LAB - 4 (RISC-V SIMULATOR)

INTRODUCTION

This project is a RISC V simulator supporting functionalities like parsing and executing various instruction formats(R,I,S,J,B,U), mapping labels, managing registers and memory operations.

CODE DESIGN

We start this code by obtaining the input file from the user. The user has the option to execute various operations such as running or stepping through it. Then we define key constants like number of registers and arrays to represent both registers (including their aliases) and memory making it straightforward to simulate data loading and storing operations.

Since labels are used in branch and jump instructions, we wrote a maplabels function that reads the input file line-by-line, detects labels, and associates them with memory addresses. The first scans are done for .data and .text so that we can store the initialized memory from .data section.All the instructions and arrays are reset every time we exit the file.

The simulator parses the instruction using a combination of string manipulation functions and breaks down the line into its instruction, respective registers and immediates/ offsets and then perform the instruction. For store instructions we store the value in the memory address according to its offset. Also, at the end of every instruction we increase the PC by 4 as each instruction is of 4 bytes.

For run command int the program, we show all the instructions and their corresponding program counter (PC). If the user selects step, they can execute one instruction at a time. The break command allows them to set breakpoints at specific instructions to aid in debugging. Additionally, we have the capability to print the memory at each 1-byte address.

Overall, the coding approach focused on creating a modular and well-documented RISC V simulator providing useful debugging tools.

CHALLENGES

- We could not get a proper implementation idea for show stack and our code did not work.

- Managing the memory allocation for the .data section proved to be difficult as we could not store multiple values from that section in the same line.

- We initially made the breakpoint oriented around PC but it did not work properly when .data and .text were involved.

- The project took a lot of time since there were many cases to check and handle.

IMPROVEMENTS

- The code is too lengthy and most of the code is too repetitive for cases like errors. Usage of multiple files could have made a much better impact on this project. The readability of the code is not so perfect and might confuse the reader which could have been handled better.

- Adding a GUI for easier debugging and execution control would make the simulator more user-friendly.

TESTING

Input.s:

```
main: addi x10, x0, 2
      lui sp, 0x50
      jal x1, fact
      beq x0, x0, exit
fact: addi sp, sp, -16
          sd x1, 8(sp)
          sd x10, 0(sp)
          addi x5, x10, -1
          blt x0, x5, L1
          addi x10, x0, 1
          addi sp, sp, 16
          jalr x0, x1(0)
L1:   addi x10, x10, -1
          jal x1, fact
          addi x6, x10, 0
          ld x10, 0(sp)
          ld x1, 8(sp)
          addi sp, sp, 16
       addi x20, x0, 0
       addi x8, x0, 0
mul: add x8, x8, x6
     addi x20, x20, 1
     bne x20, x10, mul
     add x10, x8, x0
     jalr x0, x1(0)
exit: add x0, x0, x0
```

Command Line Interface:

```
main: addi x10, x0, 2
      lui sp, 0x50
      jal x1, fact
      beq x0, x0, exit
fact: addi sp, sp, -16
          sd x1, 8(sp)
          sd x10, 0(sp)
          addi x5, x10, -1
          blt x0, x5, L1
          addi x10, x0, 1
          addi sp, sp, 16
          jalr x0, x1(0)
L1:   addi x10, x10, -1
          jal x1, fact
          addi x6, x10, 0
          ld x10, 0(sp)
          ld x1, 8(sp)
          addi sp, sp, 16
       addi x20, x0, 0
       addi x8, x0, 0
mul: add x8, x8, x6
    addi x20, x20, 1
    bne x20, x10, mul
    add x10, x8, x0
    jalr x0, x1(0)
exit: add x0, x0, x0
```

CONCLUSION

The provided C++ code is a functional and detailed RISC-V simulator that supports many functionalities for an assembly code. The development of the RISC-V simulator presented a range of challenges and learning opportunities. Managing .data section proved more difficult than expected, especially when ensuring correct memory alignment. But ultimately, the project served as a strong foundation to understand RISC-V simulation. . Although there is room for improvement, the simulator works efficiently for its main task.