Project Report: RISC-V Simulator Implementation

Team Members:

1. [Mohamed Abdelmagid], ID: 900223215
2. [Tarek Kassab], ID: 900213491
3. [Shaza Ali], ID: 900213391

1. Brief Description of Implementation

Our RISC-V simulator is designed to simulate the behavior of a RISC-V instructions, supporting a wide range of instructions. The simulator includes the basic set of instructions, arithmetic and logical operations, memory access instructions, conditional branches, and jumps. We implemented bonus features such as support for the multiplication and dicision instructions (First Bonus). We added test cases for real programs (Second Bonus). We also support providing data in binary, hexadecimal, and deciaml as well (Third Bonus).

2. Design Decisions and Assumptions

1. **Modularity:**
    - Decision: Separate the simulator into distinct modules or classes for better organization.
    - Reasoning: Modular design allows for easier maintenance, testing, and extension of the simulator.
2. **Instruction Execution Model:**
    - Decision: Implement an execution model that follows the RISC-V instruction set architecture. It is a single function called `execute` to call the other funcion that is related to the instruction.
    - Reasoning: Adhering to the RISC-V ISA ensures compatibility with RISC-V programs and facilitates easier validation against the official specification.
3. **Memory Representation:**
    - Decision: Use a memory data structure for instructions as a vector of strings and memory as a map to simulate load and store operations.
4. **Error Handling:**
    - Decision: Implement error-checking mechanisms for invalid instructions, memory access violations, or other exceptional conditions.
    - Reasoning: Robust error handling improves the simulator's reliability and aids in debugging.
5. **Bonus Features**
    - We Chose to implement RV32IM and output the results in decimal, binary, and hexadecimal as our bonuses.

Assumptions:

1. **Correct Input Programs:**
    - Assumption: Input programs are correctly written RISC-V assembly code.
    - Reasoning: The simulator focuses on the execution of valid RISC-V instructions. Handling invalid or malformed input just by telling the user that there is an error in a certain line.
2. **Little-Endian Architecture:**
    - Assumption: The simulator assumes a little-endian memory architecture unless specified otherwise.
    - Reasoning: Little-endian is a common architecture, and assuming it simplifies memory access and byte-ordering operations.
3. **Single-Threaded Execution:**
    - Assumption: The simulator assumes single-threaded execution and does not handle multi-threading or parallelism.
    - Reasoning: Simplifying the simulator to a single-threaded model reduces complexity and is suitable for many educational or introductory purposes.

3. Known Bugs or Issues

System Integration Bugs

- Bug: Issues related to integrating the simulator into a larger system or debugging environment.
- Solution: To address system integration bugs, it is crucial to ensure proper integration by adhering to any provided guidelines or specifications. Thoroughly test the simulator in the intended environment, identifying and resolving any compatibility issues that may arise during integration. This process may involve collaboration with the larger system development team to ensure seamless compatibility and interoperability.

Performance Bugs

- Bug: Suboptimal performance or excessive resource usage.
- Solution: To tackle performance bugs, it is essential to optimize critical sections of the code and implement efficient algorithms. Profiling the simulator can help identify and prioritize performance bottlenecks. By employing optimization techniques, such as algorithmic improvements or code restructuring, the simulator can achieve better overall performance. Regular profiling and performance testing should be conducted to maintain optimal system efficiency.

Instruction Execution Bugs

- Bug: Incorrect implementation of one or more RISC-V instructions.
- Solution: To mitigate instruction execution bugs, a thorough review of the execution logic for each RISC-V instruction is necessary. This review should consider both the RISC-V specification and potential edge cases that may affect correct execution. Implementing a comprehensive set of test cases, covering various scenarios and corner cases, is essential to validate the correctness of instruction execution. Continuous testing and validation efforts should be

employed to catch and rectify any discrepancies in the implementation.

4. User Guide: Compile and Run

Compilation:

1. Clone the repository from [GitHub URL].
2. Navigate to the project directory.
3. Add your instructions.txt and data.txt files in the main directory (or rename the test instructions to instruction.txt)
4. Run the file main.cpp ```

5. Simulated Programs

To initiate one of these tests, please rename its text file to "instructions" only, deleting the testx part in the name. By default, there is an empty instruction.txt available if you want to add your own assembly code in it. While these programs don't use preloaded data, the functionality is available and to showcase that there are some generic data values in the data.txt document.

