REPORT

RISC-V SIMULATOR

Youssef nader - 900226170 - youssefnader@aucegypt.edu

Youssef attalah – 900221247 – youssefemad25@aucegypt.edu

Youssef Ibrahim – 900212889 – youssefibrahim@aucegypt.edu

Course: Computer Organization and Assembly Language

Under the supervision of Prof. Cherif Salama

Description:

The implementation is a **RISC-V instruction set simulator** capable of executing a subset of RISC-V assembly instructions. It processes an input file containing assembly instructions, simulates their execution, and outputs detailed logs of register and memory states after each instruction. It also outputs the final states of all the registers and memory at the end of the assembly program.

Decision and assumptions:

- Focused on the main instructions of the RISC-V simulator (no pseudo instructions)
- Focused on integer based
- The registers should be starting with an r instead of an x

Bugs and Issues:

- The program has no issues or bugs at the time we are submitting this file

User Guide:

- You will run the program and then the terminal will open asking for the input file path for a text file where the risc-v instructions are saved.

  For example we will try to run these instructions :

```
ADDI r1, r0, 5
ADDI r2, r0, 10
ADD r3, r1, r2
SUB r4, r2, r1
AND r5, r3, r4
ORI r6, r5, 2
XORI r7, r6, 3
SLL r8, r7, r1
SRL r9, r8, r2
SRA r10, r8, r1
```

  After running we will receive this from terminal:

  We will then enter the file path of our text file that has these risc-v instructions :

```
please input the input file path: C:\Users\youss\OneDrive - aucegypt.edu\Desktop\in.txt
enter initial program count:
```

  We will then be asked to enter the initial program count, which is the line we want our program to start from and since want to run the whole program we will enter this to be 0

```
please input the input file path: C:\Users\youss\OneDrive - aucegypt.edu\Desktop\in.txt
enter initial program count: 0
ADDI: R1 = R0 + 5 (0 + 5 = 5)
Updated Program Count: 1
ADDI: R2 = R0 + 10 (0 + 10 = 10)
Updated Program Count: 2
ADD: R3 = R1 + R2 (5 + 10 = 15)
Updated Program Count: 3
SUB: R4 = R2 - R1 (10 - 5 = 5)
Updated Program Count: 4
AND: R5 = R3 & R4 (15 & 5 = 5)
Updated Program Count: 5
ORI: R6 = R5 | 2 (5 | 2 = 7)
Updated Program Count: 6
XORI: R7 = R6 ^ 3 (7 ^ 3 = 4)
Updated Program Count: 7
SLL: R8 stores R7 shifted left logically by R1 (4 << 5 = 128)
Updated Program Count: 8
SRL: R9 stores R8 shifted right logically by R2 (128 << 10 = 0)
Updated Program Count: 9
SRA: R10 stores R8 shifted right arithmetically by R1 (128 << 5 = 4)
Updated Program Count: 10
 please input the output file path:
```

It will then output the output of each instruction in the terminal and ask for the path of the output file were we want to print the final result

```
 please input the output file path: C:\Users\youss\OneDrive - aucegypt.edu\Desktop\in.txt

Final State of Registers:
R0: 0
R1: 5
R2: 10
R3: 15
R4: 5
R5: 5
R6: 7
R7: 4
R8: 128
R9: 0
R10: 4
R11: 0
R12: 0
R13: 0
R14: 0
R15: 0
R16: 0
R17: 0
R18: 0
R19: 0
R20: 0
R21: 0
R22: 0
R23: 0
R24: 0
R25: 0
R26: 0
R27: 0
R28: 0
R29: 0
R30: 0
R31: 0
```

```
Final State of Memory:
Address: 0 -> Value: 253
Address: 1 -> Value: 216
Address: 2 -> Value: 164
Address: 3 -> Value: 28
Address: 4 -> Value: 82
Address: 5 -> Value: 69
Address: 6 -> Value: 91
Address: 7 -> Value: 60
Address: 8 -> Value: 94
Address: 9 -> Value: 22
```

It will then output all the values of the registers and memory, and it will print these outputs in the output file.

The following is the output file :

```
ADDI: R1 = R0 + 5 (0 + 5 = 5)
Updated Program Count: 1

ADDI: R2 = R0 + 10 (0 + 10 = 10)
Updated Program Count: 2

ADD: R3 = R1 + R2 (5 + 10 = 15)
Updated Program Count: 3

SUB: R4 = R2 - R1 (10 - 5 = 5)
Updated Program Count: 4

AND: R5 = R3 & R4 (15 & 5 = 5)
Updated Program Count: 5

ORI: R6 = R5 | 2 (5 | 2 = 7)
Updated Program Count: 6

XORI: R7 = R6 ^ 3 (7 ^ 3 = 4)
Updated Program Count: 7

SLL: R8 stores R7 shifted left logically by R1 (4 << 5 = 128)
Updated Program Count: 8

SRL: R9 stores R8 shifted right logically by R2 (128 << 10 = 0)
Updated Program Count: 9

SRA: R10 stores R8 shifted right arithmetically by R1 (128 << 5 = 4)
Updated Program Count: 10
```

```
Final State of Registers:
R0: 0
R1: 5
R2: 10
R3: 15
R4: 5
R5: 5
R6: 7
R7: 4
R8: 128
R9: 0
R10: 4
R11: 0
R12: 0
R13: 0
R14: 0
R15: 0
R16: 0
R17: 0
R18: 0
R19: 0
R20: 0
R21: 0
R22: 0
R23: 0
R24: 0
R25: 0
R26: 0
R27: 0
R28: 0
R29: 0
R30: 0
R31: 0
```

```
Final State of Memory:
Address: 0 -> Value: 253
Address: 1 -> Value: 216
Address: 2 -> Value: 164
Address: 3 -> Value: 28
Address: 4 -> Value: 82
Address: 5 -> Value: 69
Address: 6 -> Value: 91
Address: 7 -> Value: 60
Address: 8 -> Value: 94
Address: 9 -> Value: 22
```

Programs tested:

Program 1 :

```
ADDI r1, r0, 5
ADDI r2, r0, 10
ADD r3, r1, r2
SUB r4, r2, r1
AND r5, r3, r4
ORI r6, r5, 2
XORI r7, r6, 3
SLL r8, r7, r1
SRL r9, r8, r2
SRA r10, r8, r1
```

Program 2 :

```
ADDI r1, r0, 3
ADDI r2, r0, 5
SW r2, r1, 0
LW r3, r1, 0
BEQ r3, r2, SKIP
ADDI r4, r0, 10
SKIP: ORI r5, r0, 20
```

Program 3 :

```
ADDI r1, r0, 1
ADDI r2, r0, 5
ADDI r3, r0, 0
LOOP: ADD r3, r3, r1
ADDI r1, r1, 1
BNE r1, r2, LOOP
```

Program 4:

```
ADDI r1, r0, 5
ADDI r2, r0, 10
SLT r3, r1, r2
SLTU r4, r1, r0
SLTIU r5, r1, 7
LUI r6, 100
AUIPC r7, 4
SLLI r8, r1, 2
SRLI r9, r8, 1
SRAI r10, r8, 1
SB r1, r2, 0
LB r11, r2, 0
LH r12, r2, 0
LBU r13, r2, 0
LHU r14, r2, 0
JAL r15, 4
JALR r16, r2, 8
```

Conclusion

The development of this RISC-V simulator has been an insightful journey, offering us a deeper understanding of computer architecture and low-level programming. Through implementing various RISC-V instructions and debugging complex interactions between registers, memory, and control flow, we gained a practical appreciation for how modern processors execute instructions at the hardware level.