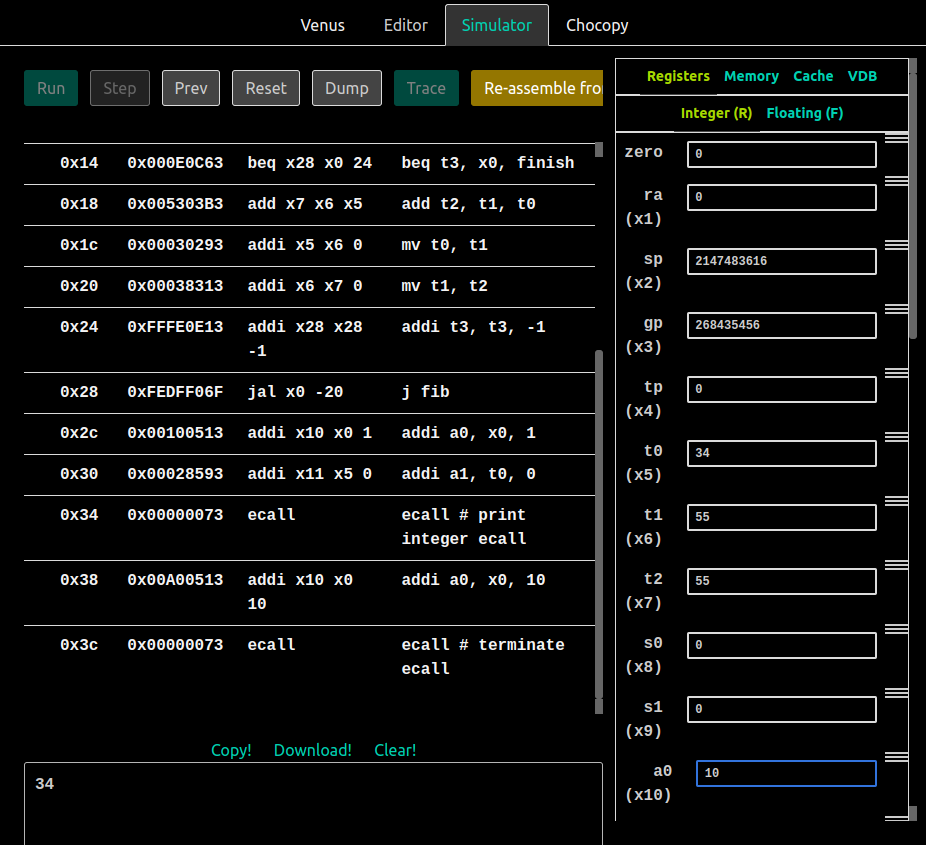
‭Together, these directives help structure the program's memory layout, ensuring data‬

and code are appropriately separated and initialized for execution.‬

**ii - Output of the program:‬**

‭The program outputs the value 34. This number represents the 10th element of the‬