1. **Program 1: Basic Arithmetic and Logic**
    - A program that covers basic arithmetic (addition, subtraction), logical (AND, OR, XOR), and immediate instructions. Program 1 logic C++:

```
int ComputeLogic(int logic,  int n, unsigned int m)
{
   int result;
      if( logic == 0) //xori
         result= n^1;
      else if(logic == 1 )// "sll"
         result= n<<m;
      else if (logic == 2 )// "slli"
         result= n<<1;
      else if (logic == 3) //"srl"
         result= n/m;
      else if (logic == 4) //"srli"
         result= n/2;
      else if (logic == 5) //"sra"
         result= n>>m;
      else if (logic == 6 ) //"srai"
         result= n>>1;
      else if(logic == 7) //"and"
         result= n&m;
      else if (logic == 8) //"andi"
         result= n&1;
      else if (logic == 9) //"or"
         result= n|m;
      else if (logic == 10 ) //"ori"
         result= n|1;
      else if (logic ==11 ) //"xor"
         result= n^m;
   return result;


}


int main()
{
   int n=1;
   int m=0;
ComputeLogic(0, n,m);
ComputeLogic(10,n, m);
ComputeLogic(9,n, m);
ComputeLogic(8, n, m);
ComputeLogic(7,  n, m);
ComputeLogic(6,  n,  m);
ComputeLogic(5, n, m);
ComputeLogic(4,  n, m);
ComputeLogic(3, n, m);
ComputeLogic(2,  n, m);
ComputeLogic(1, n, m);
ComputeLogic(11, n, m);
return 0;
}
```

Assembly:

```
main:
addi a1 zero 1
```

```
addi a1,zero,1
addi a2,zero,0
addi a0,zero,0
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,10
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,9
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,8
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,7
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,6
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,5
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,4
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,3
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,2
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,1
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
addi a0,zero,11
jal ra,ComputeLogic
addi sp,sp,-4
sw a0,0(sp)
ebreak


ComputeLogic:
beq a0,zero,f0
addi t0,zero,1
beq a0,t0,f1
addi t0,t0,1
beq a0,t0,f2
addi t0,t0,1
beq a0,t0,f3
addi t0,t0,1
beq a0,t0,f4
addi t0,t0,1
beq a0,t0,f5
addi t0,t0,1
beq a0,t0,f6
addi t0,t0,1
beq a0,t0,f7
addi t0,t0,1
beq a0,t0,f8
addi t0,t0,1
beq a0,t0,f9
addi t0,t0,1
beq a0,t0,f10
addi t0,t0,1
beq a0,t0,f11
```

```
f0:
xori a0, a1, 1
jalr zero,0(ra)


f1:
sll a0, a1, a2
jalr zero,0(ra)


f2:
slli a0, a1, 1
jalr zero,0(ra)


f3:
srl a0, a1, a2
jalr zero,0(ra)


f4:
srli a0, a1, 1
jalr zero,0(ra)


f5:
sra a0, a1, a2
jalr zero,0(ra)


f6:
srai a0, a1, 1
jalr zero,0(ra)


f7:
and a0, a1, a2
jalr zero,0(ra)


f8:
andi a0, a1, 1
jalr zero,0(ra)


f9:
or a0, a1, a2
jalr zero,0(ra)


f10:
ori a0, a1, 1
jalr zero,0(ra)


f11:
xor a0, a1, a2
jalr zero,0(ra)
```

