# Test Strategy

## RISC-V ISA Simulator

## Design Contribution:

AUIPC, BNE, BGEU, LBU, SW, XORI, SRLI, SLTU, OR, ADD - **Abhigna**

LUI, BEQ, BLTU, LW, SH, SLTIU, SLLI, SLT, SRA - **Manikandeshwar**

JAL, BLT, LB, LHU, ADDI, ORI, SRAI, XOR, SUB - **Nishka**

JALR, BGE, LH, SB, SLTI, ANDI, SLL, SRL, AND - **Swetha**

## Verification Contribution:

AUIPC, BNE, BGEU, LBU, SW, XORI, SRLI, SLTU, OR, ADD - **Manikandeshwar**

LUI, BEQ, BLTU, LW, SH, SLTIU, SLLI, SLT, SRA - **Swetha**

JAL, BLT, LB, LHU, ADDI, ORI, SRAI, XOR, SUB - **Abhigna**

JALR, BGE, LH, SB, SLTI, ANDI, SLL, SRL, AND - **Nishka**

### 1. LUI

- Tested with basic value by loading 0xABCDE. Gives result in rd as 0xABCDE.
- Tested with negative immediate value by loading 0xFFFED. Gives result in rd as 0xFFFED000
- Tested with largest positive immediate value by loading 0xFFFFF000. Gives result in rd as 0xFFFFF000.
- Tested with largest negative value by loading 0x80000000. Gives result in rd as 0x80000000.
- Tested with all zeros. Gives result in rd as 0x00000000.

### 2. AUIPC

- Test to see if adds PC value properly
- Test to see if sign extended immediate works on negative numbers
- Test when PC = 0, Imm = 0, PC & Imm == 0

### 3. JAL

- Test that the instruction jumps to the correct address when the immediate is positive
- Test that the instruction jumps to the correct address when the immediate is negative
- Test that the instruction updates the PC with the correct return address
- Test that the instruction correctly handles the 0 immediate

### 4. JALR

- Test to check if the instruction jumps to the correct address when the immediate is positive
- Test to check if the Imm field gets sign extended correctly
- Test to check if instruction jumps to the correct address when the immediate is negative
- Test to check if the instruction correctly handles the 0 immediate.
- Test to check if the instruction updates the PC with the correct return address

# 5. BEQ

- Tested for simple branching.
- Tested for multiple branching.
- Tested with Imm=0. This makes branch enter into infinite loop.
- Tested with unaligned Imm=2. Raises Trap Exception.
- Tested with Imm=-1. Raises Trap Exception.

# 6. BNE

- Test to see if PC jumps backwards as well as forwards
- Test to see when both operands are 'hf, 'h0, 'ha, 'h5
- Introduce infinite loop conditions (not allowed)

# 7. BLT

- Test that the instruction branches to the correct address when the comparison is true
- Test that the instruction doesn't branch when the comparison is false
- Test that the instruction correctly handles the edge case where the two operands are equal
- Test that the instruction correctly handles negative numbers

# 8. BGE

- Test to see if PC jumps backwards as well as forwards
- Test case with both operands equal
- Test case with both operands different and the second operand greater than the first
- Test case with both operands different and the first operand greater than the first

# 9. BLTU

- Tested with basic branching.
- Tested with rs1=-1 and rs2=1. Not taking Branch.
- Tested with rs1=-1 and rs2=-1. Equal Condition. Not taking Branch.
- Tested with rs1=0 and rs2=0. Not taking Branch.
- Tested with rs1=-79 and rs2=-19. Taking Branch.

# 10. BGEU

- Test for greater condition
- Test for equal condition
- Test when rs1 is -ve and is STILL greater than rs2 (unsigned)
- Test if PC jumps backwards as well as forwards

# 11. LB

- Test that the instruction loads the correct byte value from memory
- Test that the instruction correctly handles the sign extension for negative values
- Test that the instruction correctly handles the edge cases of loading the first or last byte of a word
- Test that the instruction correctly handles misaligned memory accesses(not required)
- Test if RD=0 is allowed?(It shouldn't be)

# 12. LH

- Test to check if the instruction loads the correct half-word value from memory
- Test to check if the instruction correctly handles the sign extension for negative Imm
- Test to check if the instruction correctly handles loading the first or last half-word of a word
- Test that the instruction correctly handles misaligned memory accesses(not required)
- Test if RD=0 is allowed?(It shouldn't be)

