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

**Section:** CALL **Task:** Programming Task

**Bare Metal Assembly on Spike -**

**Programming Task**

1. **RISC-V Assembly Code:**
   * **Code Snippet:**

.globl factorial

.text

# start code here

main:

la t0, num # t0 = address of num

lw a0, 0(t0) # a0 = num

jal factorial

j exit

factorial:

addi sp, sp, -8

sw ra, 0(sp)

sw a0, 4(sp)

li t1, 1

bgt a0, t1, else

li a0, 1

lw ra, 0(sp)

addi sp, sp, 8

jr ra

else:

addi a0, a0, -1

jal factorial

lw t2, 4(sp)

lw ra, 0(sp)

addi sp, sp, 8

mul a0, t2, a0

jr ra

exit:

add a1, a0, x0

li a0, 1

ecall

li a0, 10

ecall

# end code here

write\_tohost:

li x1, 1

sw x1, tohost, t5

j write\_tohost

.data

# start data section here

num: .word 8

result: .word 1

# end data section here

.align 12

.section ".tohost","aw",@progbits;

.align 4; .global tohost; tohost: .dword 0;

.align 4; .global fromhost; fromhost: .dword 0;

* + **Linker Script:**

#Author: Noman Rafiq

#Dated: July 3, 2024

SECTIONS

{

. = 0x80000000;

.text : { \*(.text) }

. = 0x80001000;

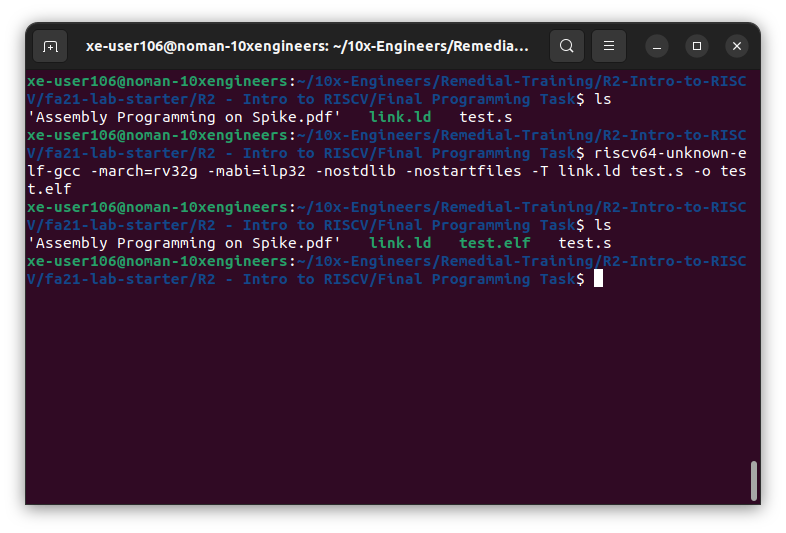
.data : { \*(.data) }

\_end = .;

}

Run this command:

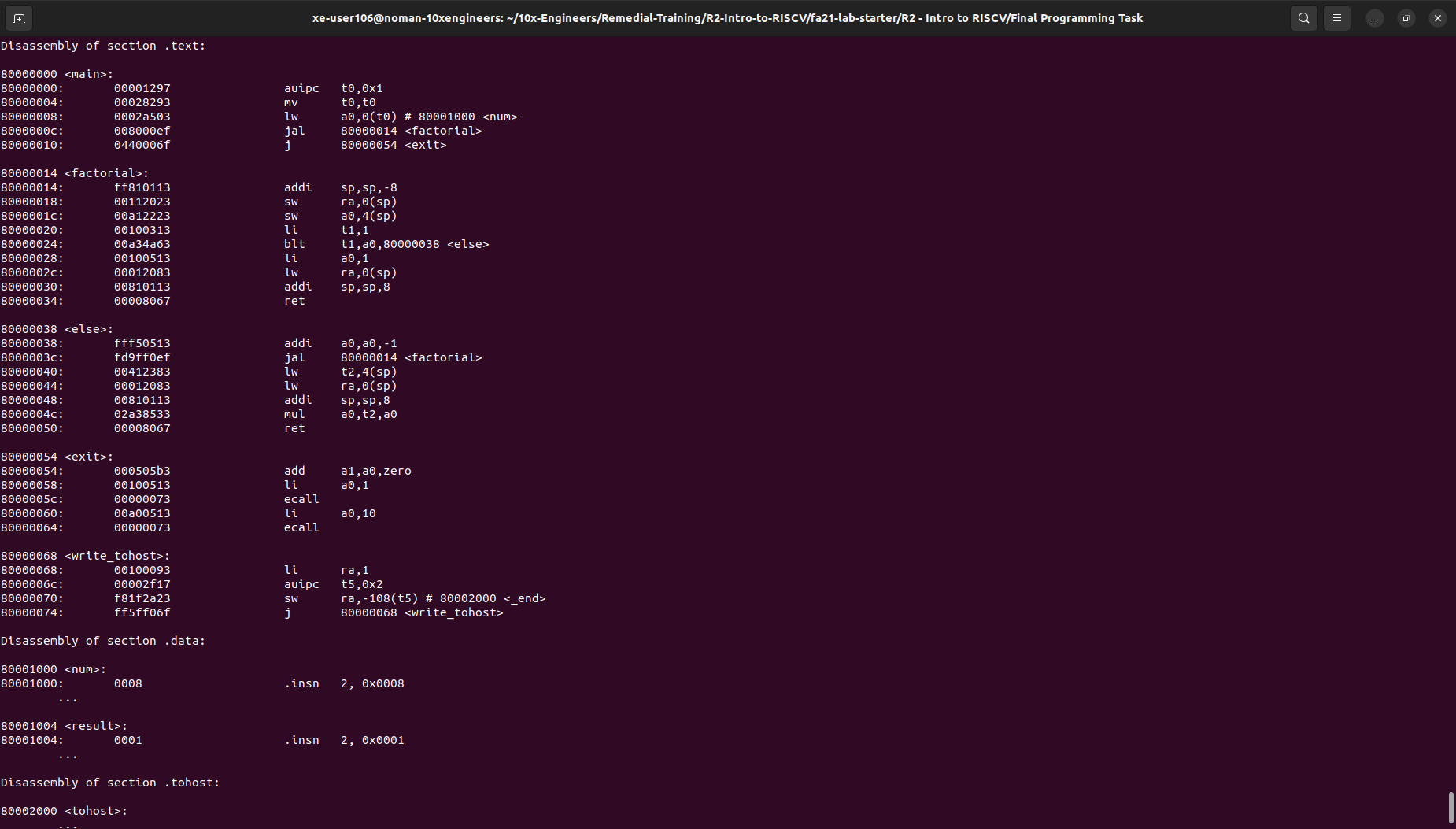
riscv64-unknown-elf-gcc -march=rv32g -mabi=ilp32 -nostdlib -nostartfiles -T link.ld test.s -o test.elf



In order to check dis-assembly file, use this command:

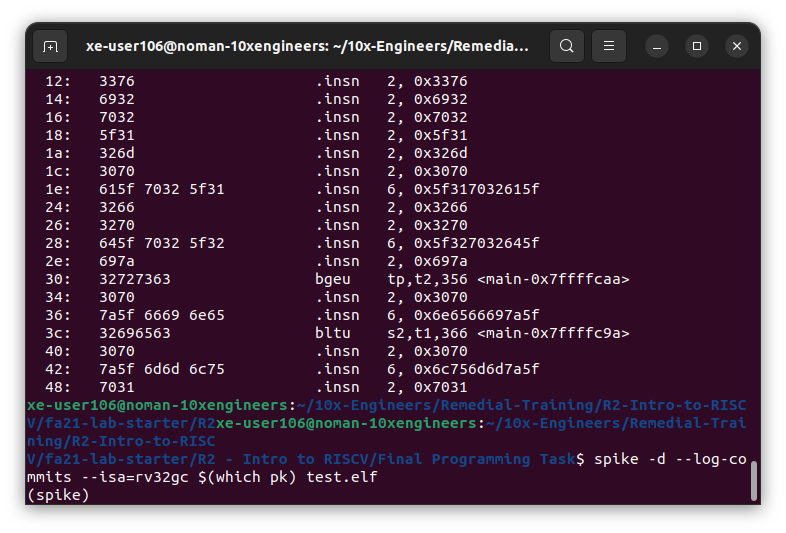
riscv64-unknown-elf-objdump -D test.elf

In the screenshot below, the dis-assembly verifies that out **.text** section started from 0x80000000 and the **.data** section started from 0x80001000



