Task Summary Report

Name: Rameen

Task Number:

Task Title: Lab 03

Lab 03 Submission: RISC-V Practice

Exercise 2: ex.s

Answers to action item:

What do `.data`, `.word`, `.text` mean?

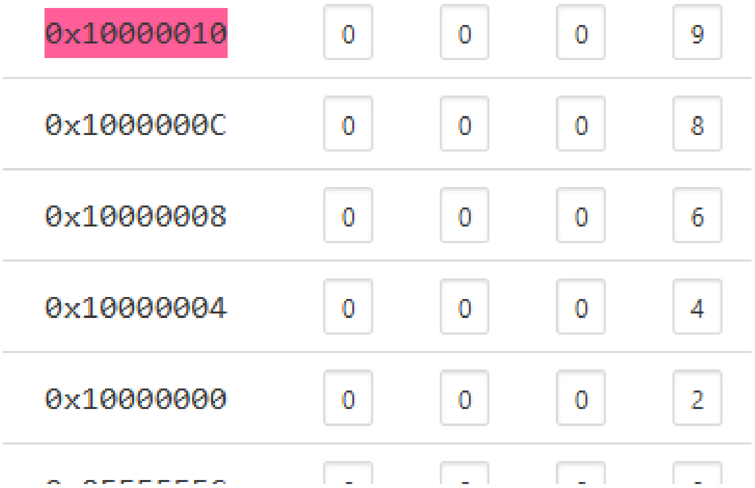
- *`.data`: Declares the data segment where variables are stored.*
- *`.word`: Reserves 4 bytes and stores a 32-bit integer in memory.*
- *`.text`: Declares the start of the code segment (instructions go here).*

What number did the program output? What does it represent?

- *The program outputs **34.***
- *This is the 9th Fibonacci number (since `n = 9` and the counting starts at 0).*

At what address is `n` stored in memory?

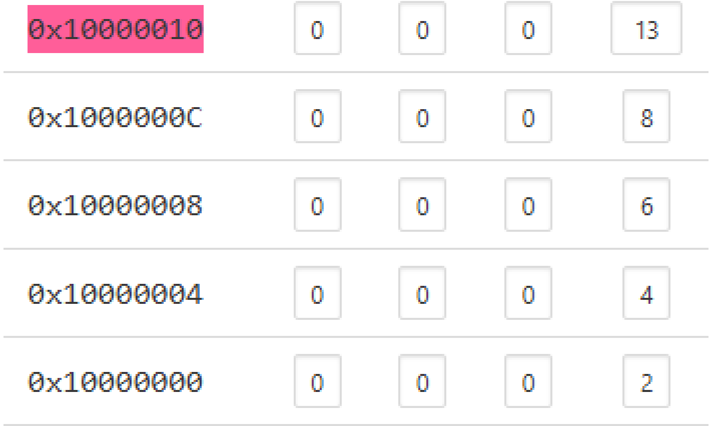
at address colored in pink:



0x10000010	0	0	0	9
0x1000000C	0	0	0	8
0x10000008	0	0	0	6
0x10000004	0	0	0	4
0x10000000	0	0	0	2

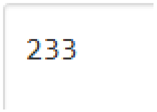
How to get the 13th Fibonacci number without editing code?

- *Before the line* `lw t3, 0(t3)` *executes, set the memory at label* `n` *to 13 using the Simulator tab:*
-
  - *Pause execution before* `lw t3, 0(t3)`
  - *In memory (where* `n:` *is), change value from* `9` *to* `13`



0x10000010	0	0	0	13
0x1000000C	0	0	0	8
0x10000008	0	0	0	6
0x10000004	0	0	0	4
0x10000000	0	0	0	2

  - *Resume the program*
- *The output will now be 233, which is the 13th Fibonacci number.*



```
233
```