# 13. LW

- Tested for simple basic load operation by giving Imm=1F.
- Tested with Imm=0.
  - Tested with Imm=-1. Works fine.
  - Unaligned Load do print Exception Message.
- Test if RD=0 is allowed?(It shouldn't be)
-

# 14. LBU

- Test to see if correct memory locations are being accessed
- Test to see if no incorrect memory locations are being accessed
- Check that no sign extension takes place by loading a value beginning with 1
- Test if RD=0 is allowed?(It shouldn't be)

# 15. LHU

- Test that the instruction loads the correct half-word value from memory
- Test that the instruction correctly handles the zero extension for given values
- Test that the instruction correctly handles the edge cases of loading the first or last half-word of a word
- Test that the instruction correctly handles misaligned memory accesses(not required)

- Test if RD=0 is allowed?(It shouldn't be)

# 16. SB

- Test to check if the instruction stores the correct byte value from memory into the destination register
- Test to check if the instruction correctly handles the sign extension for negative Imm
- Test to check if the instruction correctly handles stores a byte of a word
- Test that the instruction correctly handles misaligned memory accesses (not required)
- Tested with a negative IMM

# 17. SH

- Tested with Imm=0.
- Tested with Imm=-1.
- Unaligned memory references do not print Exception Message.
- Loaded Memory location with a Word and tested to check if only Half-Word gets loaded.

# 18. SW

- Test to see that the 32bit value is stored accurately over 4 memory locations
- Test to see that an illegal memory location is not being accessed
- Test to see if signed calculations can take place for address calculations and storing

# 19. ADDI

- Test that the instruction adds the correct immediate value to the register
- Test that the instruction correctly handles overflow and underflow
- Test that the instruction correctly updates the rd
- Test that the instruction correctly handles the edge case of adding 0
- Test for both positive and negative opearnds

# 20. SLTI

- Test to check when source register is less than the immediate value.
- Test to check when source register is greater than the immediate value.
- Test to check when source register is equal to the immediate value.
- Test to check if the sign extension is correct for a negative immediate value
- Tested a case where the destination register is the same as the source register.

# 21. SLTIU

- Tested the following:

rs1=+79 and rs2=-19. Sets rd=0

rs1=-79 and rs2=-19. Sets rd=0

rs1=-79 and rs2=+19. Sets rd=0

rs1=+79 and rs2=+19. Sets rd=0

rs1=79 an rs2=79. Sets rd=0

rs1=-19 and rs2=+79. Sets rd=1

rs1=-19 and rs2=-79. Sets rd=1

rs1=+19 and rs2=-79. Sets rd=1

rs1=+19 and rs2=+79. Sets rd=1

# 22. XORI

- Test to see if corner cases of f, a,0,5 xor with each other
- Test to see if immediate field is sign extended

# 23. ORI

- Test that the instruction performs the correct bitwise OR operation with the immediate value
- Test that the instruction correctly updates the rd
- Test that the instruction correctly handles the edge case of OR-ing with 0
- Test that the instruction correctly handles OR-ing with a negative value

# 24. ANDI

- Test to check whether Imm field is sign extending as expected
- Test case with small positive numbers
- Test case with large positive numbers
- Test case with negative numbers
- Test case with all bits set to 1

# 25. SLLI

- Tested the following:

rs1=+79 and Imm=-19. Takes Imm value as a positive decimal value.

rs1=-79 and Imm=+19.

rs1=+79 and Imm=+19

rs1=-79 and Imm=-19 Takes Imm value as a positive decimal value.

## 26. SRLI

- Test to see if LOGICAL shift takes place (default in cpp is arithmetic)
- Test where immediate field is zero (no shift)
- Test where register value is zero (answer is zero)
- Test of 32'hf where the register can never be completely right shifted out

## 27. SRAI

- Test that the instruction performs the correct arithmetic shift right operation with the immediate value
- Test that the instruction correctly updates the rd.
- Test that the instruction correctly handles the edge case of shifting by 0
- Test that the instruction correctly handles shifting negative values

## 28. ADD

- Test to see if the operation is signed
- Test where one register is 'b0
- Test where both are zero
- Test where a register is the negative of the other to see if 0 is obtained in rd

## 29. SUB

- Test that the instruction performs the correct subtraction operation between two registers
- Test that the instruction correctly handles overflow and underflow
- Test that the instruction correctly updates the rd
- Test the negative number and positive number combinations
- Test that the instruction correctly handles the edge case of subtracting 0

