**CALL Programming Task**

[**https://github.com/Its-Masoom/su21-lab-starter/tree/main/lab04/CALL\_prog\_task**](https://github.com/Its-Masoom/su21-lab-starter/tree/main/lab04/CALL_prog_task)

1. **Write the equivalent RISC-V assembly program to calculate the the factorial of a number recursively. Note that you are not allowed to use non-recursive logic to calculate the factorial. Use stack and RISC-V calling convention appropriately to successfully implement the logic.**

Below is my code for factorial using recursion:

.data

num: .word 5 # 5 can be changed to desired number to calculate factorial

result: .word 0 # Initialized result to zero

.text

main:

la a0, num # load address of num

lw a0, 0(a0) # load value of num i.e 5

la a2, result # load the address of result to a2

jal fact # jump to fact and store return address to ra

sw a1, 0(a2) # store the calculated factorial to result

j exit # jump to exit

fact:

addi sp, sp, -8 # decrement stack to store two words

sw ra, 0(sp) # store value of ra to stack

sw s0, 4(sp) # store value of s0 to stack

li a1, 1 # load 1 to a1 i.e a1 = 1

beq a0, x0, done # a0(num) == 0 then jump to done label

mv s0, a0 # s0 = a0

addi a0, a0, -1 # decrement a0(num) by 1

jal fact # save the return address and jump back to fact i.e recursion

mul a1, s0, a1 # a1 = a0 \* a1

done:

lw ra, 0(sp) # load ra with the current value of sp

lw s0, 4(sp) # load ra with the value located at address sp+4

addi sp, sp, 8 # increment stack or deallocate space

jr ra # jump to the return address

exit:

addi a0, x0, 1 # a0 = 1 means to print integer when ecall executes

ecall # perform the specific operation by looking at the value of a0

addi a1, x0, '\n' # load in ascii code for new line

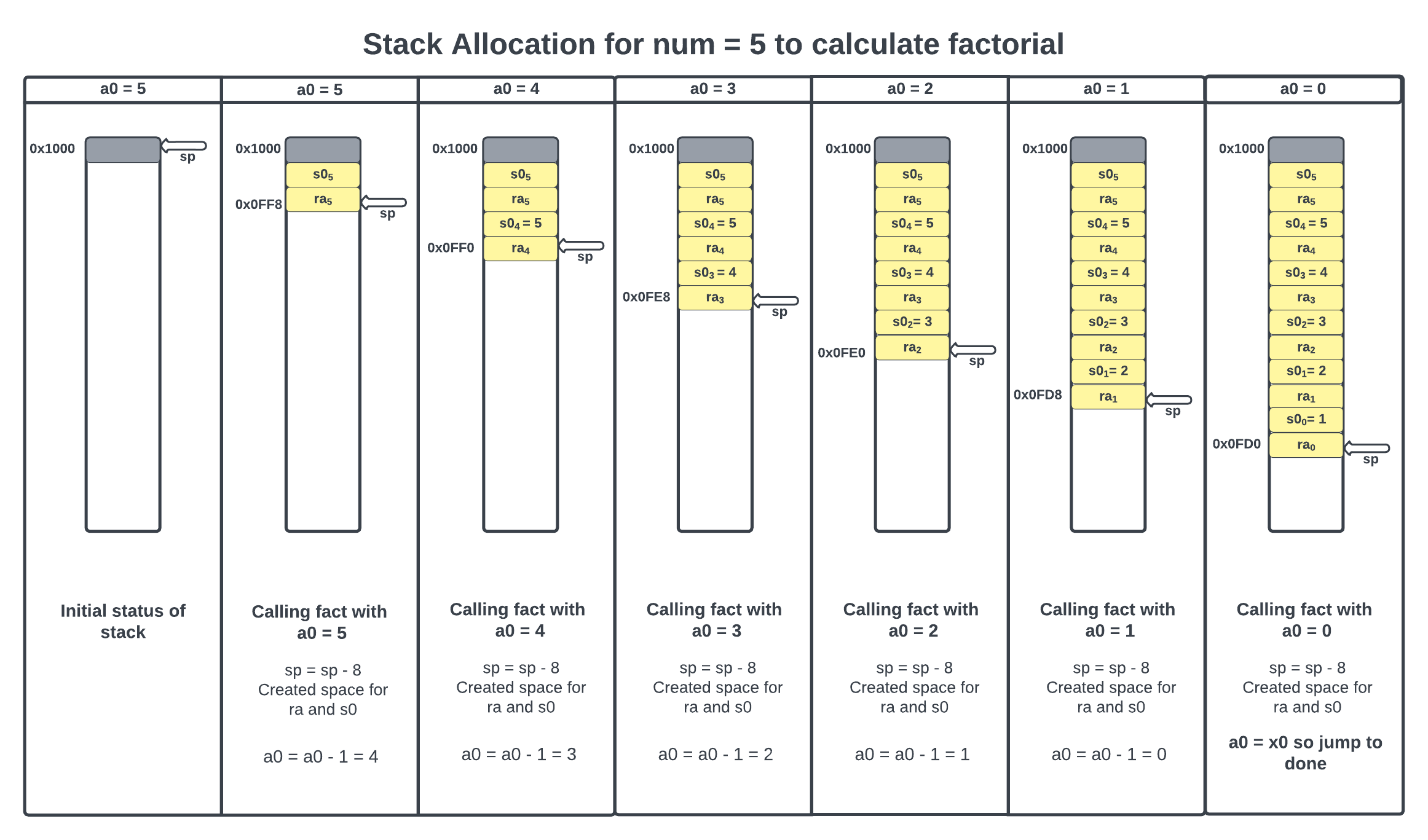
addi a0, x0, 11

ecall

addi a0, x0, 10

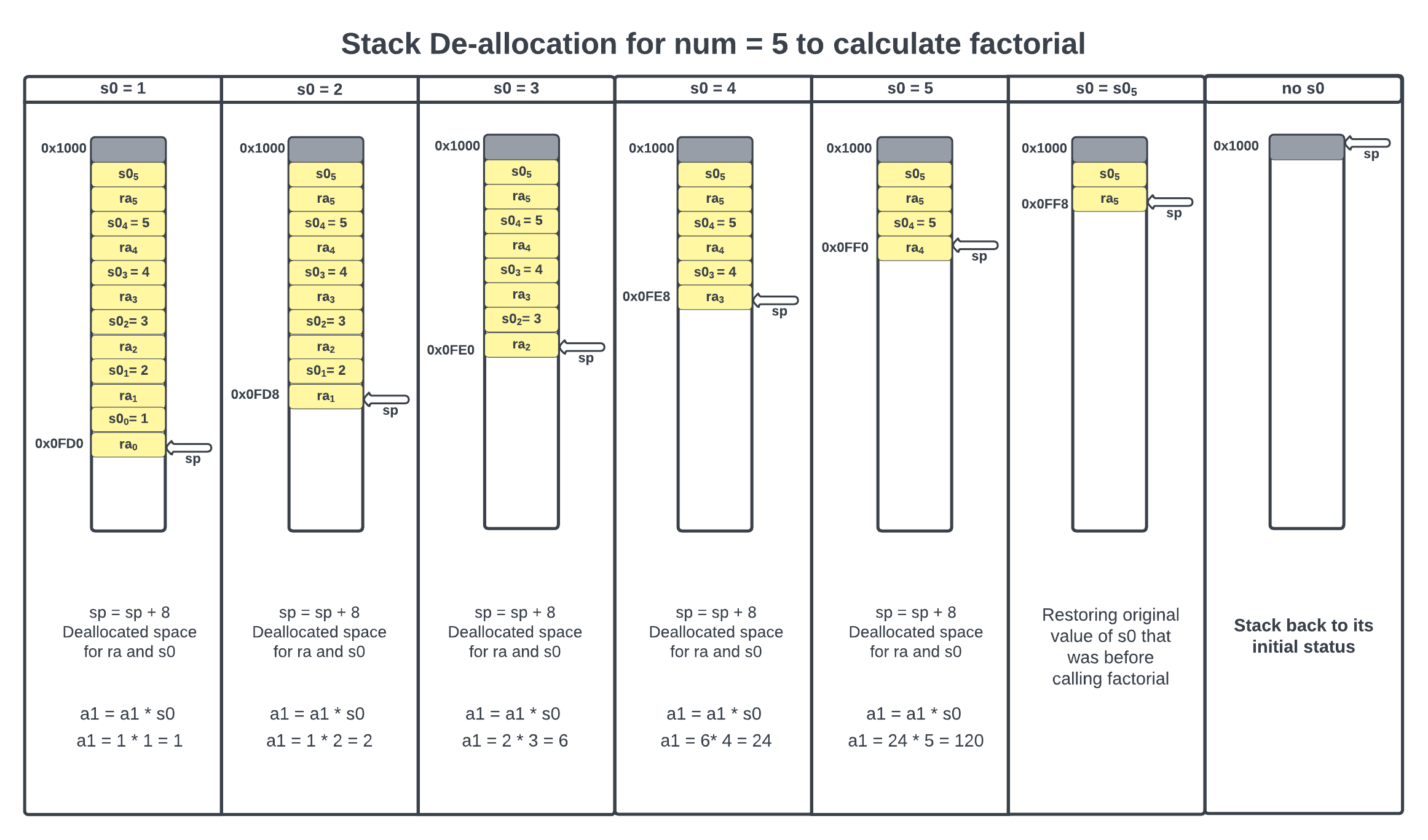
ecall # terminate program

I have created diagram by using lucid.app where status of stack for every call of factorial function is shown. You can zoom in the below pic to see that in every factorial call we are storing return address ra and s0 (which is actually a0).



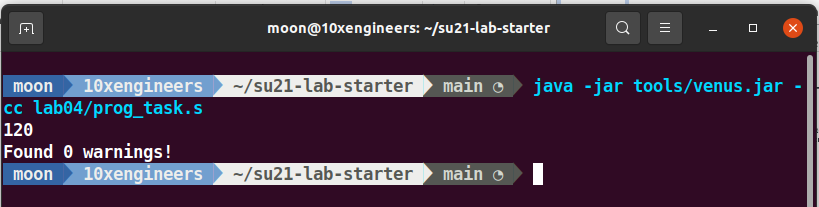
**Note:** For the sake of simplicity of understanding, I have assumed that stack is starting from 0x1000 in the figure above.

When a0 = 0, we stop pushing to stack and jump to done label. Inside done, we are loading back the values from the stack to ra and s0 registers. Then incrementing the stack or deallocating the space and after that program jump to the value of ra. Then multiplying s0 with a1 will give factorial of num as shown in the diagram below which is created by using lucid.app. (Please zoom in the diagram for better view)



**Output:**

I have run the above code using venus and also using –cc flag to see if it is following RISCV-conventions or not.



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

1. **Create your own linker script such that:**
   * + **Code section starts at 0x80000000**
     + **Data section starts at 0x80001000**

I have created linker file as shown below:

ENTRY(\_start)

SECTIONS {

. = 0x80000000;

.text : {

\*(.text)

}

. = 0x80001000;

.data : {

\*(.data)