2. **Program 2: Memory Access and Branching**
    - This program involves memory access (load and store instructions) and conditional branching. C++:

```
int ComputeComp(int comp,  int n,int m, unsigned int u1,unsigned int u2)
{
   int result;
      if( comp == 0) //beq
         if(n==m)result= 1;
      else if(comp == 1 )// "bne"
         if(n!=m) result= 1;
      else if(comp == 2 )// "blt"
         if(n>m) result= 1;
      else if(comp == 3 )// "bltu"
         if(u2<u1) result= 1;
      else if(comp == 4 )// "bgeu"
         if(u2>=u1) result= 1;


      else if(comp == 5 )// "bge"
         if(m>=n) result= 1;


      else if(comp == 6)// "slt"
         result= n<m;


      else if(comp == 7 )// "slti"
         result= n<-1;


      else if(comp == 8)// "sltu"
         result= n<u1;


      else if(comp == 9 )// "sltui"
         result= n<m;







   return result;


}


int main()
{
   int n=1;
   int m=0;
   int u1=0;
   int u2=1;
ComputeLogic(0, n,m,u1,u2);
ComputeLogic(9,n, m,u1,u2);
ComputeLogic(8, n, m,u1,u2);
ComputeLogic(7,  n, m,u1,u2);
ComputeLogic(6,  n,  m,u1,u2);
ComputeLogic(5, n, m,u1,u2);
ComputeLogic(4,  n, m,u1,u2);
ComputeLogic(3, n, m,u1,u2);
ComputeLogic(2,  n, m,u1,u2);
ComputeLogic(1, n, m,u1,u2);
exit(0);
}
```

Assembly:

```
ComputeLogic:
      addi    sp,sp,-64
      sw      s0,60(sp)
      addi    s0,sp,64
      sw      a0,-36(s0)
      sw      a1,-40(s0)
      sw      a2,-44(s0)
      sw      a3,-48(s0)
      sw      a4,-52(s0)
```

```
        lw      a5,-36(s0)
        bne     a5,zero,.L2
        lw      a4,-40(s0)
        lw      a5,-44(s0)
        bne     a4,a5,.L3
        addi    a5,zero,1
        sw      a5,-20(s0)
        jal        ra,.L2
.L3:
        lw      a4,-36(s0)
        addi      a5,zero,1
        bne     a4,a5,.L2
        lw      a4,-40(s0)
        lw      a5,-44(s0)
        beq     a4,a5,.L4
        addi      a5,zero,1
        sw      a5,-20(s0)
        jal       ra,.L2
.L4:
        lw      a4,-36(s0)
        addi      a5,zero,2
        bne     a4,a5,.L2
        lw      a4,-40(s0)
        lw      a5,-44(s0)
        bge     a5,a4,.L5
        addi      a5,zero,1
        sw      a5,-20(s0)
        jal       ra,.L2
.L5:
        lw      a4,-36(s0)
        addi      a5,zero,3
        bne     a4,a5,.L2
        lw      a4,-52(s0)
        lw      a5,-48(s0)
        bgeu    a4,a5,.L6
        addi      a5,zero,1
        sw      a5,-20(s0)
        jal       ra,.L2
.L6:
        lw      a4,-36(s0)
        addi      a5,zero,4
        bne     a4,a5,.L2
        lw      a4,-52(s0)
        lw      a5,-48(s0)
        bltu    a4,a5,.L7
        addi      a5,zero,1
        sw      a5,-20(s0)
        jal       ra,.L2
.L7:
        lw      a4,-36(s0)
        addi      a5,zero,5
        bne     a4,a5,.L2
        lw      a4,-44(s0)
        lw      a5,-40(s0)
        blt     a4,a5,.L8
        addi      a5,zero,1
        sw      a5,-20(s0)
        jal       ra,.L2
.L8:
        lw      a4,-36(s0)
        addi      a5,zero,6
        bne     a4,a5,.L9
        lw      a4,-40(s0)
        lw      a5,-44(s0)
        slt     a5,a4,a5
        andi    a5,a5,255
        sw      a5,-20(s0)
        jal       ra,.L2
.L9:
        lw      a4,-36(s0)
        addi      a5,zero,7
        bne     a4,a5,.L10
        lw      a5,-40(s0)
        slti    a5,a5,-1
        andi    a5,a5,255
        sw      a5,-20(s0)
        jal       ra,.L2
```

```
        jal     ra,.L2
.L10:
        lw      a4,-36(s0)
        addi      a5,zero,8