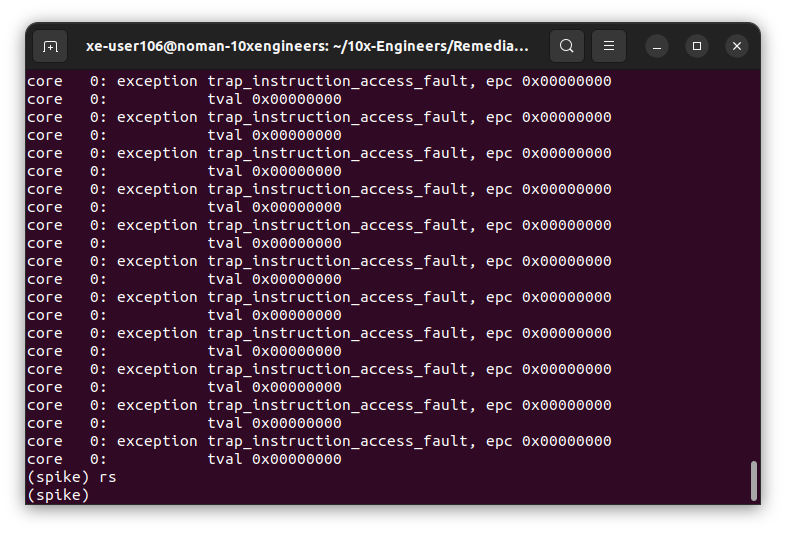
Now, we can manually check the register to verify the output. We can use the following‬ ‭command to go to the debug mode.

spike -d --log-commits --isa=rv32gc $(which pk) test.elf

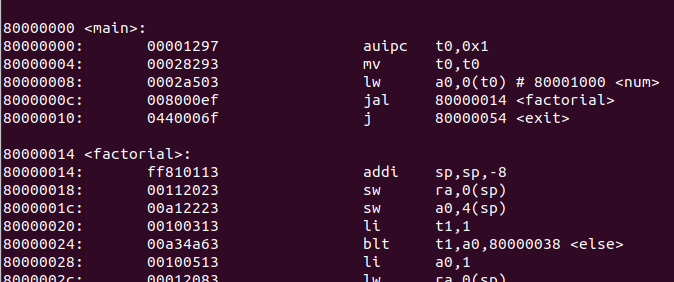


Now, we can manually check the register to verify the output. We can use the following‬ ‭command to go to the debug mode.

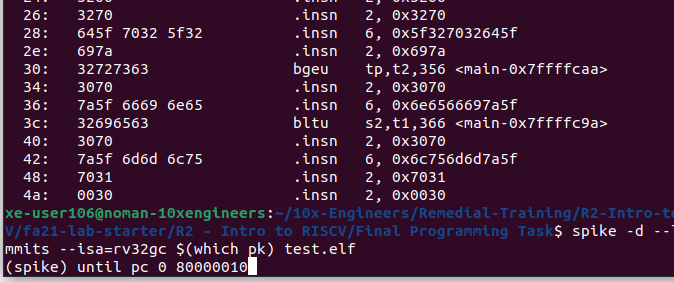
rs



We’ll add a breakpoint to verify the output of our program. The output for **8! = 40320,** hence the output should be 40320. In our dis-assembly file, we can see that the function is returning the **a0** (factorial) value at PC = 0x80000010, hence we’ll add a break point to our execution until this **PC.**

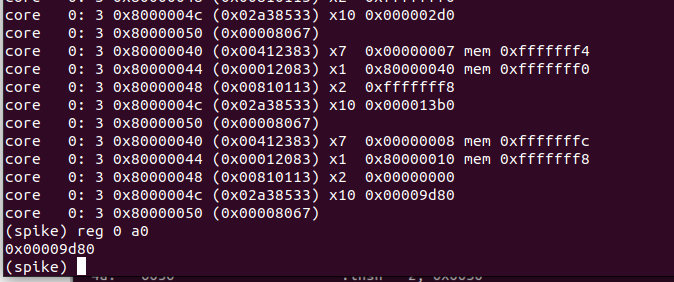
****

Adding a breakpoint at 0x80000010.



After that, we’ll view the contents of the‬ ‭registers to check values, use the following command:

reg 0 a0



The value at this location is 0x00009d80 which is equal to 40320 in Decimal. We can convert the value back to decimal to‬ ‭check for any wrong value. The value of the a0 in our case is 0x00009d80 which is‬ ‭equal to 40320 (decimal) i.e.‬‭factorial for number 8(8! = 40320).

**3. Conclusion:‬**

‭We have successfully compiled and executed the assembly code with custom‬

‭linker script using RISC-V GNU toolchain for 32-bit architecture, without any‬

‭warnings or errors.‬