# Project 2: **RISC-V Simulator**

## 1. Description

The goal of this project is to write a simulator program that simulates the RISC-V microarchitecture (single cycle execution). You need to model the basic operations of the RISC-V microarchitecture, such as fetching instructions from the instruction memory, decoding instructions, executing the proper arithmetic operations, reading data from the register file or the data memory, and determining the address of the next instructions. Your simulator will read an executable file (e.g., runme.hex) that contains several instructions represented in hexadecimal.

The simulator should have two modes: debug and run mode. In the debug mode, the simulator will show the clock cycles, PC, and register values after executing an instruction every cycle. In the run mode, your simulator will run the given executable file and show the clock cycles, PC, and register values at the end of the simulation.

You can download a zip file (P2_codes.zip) that includes skeleton code (riscv_sim.c) and several executable files (addi_test.hex, add_test.hex, jal_test.hex, jalr_test.hex, beq_test.hex, sd_ld_test.hex, and runme.hex) from the project section of the LMS. You can use the skeleton code to write your simulator program. The files named "(*_test.hex)" will be useful for testing your simulator to ensure that it can execute the individual instruction correctly.

Please upload the **"riscv_sim.c"** file and a **"report"** to the LMS. In the report, you need to explain your code briefly.

## 2. Instruction you must simulate

add, addi, jal, jalr, ld, sd, beq

## 3. How to run

./riscv_sim ./runme.hex 0 // debug mode

./riscv _sim ./runme.hex 1 // run mode

# 4. Expected output of your simulator

## a. Debug mode

* All numbers are decimal number

```
$./riscv_sim runme.hex 0

-------------------------------------------------
Clock cycles = 1
PC        = ?

x0   = 0
x1   = 0
x2   = ?
x3   = 0
x4   = ?
x5   = 0
x6   = 0
x7   = 0
x8   = ?
x9   = 0
x10  = ?
x11  = 0
x12  = 0
x13  = 0
x14  = 0
x15  = 0
x16  = 0
x17  = 0
x18  = 0
x19  = 0
x20  = 0
x21  = 0
x22  = 0
x23  = 0
x24  = 0
x25  = 0
x26  = 0
x27  = 0
x28  = 0
x29  = ?
x30  = 0
x31  = ?

-------------------------------------------------
Clock cycles = 2
PC        = ?

x0   = 0
x1   = 0
x2   = ?
x3   = 0
x4   = ?
x5   = 0
x6   = 0
x7   = 0
x8   = ?
x9   = 0
x10  = ?
```

```
x11  = 0
x12  = 0
x13  = 0
x14  = 0
x15  = 0
x16  = 0
x17  = 0
x18  = 0
x19  = 0
x20  = 0
x21  = 0
x22  = 0
x23  = 0
x24  = 0
x25  = 0
x26  = 0
x27  = 0
x28  = 0
x29  = ?
x30  = 0
x31  = ?

------------------------------------------------
Clock cycles = 3
PC        = ?

x0   = 0
x1   = ?
x2   = ?
x3   = 0
x4   = ?
x5   = 0
x6   = 0
x7   = 0
x8   = ?
x9   = 0
x10  = ?
x11  = 0
x12  = 0
x13  = 0
x14  = 0
x15  = 0
x16  = 0
x17  = 0
x18  = 0
x19  = 0
x20  = 0
x21  = 0
x22  = 0
x23  = 0
x24  = 0
x25  = 0
x26  = 0
x27  = 0
x28  = 0
x29  = ?
x30  = 0
x31  = ?
………
```

## b. run mode

* clock cycle is decimal number

```
$./riscv_sim runme.hex 1

Clock cycles = ?
PC        = ?

x0   = 0
x1   = 12
x2   = 800
x3   = 0
x4   = 0
x5   = 0
x6   = 45
x7   = 0
x8   = 0
x9   = 10
x10  = 55
x11  = 0
x12  = 0
x13  = 0
x14  = 0
x15  = 0
x16  = 0
x17  = 0
x18  = 0
x19  = 0
x20  = 0
x21  = 0
x22  = 0
x23  = 0
x24  = 0
x25  = 0
x26  = 0
x27  = 0
x28  = 0
x29  = 0
x30  = 0
x31  = 0
```

## 5. Testing

You can find several executable files ("*_test.hex") that you can use to test your simulator.