‭Fibonacci series.‬



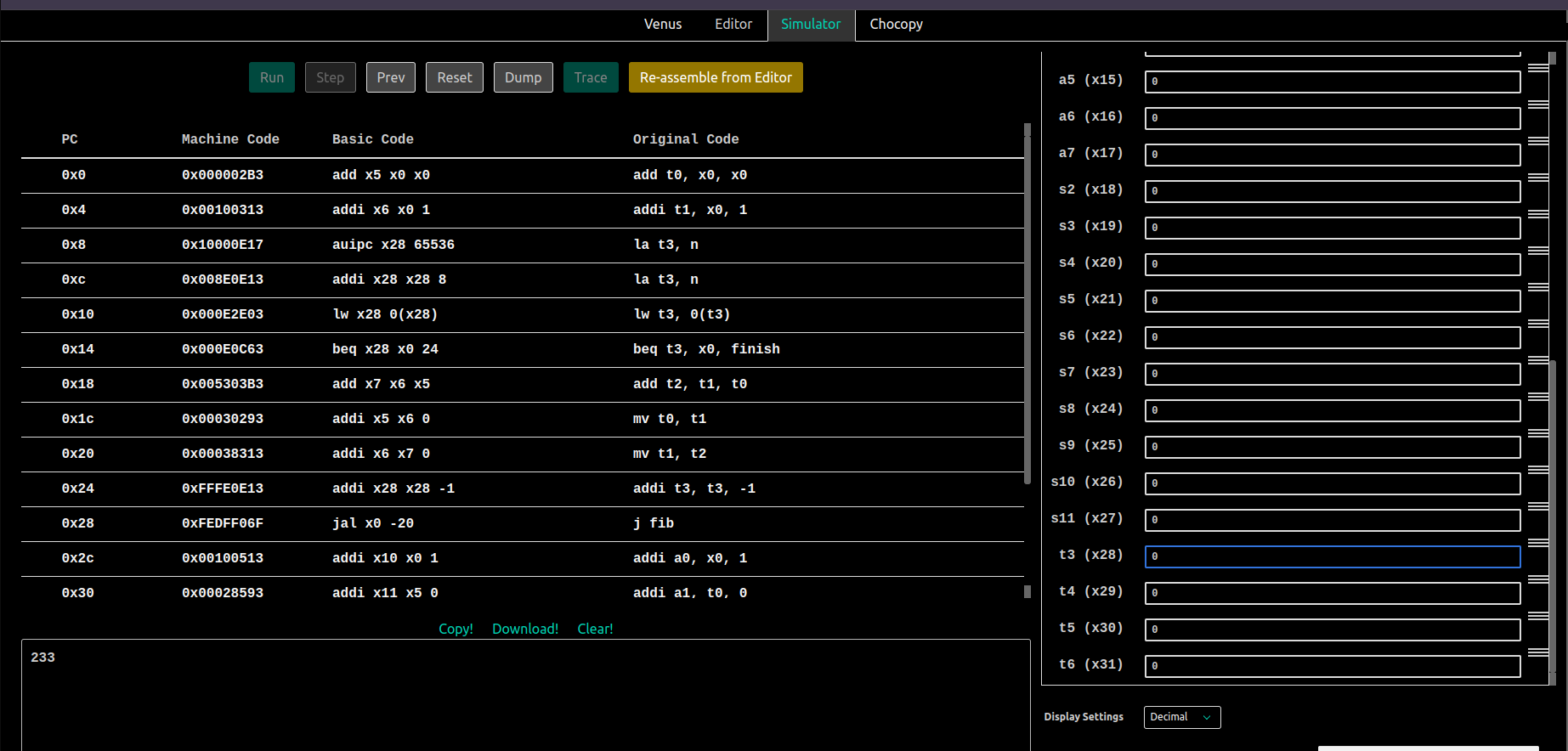
**‭iii - Address of n:**‬

‭The address of n is **‬‭0x10000010‬‭** as shown by the screenshot.



**iv - 13th fib number:‬**

‭Edited the value of **‬‭t3‬** ‭to 12 to print the 13th Fibonacci ‬‭number in series.‬



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

1. **The register representing the variable k:** ‬‭Register‬‭ “‭**t‬0‬‭**” is used to represent the‬ variable “**k**”. It is initialized to‬‭ 0 (“**‬‭addi t0, x0,‬‭0‭**”‬ ) before the loop starts and is‬

incremented within the loop (“**addi t0, t0, 1**”).

1. ‭**‭The register representing the variable sum:** ‬‭Register‬‭ “‭**s0‬‭**” is used to represent‬

‭the variable “**sum**”. It is initialized to 0 (**‬‭addi s0,‬‭x0, 0‬‭**) before the loop starts and‬

