RISC-V Simulator: Implementation Report

Student Name	Roll No.
Dheeraj Agarwal	CS23BTECH11016
Nenavath Kalyan	CS23BTECH11039

1 Introduction

This report describes the implementation of a RISC-V simulator capable of executing a subset of RISC-V instructions, tracking the execution flow with a call stack, and providing various functionalities such as stepping through instructions, viewing register contents, examining memory, and managing breakpoints.

The simulator supports:

- Execution of RISC-V instructions like arithmetic, load/store, branches, and jumps.
- Memory operations and register updates.
- Call stack tracking for function invocations and returns.
- Stepping through instructions, setting and deleting breakpoints, and inspecting the call stack, registers, and memory.

2 Instruction Support

The simulator implements several RISC-V instruction formats:

- **R-format**: Arithmetic operations .
- **I-format**: Load, immediate arithmetic, and jump instructions
- **S-format**: Store instructions
- **B-format**: Conditional branches
- **J-format**: Jump instructions,
- **U-format**: Upper immediate instructions

3 Key Features and Design

3.1 Execution Flow

The instruction execution starts with the `step()` function, which fetches the current instruction at the Program Counter (PC), decodes it, and executes the corresponding operation. Each instruction increments the PC by 4 after execution, reflecting the 4-byte alignment of RISC-V instructions.

3.2 Register and Memory Management

- Registers are simulated using an array `registers[]`, mimicking RISC-V's 32 general-purpose registers.
- Memory is simulated using an array `memory[]`. Store and load instructions access this array, with bounds checking to prevent invalid accesses.

3.3 Call Stack Management

The call stack tracks function calls and returns:

- `jal` pushes the return address onto the stack, then jumps to the target.
- `jalr` pops the stack to return to the calling function.

The `show-stack` command displays the last executed line for each function in the call stack, ensuring that the stack reflects the correct state during execution.

3.4 Breakpoints

Breakpoints pause execution at specific lines, enabling the user to inspect the program's state. The simulator uses the `break` command to set breakpoints and the `del` command to remove them.

4 Commands Implemented

The following commands allow users to interact with the simulator:

- `load [filename]`: Load instructions from a file.
- `run`: Run the program until a breakpoint or the end.
- `step`: Execute a single instruction.
- `regs`: Display all register contents.
- `mem [addr] [count]`: Display memory contents starting at a specific address.
- `show-stack`: Display the current call stack.
- `break [line]`: Set a breakpoint at a specified line.
- `del break [line]`: Delete a breakpoint at a specified line.
- `exit`: Exit the simulator.

5 Call Stack and Stepping Example

The call stack is updated during execution, reflecting the current state of function invocations:

- At the beginning of execution, the call stack shows `main: 0`, indicating no instructions have been executed.
- When the main function calls `fact()` at line 3, the call stack shows `main: 3`.
- After stepping into the `fact()` function, the last executed instruction in `fact` is shown (e.g., `fact: 13`).

6 Conclusion

The RISC-V simulator provides a simple yet effective platform for simulating RISC-V assembly programs. It supports a range of RISC-V instructions, manages a call stack, and offers commands for stepping through code, inspecting registers and memory, and managing breakpoints. This allows users to trace and debug RISC-V programs efficiently.

Input.s

```
.data
.dword 3, 3, 4, 4, 8, 3, 6
.text
lui x3, 0x10
;load the value of count_of_gcd in x4 register
ld x4,0(x3)
addi x5,x3,0x2    ;store the starting address of gcd in x5
addi x10,x3,8   ; for taking input
Loop:   ld x6,0(x10)
   ld x7,8(x10)
   ; if any number is zero
   beq x6,x0,gcd_is_zero
   beq x7,x0,gcd_is_zero
   beq x0,x0,Calculate_GCD   ; to jump from gcd_is_zero
gcd_is_zero: sd x0,0(x5)          ;store 0 in x5
   beq x0,x0, Next_Group    ; unconditional loop to go to Next_Group
Calculate_GCD: beq x6,x7,GCD    ; if both number are equal then return gcd
   blt x6,x7,swap   ; if x6<x7 then swap the number
   sub x6,x6,x7    ; x6 = x6-x7
   beq x0,x0,Calculate_GCD
swap: addi x8,x6,0
   addi x6,x7,0
   addi x7,x8,0
   beq x0, x0, Calculate_GCD
GCD: sd x6,0(x5)
Next_Group: addi x5,x5,8
```

```
    addi x10,x10,16
    addi x4,x4,-1
    blt x0,x4,Loop
Exit: add x0,x0, x0
```