- Expected output of your simulator for each executable file

  a. addi_test.hex

```
$./riscv_sim addi_test.hex 1

Clock cycles = 32
PC        = 128

x0    = 0
x1    = 10
x2    = 10
x3    = 10
x4    = 10
x5    = 10
x6    = 10
x7    = 10
x8    = 10
x9    = 10
x10   = 10
x11   = 10
x12   = 10
x13   = 10
x14   = 10
x15   = 10
x16   = 10
x17   = 10
x18   = 10
x19   = 10
x20   = 10
x21   = 10
x22   = 10
x23   = 10
x24   = 10
x25   = 10
x26   = 10
x27   = 10
x28   = 10
x29   = 10
x30   = 10
x31   = 10
```

b. add_test.hex

```
$./riscv_sim add_test.hex 1

Clock cycles = 4
PC        = 16

x0    = 0
x1    = 0
x2    = 0
x3    = 0
x4    = 0
x5    = 0
x6    = 0
x7    = 0
x8    = 0
x9    = 10
x10   = 0
x11   = 0
x12   = 0
x13   = 0
x14   = 0
x15   = 0
x16   = 10
x17   = 20
x18   = 30
x19   = 0
x20   = 0
x21   = 0
x22   = 0
x23   = 0
x24   = 0
x25   = 0
x26   = 0
x27   = 0
x28   = 0
x29   = 0
x30   = 0
x31   = 0
```

c. jal_test.hex

```
$./riscv_sim jal_test.hex 1

Clock cycles = 3
PC      = 16

x0    = 0
x1    = 4
x2    = 0
x3    = 0
x4    = 0
x5    = 0
x6    = 0
x7    = 0
x8    = 0
x9    = 10
x10   = 0
x11   = 0
x12   = 0
x13   = 0
x14   = 0
x15   = 0
x16   = 0
x17   = 20
x18   = 0
x19   = 0
x20   = 0
x21   = 0
x22   = 0
x23   = 0
x24   = 0
x25   = 0
x26   = 0
x27   = 0
x28   = 0
x29   = 0
x30   = 0
x31   = 0
```

d. jalr_test.hex

```
$./riscv_sim jalr_test.hex 1

Clock cycles = 7
PC      = 36

x0    = 0
x1    = 0
x2    = 0
x3    = 0
x4    = 0
x5    = 0
x6    = 0
x7    = 0
x8    = 0
x9    = 10
x10   = 0
x11   = 0
x12   = 0
x13   = 0
x14   = 0
x15   = 0
x16   = 10
x17   = 20
x18   = 0
x19   = 20
x20   = 0
x21   = 0
x22   = 0
x23   = 0
x24   = 0
x25   = 0
x26   = 0
x27   = 0
x28   = 0
x29   = 0
x30   = 0
x31   = 4
```

e. sd_ld_test.hex

```
$./riscv_sim sd_ld_test.hex 1

Clock cycles = 5
PC      = 20

x0    = 0
x1    = 0
x2    = 0
x3    = 0
x4    = 30
x5    = 0
x6    = 0
x7    = 0
x8    = 0
x9    = 10
x10   = 0
x11   = 0
x12   = 0
x13   = 0
x14   = 0
x15   = 0
x16   = 30
x17   = 0
x18   = 0
x19   = 0
x20   = 0
x21   = 0
x22   = 0
x23   = 0
x24   = 0
x25   = 0
x26   = 0
x27   = 0
x28   = 0
x29   = 0
x30   = 0
x31   = 0
```

f. beq_test.hex

```
$./riscv_sim beq_test.hex 1

Clock cycles = 4
PC      = 20

x0    = 0
x1    = 0
x2    = 0
x3    = 0
x4    = 0
x5    = 0
x6    = 0
x7    = 0
x8    = 0
x9    = 10
x10   = 0
x11   = 0
x12   = 0
x13   = 0
x14   = 0
x15   = 0
x16   = 30
x17   = 30
x18   = 0
x19   = 0
x20   = 0
x21   = 0
x22   = 0
x23   = 0
x24   = 0
x25   = 0
x26   = 0
x27   = 0
x28   = 0
x29   = 0
x30   = 0
x31   = 0
```

6. **Evaluation** [total: 100 points]

   a. **Unit test** [45 points] **:** test your simulator to ensure that it can execute individual instruction (add, addi, jal, jalr, ld, sd, beq) correctly.

   b. **Full test** [55 points] **:** test your simulator to ensure that it can correctly execute the given executable file. We will use the “runme.hex” file for the full test.