is updated within the loop (**‬‭add s0, s0, t2‬‭**).‬

1. **‭The registers acting as pointers to the source and dest arrays:** ‬‭Registers‬ “**‬‭s1‬‭**” and “**‬‭s2‬‭**” are used as pointers to the “**‬‭source‬‭**”‬‭and “**‬‭dest‬‭” arrays**”‬ respectively. They are loaded with the addresses of the arrays (“**‬‭la s1, source‬‭**”‬ ‭and “**‬‭la s2, dest‬‭**”) before the loop starts, and they‬‭ are manipulated within the loop‬ ‭to access elements of the arrays (**‬‭add t1, s1, s3** ‬‭and‬‭ **add t3, s2, s3‬‭**).‬
2. **‭The assembly code for the loop found in the C code:** ‬‭It starts with a label‬ (**‬‭loop:**‬‭) and ends with a branch instruction (**‬‭jal x0,‬‭loop‬‭**). Inside the loop,‬ ‭elements of the “**‬‭source‬‭**” array are loaded (**‬‭lw t2,‬‭0(t1)**‬‭), and the loop continues‬ ‭until a 0 is encountered in the “**‬‭source‬‭**” array (**‬‭beq ‬‭t2, x0, exit‬‭**).‬
3. **‭How the pointers are manipulated in the assembly code:** ‬‭The pointers to the‬ source‬‭ and‬‭ dest ‬‭arrays (**‬‭s1 ‬‭and ‬‭s2‬‭**) are manipulated‬‭ using arithmetic operations.‬ ‭Within the loop, the index “**‬‭t0‬‭**” is left-shifted by‬‭2 (**‬‭slli s3, t0, 2‬‭**) to calculate the offset for accessing elements of the arrays. Then, this offset is added to the base‬ ‭address of each array (**‬‭add t1, s1, s3‬** ‭and‬‭ **add t3,‬‭s2, s3‬‭**) to obtain the address‬ ‭of the current element in each array.‬

**Exercise 4: Factorial of a Number**

* + **Code Snippet:**

#Author: Noman Rafiq

#Dated: 30 June, 2024

#Description: The program uses a variable n to calculate the factorial of that number.

.globl factorial

.data

n: .word 8

.text

main:

la t0, n

lw a0, 0(t0)

jal ra, factorial

addi a1, a0, 0

addi a0, x0, 1

ecall # Print Result

addi a1, x0, '\n'

addi a0, x0, 11

ecall # Print newline

addi a0, x0, 10

ecall # Exit

factorial:

# YOUR CODE HERE

addi t0, x0, 1 #factorial = t0 = 1

loop:

mul t0, t0, a0 #factorial = 1 \* n

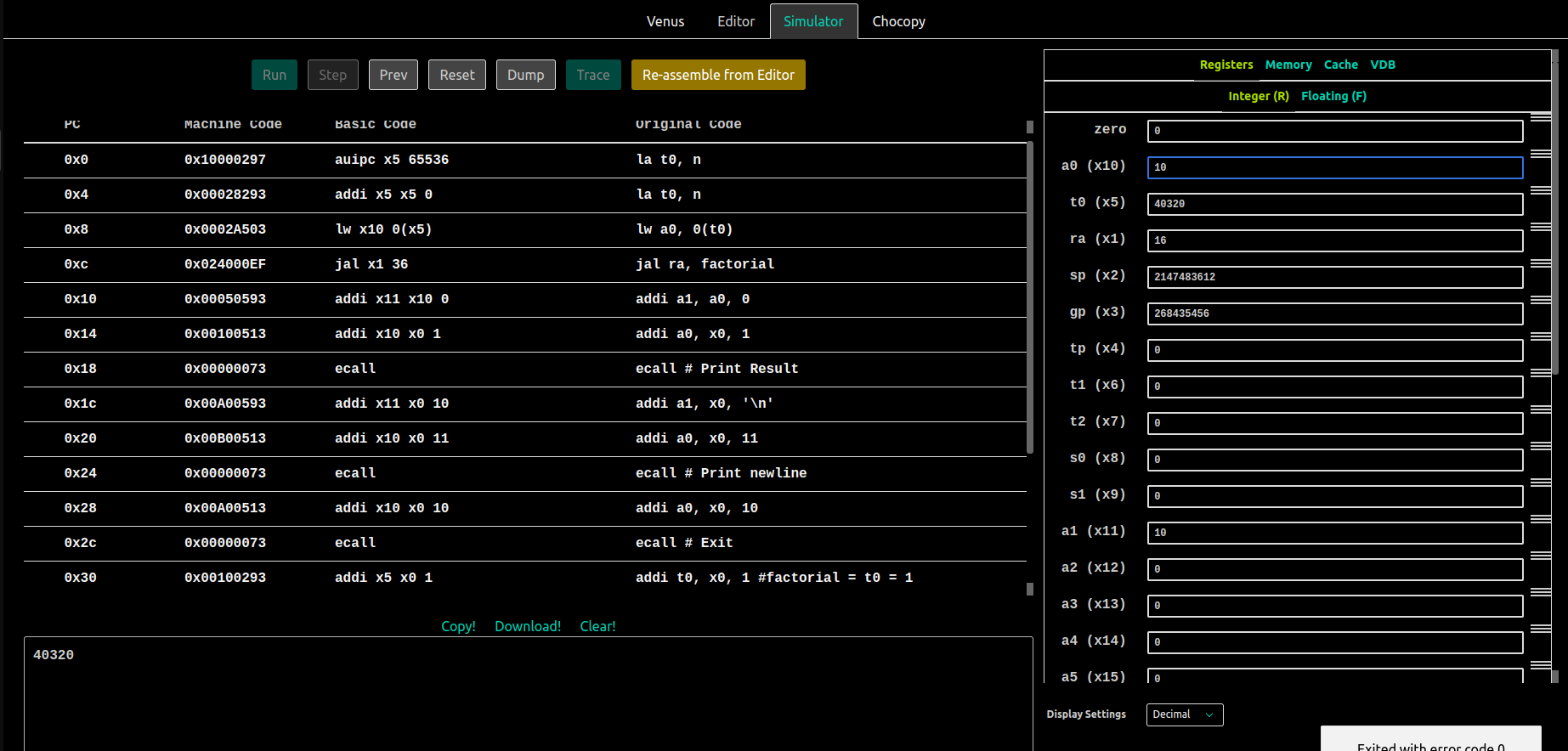
addi a0, a0, -1 # n = n - 1

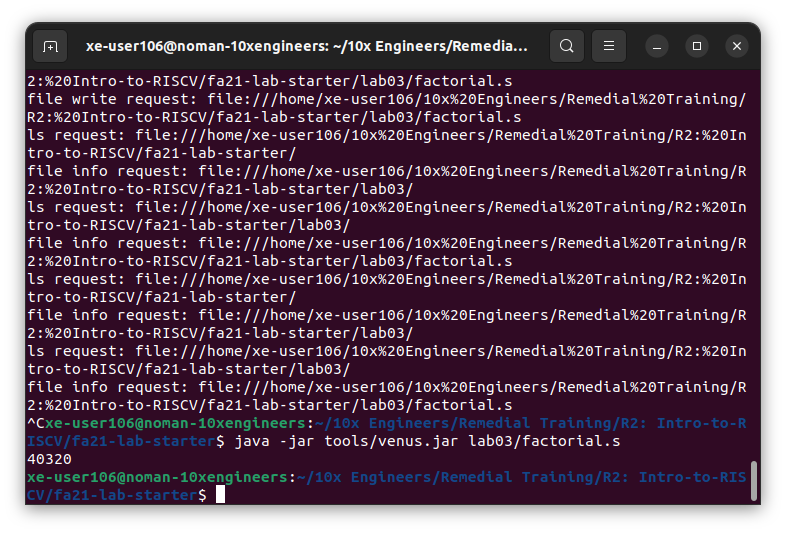
bne a0, x0, loop

mv a0, t0

jr ra

* + **Output:**

****

****

**Exercise 5: Function Calling with Map**

* + **Code Snippet:**

#Author: Noman Rafiq

#Dated: 30 June, 2024

#Description: The program uses a map function that performs an in-place upate of a linked-list by applying a called function to all the nodes in the list.