        bne     a4,a5,.L11
        lw      a5,-40(s0)
        lw      a4,-48(s0)
        sltu   a5,a5,a4
        andi    a5,a5,255
        sw      a5,-20(s0)
        jal      ra,.L2
.L11:
        lw      a4,-36(s0)
        addi      a5,zero,9
        bne     a4,a5,.L2
        lw      a4,-40(s0)
        lw      a5,-44(s0)
        slt     a5,a4,a5
        andi    a5,a5,255
        sw      a5,-20(s0)
.L2:
        lw      a5,-20(s0)
        addi     a0,a5,0
        lw      s0,60(sp)
        addi    sp,sp,64
        jalr     x0, 0(ra)
main:
        addi    sp,sp,-32
        sw      ra,28(sp)
        sw      s0,24(sp)
        addi    s0,sp,32
        addi      a5,zero,1
        sw      a5,-20(s0)
        sw      zero,-24(s0)
        sw      zero,-28(s0)
        addi      a5,zero,1
        sw      a5,-32(s0)
        lw      a4,-32(s0)
        lw      a3,-28(s0)
        lw      a2,-24(s0)
        lw      a1,-20(s0)
        addi      a0,zero,0
        jal    ra,    ComputeLogic
        lw      a4,-32(s0)
        lw      a3,-28(s0)
        lw      a2,-24(s0)
        lw      a1,-20(s0)
        addi      a0,zero,9
        jal    ra,    ComputeLogic
        lw      a4,-32(s0)
        lw      a3,-28(s0)
        lw      a2,-24(s0)
        lw      a1,-20(s0)
        addi      a0,zero,8
        jal    ra,    ComputeLogic
        lw      a4,-32(s0)
        lw      a3,-28(s0)
        lw      a2,-24(s0)
        lw      a1,-20(s0)
        addi      a0,zero,7
        jal    ra,    ComputeLogic
        lw      a4,-32(s0)
        lw      a3,-28(s0)
        lw      a2,-24(s0)
        lw      a1,-20(s0)
        addi      a0,zero,6
        jal    ra,    ComputeLogic
        lw      a4,-32(s0)
        lw      a3,-28(s0)
        lw      a2,-24(s0)
        lw      a1,-20(s0)
        addi      a0,zero,5
        jal    ra,    ComputeLogic
        lw      a4,-32(s0)
        lw      a3,-28(s0)
        lw      a2,-24(s0)
        lw      a1,-20(s0)
        addi      a0,zero,4
```

```
addi      a0,zero,4
jal    ra,    ComputeLogic
lw      a4,-32(s0)
lw      a3,-28(s0)
lw      a2,-24(s0)
lw      a1,-20(s0)
addi      a0,zero,3
jal    ra,    ComputeLogic
lw      a4,-32(s0)
lw      a3,-28(s0)
lw      a2,-24(s0)
lw      a1,-20(s0)
addi      a0,zero,2
jal    ra,    ComputeLogic
lw      a4,-32(s0)
lw      a3,-28(s0)
lw      a2,-24(s0)
lw      a1,-20(s0)
addi      a0,zero,1
jal    ra,    ComputeLogic
addi      a0,zero,0
ecall
```

3. **Program 3: Looping Structure**
    - A program that includes a loop, demonstrating the capability of the simulator to handle iterative structures. C++:

```
 unsigned int factorial(unsigned int x)
{
   if ( x== 0)
      return 1;
   return x * factorial(x - 1);
}

int main()
{
   int x = 5;
   int result= factorial(x);
}
```

Assembly:

```
fact:
   addi  sp, sp, -8
   sw ra, 0(sp)
   addi t0, zero, 2
   blt a0, t0, ret_one
   sw a0, 4(sp)
   addi a0, a0, -1
   jal ra, fact
   lw t0, 4(sp)
   mul a0, t0, a0
   beq zero,zero,done
ret_one:
   addi a0, zero, 1
done:
   lw ra, 0(sp)
   addi sp, sp, 8
   jalr zero,0(ra)

main:
   addi a0,zero, 5
   jal ra,fact
   addi a1, zero, 96
   sw a0,0(a1)
   fence
```

```
```

Conclusion:

Our RISC-V simulator provides a robust and versatile platform for simulating RISC-V programs. The modular design, error handling, and bonus features contribute to a comprehensive tool for educational and research purposes. Continuous testing and feedback will be crucial for further enhancements and bug fixes.