**Late Day Policy**

No late submission is allowed.

**Plagiarism**

No plagiarism will be tolerated. If the instructor determines that there are substantial similarities between your code and others, you will be given 0 points for all project assignments. Code similarity will be checked with a plagiarism detector tool.

# Appendix

| Category | Instruction | Example | Meaning | Comments |
|---|---|---|---|---|
| Arithmetic | Add | add x5, x6, x7 | x5 = x6 + x7 | Three register operands; add |
| | Subtract | sub x5, x6, x7 | x5 = x6 - x7 | Three register operands; subtract |
| | Add immediate | addi x5, x6, 20 | x5 = x6 + 20 | Used to add constants |
| Data transfer | Load doubleword | ld x5, 40(x6) | x5 = Memory[x6 + 40] | Doubleword from memory to register |
| | Store doubleword | sd x5, 40(x6) | Memory[x6 + 40] = x5 | Doubleword from register to memory |
| | Load word | lw x5, 40(x6) | x5 = Memory[x6 + 40] | Word from memory to register |
| | Load word, unsigned | lwu x5, 40(x6) | x5 = Memory[x6 + 40] | Unsigned word from memory to register |
| | Store word | sw x5, 40(x6) | Memory[x6 + 40] = x5 | Word from register to memory |
| | Load halfword | lh x5, 40(x6) | x5 = Memory[x6 + 40] | Halfword from memory to register |
| | Load halfword, unsigned | lhu x5, 40(x6) | x5 = Memory[x6 + 40] | Unsigned halfword from memory to register |
| | Store halfword | sh x5, 40(x6) | Memory[x6 + 40] = x5 | Halfword from register to memory |
| | Load byte | lb x5, 40(x6) | x5 = Memory[x6 + 40] | Byte from memory to register |
| | Load byte, unsigned | lbu x5, 40(x6) | x5 = Memory[x6 + 40] | Byte unsigned from memory to register |
| | Store byte | sb x5, 40(x6) | Memory[x6 + 40] = x5 | Byte from register to memory |
| | Load reserved | lr.d x5, (x6) | x5 = Memory[x6] | Load; 1st half of atomic swap |
| | Store conditional | sc.d x7, x5, (x6) | Memory[x6] = x5; x7 = 0/1 | Store; 2nd half of atomic swap |
| | Load upper immediate | lui x5, 0x12345 | x5 = 0x12345000 | Loads 20-bit constant shifted left 12 bits |
| Logical | And | and x5, x6, x7 | x5 = x6 & x7 | Three reg. operands; bit-by-bit AND |
| | Inclusive or | or x5, x6, x8 | x5 = x6 \| x8 | Three reg. operands; bit-by-bit OR |
| | Exclusive or | xor x5, x6, x9 | x5 = x6 ^ x9 | Three reg. operands; bit-by-bit XOR |
| | And immediate | andi x5, x6, 20 | x5 = x6 & 20 | Bit-by-bit AND reg. with constant |
| | Inclusive or immediate | ori x5, x6, 20 | x5 = x6 \| 20 | Bit-by-bit OR reg. with constant |
| | Exclusive or immediate | xori x5, x6, 20 | x5 = x6 ^ 20 | Bit-by-bit XOR reg. with constant |
| Shift | Shift left logical | sll x5, x6, x7 | x5 = x6 << x7 | Shift left by register |
| | Shift right logical | srl x5, x6, x7 | x5 = x6 >> x7 | Shift right by register |
| | Shift right arithmetic | sra x5, x6, x7 | x5 = x6 >> x7 | Arithmetic shift right by register |
| | Shift left logical immediate | slli x5, x6, 3 | x5 = x6 << 3 | Shift left by immediate |
| | Shift right logical immediate | srli x5, x6, 3 | x5 = x6 >> 3 | Shift right by immediate |
| | Shift right arithmetic immediate | srai x5, x6, 3 | x5 = x6 >> 3 | Arithmetic shift right by immediate |