}

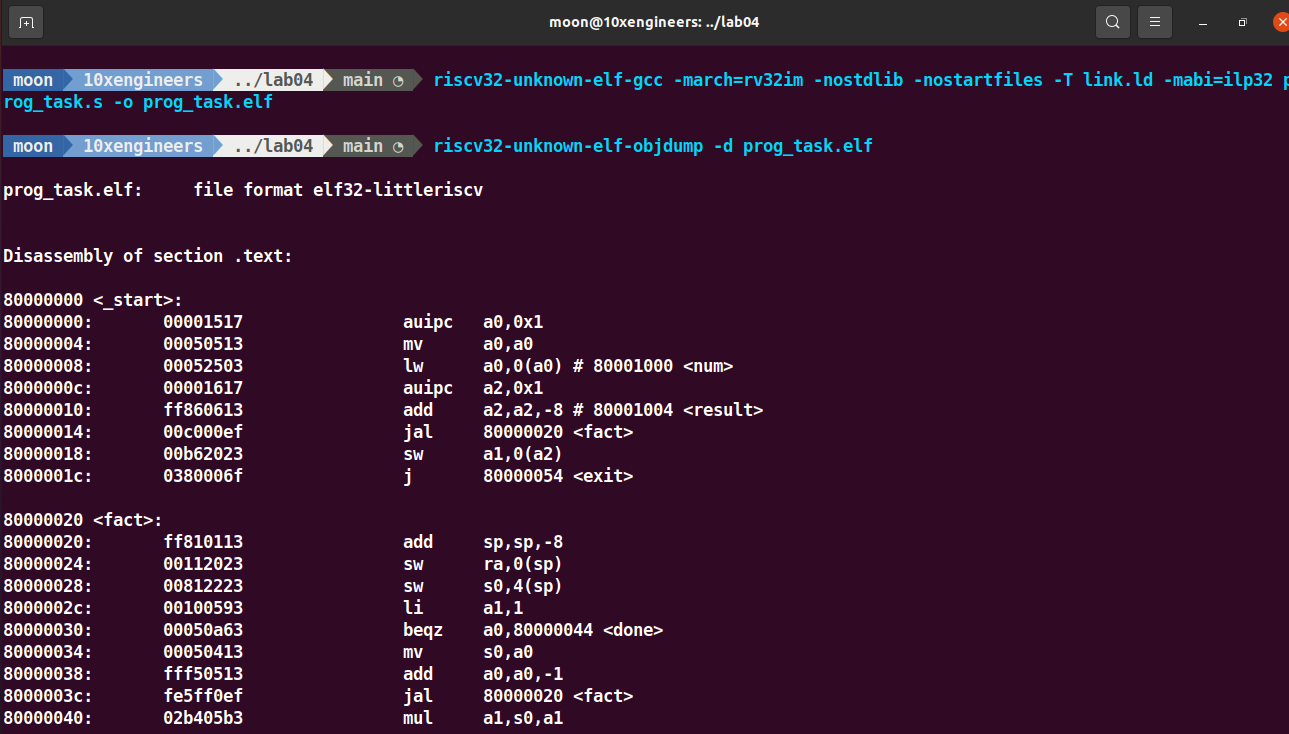
}

1. **Compile your assembly code with the custom linker script using RISC-V GNU toolchain for 32-bit architecture, without any warnings or errors.**

I have used the following command:

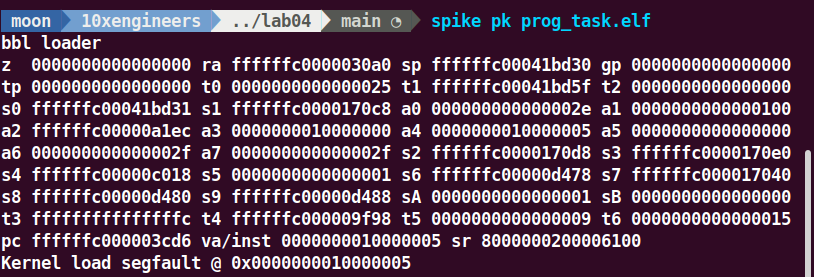
riscv32-unknown-elf-gcc -march=rv32im -nostdlib -nostartfiles -T link.ld -mabi=ilp32 prog\_task.s -o prog\_task.elf

A part of disassembly is also shown in the screenshot below:



1. **Execute (or debug) the program on SPIKE (ISA simulator).**

I have executed by using the following command:

****

**Note:** There is segmentation fault because spike can’t overwrite to the default addresses of its own.