## 30. SLL

- Tests to check whether the lower 5 bits is getting extracted from the rs2
- Input value is zero and the shift amount is zero
- Input value is all ones, and the shift amount is zero
- Test with some input value and shift amount is equal to 31
- Input value is a positive number, and the shift amount is 1
- Test with random values and shift amounts within the range of 0 to 31

## 31. SLT

- Tested with basic values.
- Following are the test cases:

rs1=+79 and rs2=-19. Sets rd=0

rs1=-79 and rs2=-19. Sets rd=1

rs1=-79 and rs2=+19. Sets rd=1

rs1=+79 and rs2=+19. Sets rd=0

rs1=-1 and rs2=0. Sets rd=1.

rs1=0 and rs2=0. Sets rd=0.

# 32. SLTU

- Test for one operand zero
- Test for both operands negative (unsigned so converted to positive
- Test for both operands equal
- Test for both operands 0

# 33. XOR

- Test that the instruction performs the correct bitwise XOR operation between two registers
- Test that the instruction correctly updates the rd
- Test that the instruction correctly handles the edge case of XOR-ing with 0
- Test that the instruction correctly handles the edge case of XOR-ing with 1

# 34. SRL

- Tests to check whether the lower 5 bits is getting extracted from the rs2
- Input value is zero and the shift amount is zero
- Input value is all ones, and the shift amount is zero
- Test with some input value and shift amount is equal to 31
- Input value is a positive number, and the shift amount is 1
- Test with random values and shift amounts within the range of 0 to 31

# 35. SRA

- Tested with basic shifting.
- Tested with MSB=1 value.
- Tested for the following values:

rs1=-79 and rs2=4.

rs1=+79 and rs2=4.

rs1=-79 and rs2=-4. Lower 5 bits are taken and treated as positive decimal shift amount.

rs1=+79 and rs2=-4. Lower 5 bits are taken and treated as positive decimal shift amount.

rs1=+79 and rs2=-1. Lower 5 bits are taken and treated as positive decimal shift amount.

# 36. OR

- Test where one register is 'b0
- Test where one register is 'b1
- Test where one register is 'ba and another is 'b5

# 37. AND

- **Test 1: Operand1 is all 0's and Operand 2 is all 0's**
- **Test 2: Operand 1 is all 0's and Operand 2 is all 1's**
- **Test 3: Operand 1 and Operand 2 are large positive numbers**

# 38. MUL

- Tested with following test cases:

rs1=0x97812451 and rs2=0x78964129. Sets rd=0x4D5D61F9

rs1=-1 and rs2=-1. Sets rd=1.

rs1=0x 12393 and rs2=0x23456. Sets rd=0x82C1CF62.

rs1=0 and rs2=0. Sets rd=0

rs1=0x59687999 and rs2=0. Sets rd=0

rs1= 0x59687999 and rs2=0. Sets rd=0

# 39. MULH

- Tested for some basic test cases.
- Tested for a value that will result in MSB( of 64-bit number)=1. So to check if right shift leads to any sign extension.

# 40. MULHSU

- Test that the instruction performs the correct signed multiplication operation between a signed and an unsigned register.
- Test that the instruction correctly handles overflow and underflow
- Test that the instruction correctly updates the rd
- Test that the instruction correctly handles the edge case of multiplying by 0

# 41. MULHU

- Test that the instruction performs the correct unsigned multiplication operation between two unsigned registers
- Test that the instruction correctly handles overflow and underflow
- Test that the instruction correctly updates the rd
- Test that the instruction correctly handles the edge case of multiplying by 0

# 42. DIV

- Test for rs1=rs2
- Test for –ve/-ve +ve/+ve –ve/+ve +ve/-ve
- Test for rs2 = 0
- Test for rs1=0

# 43. DIVU

- Test to confirm that the operation is unsigned
- Test with rs1 = 0
- Test with rs2 = 0
- Test with rs1=rs2

# 44. REM

- Test to check basic functionality
- Test to check when operands are–ve/-ve +ve/+ve –ve/+ve +ve/-ve
- Test with dividend as 0
- Test with divisor as 0
- Test to check if the remainder is taking the sign of the quotient
- Test to check the result when overflow occurs (rs1 = 80000000 and rs2 = FFFFFFFF, Floting point Exception occurs)

# 45. REMU

- Test to check if the operation is unsigned
- Test with the dividend as 0
- Test with the divisor as 0