.globl map

.text

main:

jal ra, create\_default\_list

add s0, a0, x0 # a0 (and now s0) is the head of node list

# Print the list

add a0, s0, x0

jal ra, print\_list

# Print a newline

jal ra, print\_newline

# === Calling `map(head, &square)` ===

# Load function arguments

add a0, s0, x0 # Loads the address of the first node into a0

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

### YOUR CODE HERE ###

la a1, square

# Issue the call to map

jal ra, map

# Print the squared list

add a0, s0, x0

jal ra, print\_list

jal ra, print\_newline

# === Calling `map(head, &decrement)` ===

# Because our `map` function modifies the list in-place, the decrement takes place after

# the square does

# Load function arguments

add a0, s0, x0 # Loads the address of the first node into a0

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

### YOUR CODE HERE ###

la a1, decrement

# Issue the call to map

jal ra, map

# Print decremented list

add a0, s0, x0

jal ra, print\_list

jal ra, print\_newline

addi a0, x0, 10

ecall # Terminate the program

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)

beq a0, x0, done # If we were given a null pointer (address 0), we're done.

add s0, a0, x0 # Save address of this node in s0

add s1, a1, x0 # Save address of function in s1

# Remember that each node is 8 bytes long: 4 for the value followed by 4 for the pointer to next.

# What does this tell you about how you access the value and how you access the pointer to next?

# Load the value of the current node into a0

# THINK: Why a0?

### 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".

### YOUR CODE HERE ###

jalr ra, s1, 0

# Store the returned value back into the node

# Where can you assume the returned value is?

### 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?

### YOUR CODE HERE ###

mv a1, 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

jr ra # Return to caller

# === Definition of the "square" function ===

square:

mul a0, a0, a0

jr ra

# === Definition of the "decrement" function ===

decrement:

addi a0, a0, -1

jr ra

# === Helper functions ===

# You don't need to understand these, but reading them may be useful

create\_default\_list:

addi sp, sp, -12

sw ra, 0(sp)

sw s0, 4(sp)

sw s1, 8(sp)

li s0, 0 # Pointer to the last node we handled

li s1, 0 # Number of nodes handled

loop: # do...

li a0, 8

jal ra, malloc # Allocate memory for the next node

sw s1, 0(a0) # node->value = i

sw s0, 4(a0) # node->next = last

add s0, a0, x0 # last = node

addi s1, s1, 1 # i++

addi t0, x0, 10

bne s1, t0, loop # ... while i!= 10

lw ra, 0(sp)

lw s0, 4(sp)

lw s1, 8(sp)

addi sp, sp, 12

jr ra

print\_list:

bne a0, x0, print\_me\_and\_recurse

jr ra # Nothing to print

print\_me\_and\_recurse:

add t0, a0, x0 # t0 gets current node address

lw a1, 0(t0) # a1 gets value in current node

addi a0, x0, 1 # Prepare for print integer ecall

ecall

addi a1, x0, ' ' # a0 gets address of string containing space

addi a0, x0, 11 # Prepare for print char syscall

ecall

lw a0, 4(t0) # a0 gets address of next node

jal x0, print\_list # Recurse. The value of ra hasn't been changed.

print\_newline:

addi a1, x0, '\n' # Load in ascii code for newline

addi a0, x0, 11

ecall

jr ra

malloc:

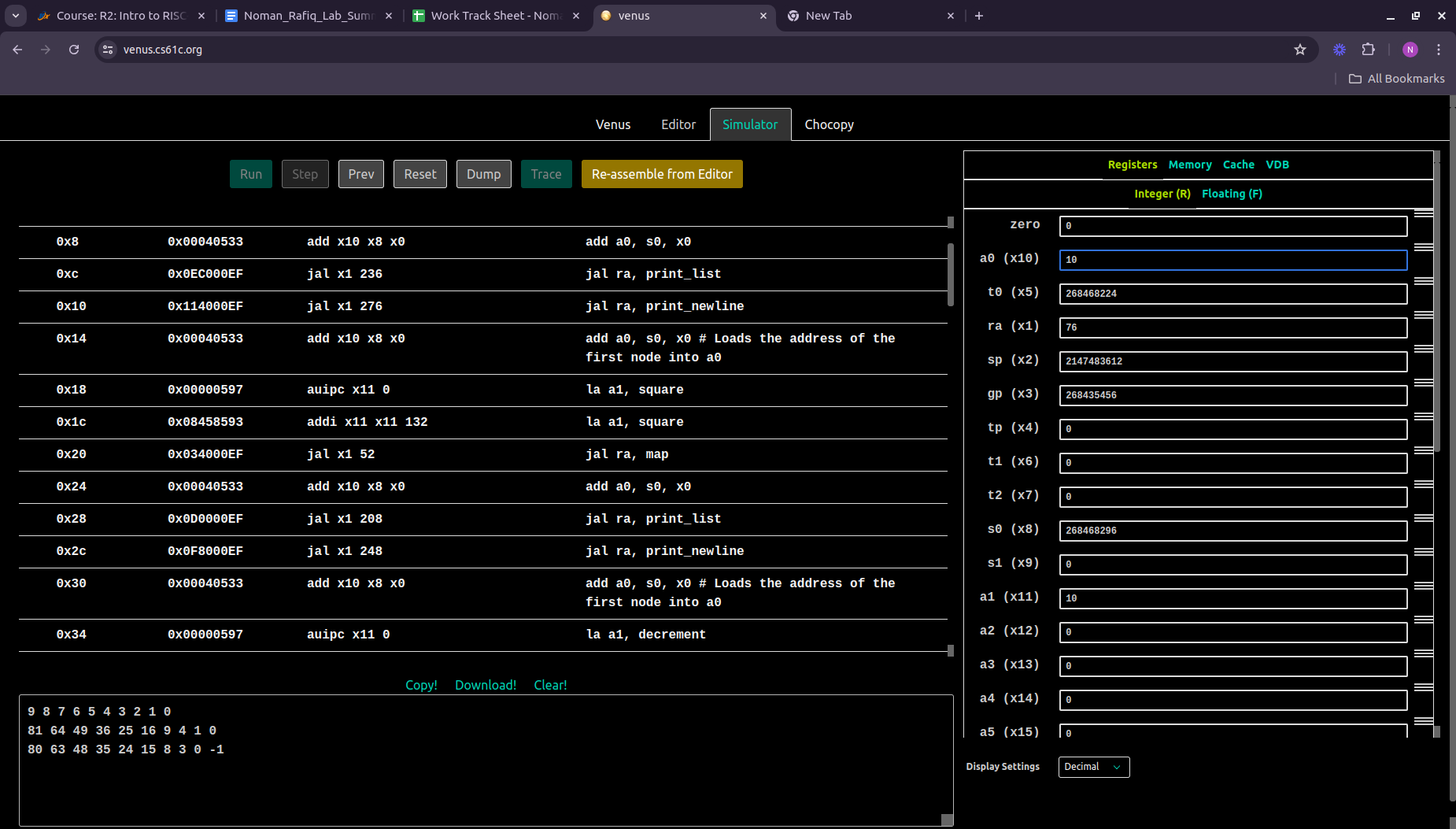
addi a1, a0, 0

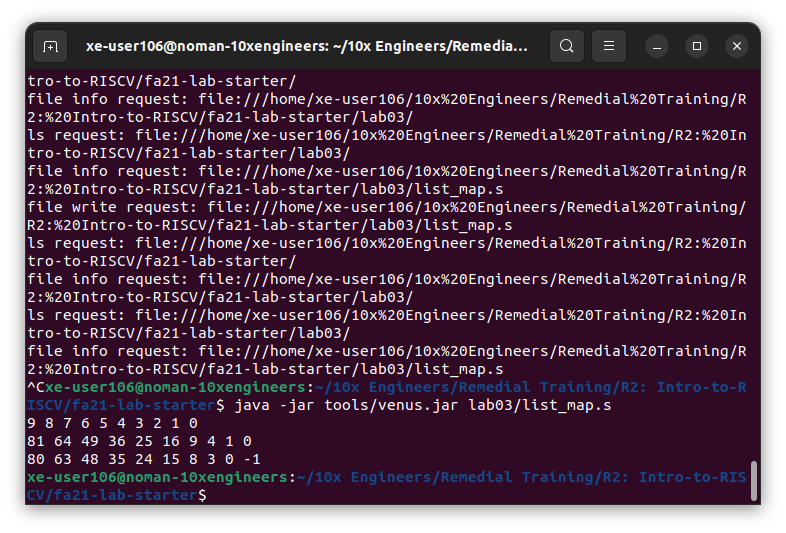
addi a0, x0, 9

ecall

jr ra

* + **Output:**

****

****