Exercise 3: ex2.c,ex2.s

- The register representing the variable `k`: **`t0`**

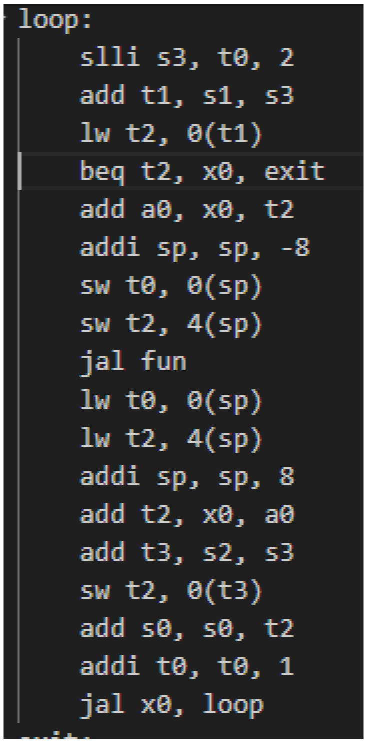
- The register representing the variable `sum`: **`s0`**

- The registers acting as pointers to the source and dest arrays:

    - **`s1` → pointer to `source`**

    - **`s2` → pointer to `dest`**

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



```
loop:
    slli s3, t0, 2
    add t1, s1, s3
    lw t2, 0(t1)
    beq t2, x0, exit
    add a0, x0, t2
    addi sp, sp, -8
    sw t0, 0(sp)
    sw t2, 4(sp)
    jal fun
    lw t0, 0(sp)
    lw t2, 4(sp)
    addi sp, sp, 8
    add t2, x0, a0
    add t3, s2, s3
    sw t2, 0(t3)
    add s0, s0, t2
    addi t0, t0, 1
    jal x0, loop
```

- How the pointers are manipulated in the assembly code:

    - **`slli s3, t0, 2`: Computes byte offset for index k**
    - **`add t1, s1, s3`: Computes &`source[k]`**
    - **`add t3, s2, s3`: Computes &`dest[k]`**
    - **These simulate `source[k]` and `dest[k]` access using base pointer + offset**

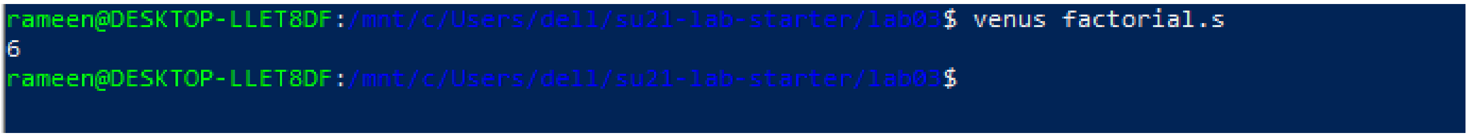
Exercise 4: Factorial

Task:

Implement the factorial function in RISC-V using either iteration or recursion.

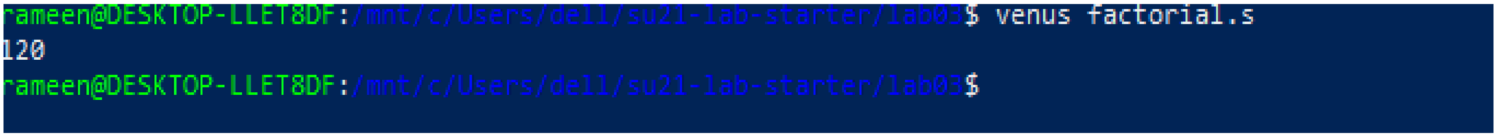
Testing Results:

1. **Input: 3 → Output: 6**



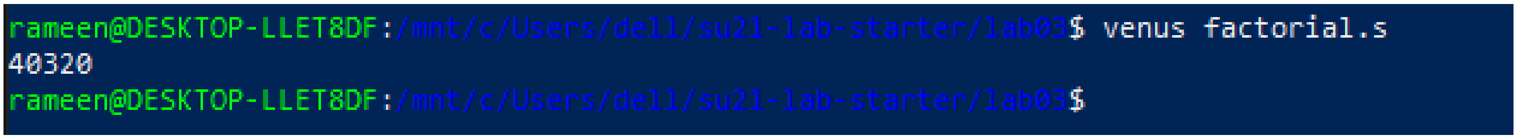
```
rameen@DESKTOP-LLET8DF:/mnt/c/Users/dell/su21-lab-starter/lab03$ venus factorial.s
6
rameen@DESKTOP-LLET8DF:/mnt/c/Users/dell/su21-lab-starter/lab03$
```

2. **Input: 5 → Output: 120**



```
rameen@DESKTOP-LLET8DF:/mnt/c/Users/dell/su21-lab-starter/lab03$ venus factorial.s
120
rameen@DESKTOP-LLET8DF:/mnt/c/Users/dell/su21-lab-starter/lab03$
```

3. **Input: 8 → Output: 40320**



```
rameen@DESKTOP-LLET8DF:/mnt/c/Users/dell/su21-lab-starter/lab03$ venus factorial.s
40320
rameen@DESKTOP-LLET8DF:/mnt/c/Users/dell/su21-lab-starter/lab03$
```

Code link (factorial.s):

https://github.com/rmknae/Meds_repo/blob/main/Remedial/R2/Task%3A%20RISC-V%20Instruction%20Formats/exercise4/factorial.s

Exercise 5: Linked List Map

Task:

Complete the `map` function in RISC-V to apply a function to each element of a linked list. Implement function pointer handling using `jalr`.

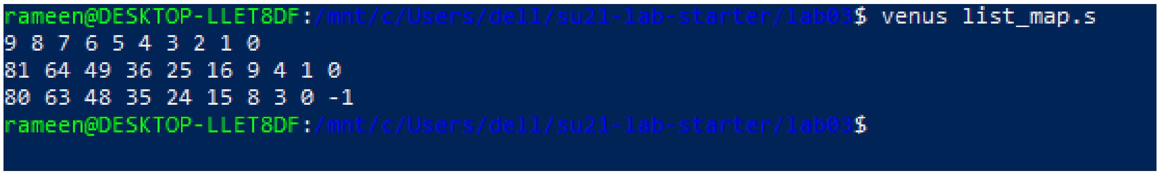
- **Expected Output:**

  9 8 7 6 5 4 3 2 1 0

  81 64 49 36 25 16 9 4 1 0

  80 63 48 35 24 15 8 3 0 -1

- **Output on console:**



```
rameen@DESKTOP-LLET8DF:/mnt/c/Users/dell/su21-lab-starter/lab03$ venus list_map.s
9 8 7 6 5 4 3 2 1 0
81 64 49 36 25 16 9 4 1 0
80 63 48 35 24 15 8 3 0 -1
rameen@DESKTOP-LLET8DF:/mnt/c/Users/dell/su21-lab-starter/lab03$
```

Map Function Code (link):

https://github.com/rmknae/Meds_repo/blob/main/Remedial/R2/Task%3A%20RISC-V%20Instruction%20Formats/exercise5/list_map.s

Answers to Questions in code::

- Why use `a0` to load the value of the current node?

  Because `a0` is the standard register for the first argument to a function. We're preparing to call the function pointer with the node's value.

- Why not use a label when calling the function?

  We don't use a specific label because we want `map` to work with any function. By using a function pointer in a register, `map` becomes reusable. This way, we can use it for both `square` and `decrement` without changing the code.

- Where is the returned value from the function?

  It's returned in `a0` ,the standard return value register in RISC-V.

- Why store the function address back into `a1` before the recursive call?

  Because `map` expects the function pointer as its second argument in `a1`. Register `a1` may have been overwritten during the function call, so it must be restored.

- What about `a0`?

  It holds the pointer to the next node, which becomes the first argument for the recursive call to `map`.