Command Line Interface:

Since the file begins with the .data, followed by .dword 3,4.... and .text, so the number of lines that have been executed are 3. After that the opcode lui is seen. So, initially the main function has a value of 3, as you can see.

```
> load input.s

> show-stack
Call Stack:
main:3

> step
Executed: lui x3, 0x10 ;   PC=0x00000000

> break 12
Breakpoint set at line 12

> run
Executed: ld x4,0(x3) ;   PC=0x00000004
Executed: addi x5,x3,0x2 ;   PC=0x00000008
Executed: addi x10,x3,8 ;   PC=0x0000000c
```

Executed: ld x6,0(x10) ; PC=0x00000010

Executed: ld x7,8(x10) ; PC=0x00000014

Executed: beq x6,x0,gcd_is_zero ; PC=0x00000018

Executed: beq x7,x0,gcd_is_zero ; PC=0x0000001c

Executed: beq x0,x0,Calculate_GCD ; PC=0x00000020

Execution stopped at breakpoint

> regs

x0: 0x0

x1: 0x0

x2: 0x0

x3: 0x10000

x4: 0x3

x5: 0x10002

x6: 0x3

x7: 0x4

x8: 0x0

x9: 0x0

x10: 0x10008

x11: 0x0

x12: 0x0

x13: 0x0

x14: 0x0

x15: 0x0

x16: 0x0

x17: 0x0

x18: 0x0

x19: 0x0

x20: 0x0

x21: 0x0

x22: 0x0

x23: 0x0

x24: 0x0

x25: 0x0

x26: 0x0

x27: 0x0

x28: 0x0

x29: 0x0

x30: 0x0

x31: 0x0

> mem 0x10000 9

Memory[0x10000] = 0x03

Memory[0x10001] = 0x00

Memory[0x10002] = 0x00

Memory[0x10003] = 0x00

Memory[0x10004] = 0x00

Memory[0x10005] = 0x00

Memory[0x10006] = 0x00

Memory[0x10007] = 0x00

Memory[0x10008] = 0x03

> step

Executed: beq x6,x7,GCD ; PC=0x0000002c

> run

Executed: blt x6,x7,swap ; PC=0x00000030

Executed: addi x8,x6,0 ; PC=0x0000003c

Executed: addi x6,x7,0 ; PC=0x00000040

Executed: addi x7,x8,0 ; PC=0x00000044

Executed: beq x0, x0, Calculate_GCD ; PC=0x00000048

Execution stopped at breakpoint

> show-stack

Call Stack:

main:22

> step

Executed: beq x6,x7,GCD ; PC=0x0000002c

> run

Executed: blt x6,x7,swap ; PC=0x00000030

Executed: sub x6,x6,x7 ; PC=0x00000034

Executed: beq x0,x0,Calculate_GCD ; PC=0x00000038

Execution stopped at breakpoint

> step

Executed: beq x6,x7,GCD ; PC=0x0000002c

> run

Executed: blt x6,x7,swap ; PC=0x00000030

Executed: addi x8,x6,0 ; PC=0x0000003c

Executed: addi x6,x7,0 ; PC=0x00000040

Executed: addi x7,x8,0 ; PC=0x00000044

Executed: beq x0, x0, Calculate_GCD ; PC=0x00000048

Execution stopped at breakpoint