| | | | | |
|---|---|---|---|---|
| Conditional branch | Branch if equal | beq x5, x6, 100 | if (x5 == x6) go to PC+100 | PC-relative branch if registers equal |
| | Branch if not equal | bne x5, x6, 100 | if (x5 != x6) go to PC+100 | PC-relative branch if registers not equal |
| | Branch if less than | blt x5, x6, 100 | if (x5 < x6) go to PC+100 | PC-relative branch if registers less |
| | Branch if greater or equal | bge x5, x6, 100 | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal |
| | Branch if less, unsigned | bltu x5, x6, 100 | if (x5 < x6) go to PC+100 | PC-relative branch if registers less, unsigned |
| | Branch if greater or equal, unsigned | bgeu x5, x6, 100 | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal, unsigned |
| Unconditional branch | Jump and link | jal x1, 100 | x1 = PC+4; go to PC+100 | PC-relative procedure call |
| | Jump and link register | jalr x1, 100(x5) | x1 = PC+4; go to x5+100 | Procedure return; indirect call |

| Name | Field | | | | | | Comments |
|---|---|---|---|---|---|---|---|
| (Field size) | 7 bits | 5 bits | 5 bits | 3 bits | 5 bits | 7 bits | |
| R-type | funct7 | rs2 | rs1 | funct3 | rd | opcode | Arithmetic instruction format |
| I-type | immediate[11:0] | | rs1 | funct3 | rd | opcode | Loads & immediate arithmetic |
| S-type | immed[11:5] | rs2 | rs1 | funct3 | immed[4:0] | opcode | Stores |
| SB-type | immed[12,10:5] | rs2 | rs1 | funct3 | immed[4:1,11] | opcode | Conditional branch format |
| UJ-type | immediate[20,10:1,11,19:12] | | | | rd | opcode | Unconditional jump format |
| U-type | immediate[31:12] | | | | rd | opcode | Upper immediate format |

| Format | Instruction | Opcode | Funct3 | Funct6/7 |
| --- | --- | --- | --- | --- |
| R-type | add | 0110011 | 000 | 0000000 |
| | sub | 0110011 | 000 | 0100000 |
| | sll | 0110011 | 001 | 0000000 |
| | xor | 0110011 | 100 | 0000000 |
| | srl | 0110011 | 101 | 0000000 |
| | sra | 0110011 | 101 | 0100000 |
| | or | 0110011 | 110 | 0000000 |
| | and | 0110011 | 111 | 0000000 |
| | lr.d | 0110011 | 011 | 0001000 |
| | sc.d | 0110011 | 011 | 0001100 |
| I-type | lb | 0000011 | 000 | n.a. |
| | lh | 0000011 | 001 | n.a. |
| | lw | 0000011 | 010 | n.a. |
| | ld | 0000011 | 011 | n.a. |
| | lbu | 0000011 | 100 | n.a. |
| | lhu | 0000011 | 101 | n.a. |
| | lwu | 0000011 | 110 | n.a. |
| | addi | 0010011 | 000 | n.a. |
| | slli | 0010011 | 001 | 000000 |
| | xori | 0010011 | 100 | n.a. |
| | srli | 0010011 | 101 | 000000 |
| | srai | 0010011 | 101 | 010000 |
| | ori | 0010011 | 110 | n.a. |
| | andi | 0010011 | 111 | n.a. |
| | jalr | 1100111 | 000 | n.a. |

| Format | Instruction | Opcode | Funct3 | Funct6/7 |
| --- | --- | --- | --- | --- |
| S-type | sb | 0100011 | 000 | n.a. |
| | sh | 0100011 | 001 | n.a. |
| | sw | 0100011 | 010 | n.a. |
| | sd | 0100011 | 011 | n.a. |
| SB-type | beq | 1100011 | 000 | n.a. |
| | bne | 1100011 | 001 | n.a. |
| | blt | 1100011 | 100 | n.a. |
| | bge | 1100011 | 101 | n.a. |
| | bltu | 1100011 | 110 | n.a. |
| | bgeu | 1100011 | 111 | n.a. |
| U-type | lui | 0110111 | n.a. | n.a. |
| UJ-type | jal | 1101111 | n.a. | n.a. |