TEST PLAN #1 - Generic test plan

I - TYPE

1. ADDI (Add Immediate)

Test Case 1:

- **Input:** 0x00C30313
- **Decoded:** ADDI rd=6, rs1=6, imm=12
- **Expected Output:** register[6] = 0x1C (0x10 + 0xC)

Test Case 2:

- **Input:** 0xFFF28293
- **Decoded:** ADDI rd=5, rs1=5, imm=-1
- **Expected Output:** register[5] = 0x0

2. SLTI (Set Less Than Immediate - Signed)

Test Case 1:

- **Input:** 0x00230213
- **Decoded:** SLTI rd=4, rs1=6, imm=2
- **Expected Output:** register[4] = 1 (1 < 2)

Test Case 2:

- **Input:** 0xFF830313
- **Decoded:** SLTI rd=6, rs1=6, imm=-8
- **Expected Output:** register[6] = 0 (5 is not less than -8)

3. SLTIU (Set Less Than Immediate - Unsigned)

Test Case 1:

- **Input:** 0x00230313
- **Decoded:** SLTIU rd=6, rs1=6, imm=2
- **Expected Output:** register[6] = 1

Test Case 2:

- **Input:** 0xFFF30313
- **Decoded:** SLTIU rd=6, rs1=6, imm=-1 (0xFFFFFFFF)
- **Expected Output:** register[6] = 1 (0x7FFFFFFF < 0xFFFFFFFF)

4. XORI (XOR Immediate)

Test Case 1:

- **Input:** 0x00F28293
- **Decoded:** XORI rd=5, rs1=5, imm=15
- **Expected Output:** register[5] = 0xA (0x5 ^ 0xF)

Test Case 2:

- **Input:** 0x00F38393
- **Decoded:** XORI rd=7, rs1=7, imm=15
- **Expected Output:** register[7] = 0x00

5. ORI (OR Immediate)

Test Case 1:

- **Input:** 0x00F282B3
- **Decoded:** ORI rd=5, rs1=5, imm=15
- **Expected Output:** register[5] = 0xF (0x5 | 0xF)

6. ANDI (AND Immediate)

Test Case 1:

- **Input:** 0x00F30313
- **Decoded:** ANDI rd=6, rs1=6, imm=0x0F
- **Expected Output:** register[6] = 0x0F (0xFF & 0x0F)

7. SLLI (Shift Left Logical Immediate)

Test Case 1:

- **Input:** 0x00228293
- **Decoded:** SLLI rd=5, rs1=5, imm=2
- **Expected Output:** register[5] = 0x4

8. SRLI (Shift Right Logical Immediate)

Test Case 1:

- **Input:** 0x00229293
- **Decoded:** SRLI rd=5, rs1=5, imm=2
- **Expected Output:** register[5] = 0x4

9. SRAI (Shift Right Arithmetic Immediate)

Test Case 1:

- **Input:** 0x40229293
- **Decoded:** SRAI rd=5, rs1=5, imm=2
- **Expected Output:** register[5] = -4 (0xFFFFFFFC)

10. Load Byte (LB)

Test Case 1: imm = 0

Setup:

- Memory:
  - Address 0x0 = 0xAAAAAAAA
- Registers:
  - Register 5 = 0

Input: 0x00028283

Decoded: LB rd=r5, rs1=r5, imm=0

Expected Output: register[5] = sign-extended byte at memory[register[5]] = 0xFFFFFFAA

Test Case 2: imm > 0

Setup:

- Memory:
  - Address 0x4 = 0x55555555
- Registers:
  - Register 5 = 0
  - Register 6 = 0

Input: 0x00428303

Decoded: LB rd=r6, rs1=r5, imm=4

Expected Output: register[6] = sign-extended byte at memory[register[5]] = 0x00000055

Test Case 3: imm < 0

Setup:

- Memory:
  - Address 0x0 = 0xAAAAAAAA
- Registers:
  - Register 5 = 4
  - Register 7 = 0

Input: 0xFFC28383

Decoded: LB rd=7, rs1=5, imm=-4

Expected Output: register[6] = sign-extended byte at memory[register[5]] = 0xFFFFFFAA

11. Load Halfword (LH)

-

Test Case 1: imm = 0

Setup:

- Memory:
  - Address 0x0 = 0xAAAAAAAA
- Registers:
  - Register 5 = 0

Input: 0x00028283

Decoded: LH rd=r5, rs1=r5, imm=0

Expected Output: register[5] = sign-extended half-word at memory[register[5]] = 0xFFFFAAAA

Test Case 2: imm > 0

Setup:

- Memory:
  - Address 0x4 = 0x55555555
- Registers:
  - Register 5 = 0
  - Register 6 = 0

Input: 0x00428303

Decoded: LH rd=r6, rs1=r5, imm=4

Expected Output: register[6] = sign-extended half-word at memory[register[5]] = 0x00005555

Test Case 3: imm < 0

Setup:

- Memory:
  - Address 0x0 = 0xAAAAAAAA
- Registers:
  - Register 5 = 4
  - Register 7 = 0

Input: 0xFFC28383

Decoded: LH rd=7, rs1=5, imm=-4

Expected Output: register[6] = sign-extended half-word at memory[register[5]] = 0xFFFFAAAA

12. Load Word (LW)

Test Case 1: imm = 0

Setup:

- Memory:
    - Address 0x0 = 0xAAAAAAAA
- Registers:
    - Register 5 = 0

Input: 0x00028283

Decoded: LW rd=r5, rs1=r5, imm=0

Expected Output: register[5] = sign-extended word at memory[register[5]] = 0xAAAAAAAA

Test Case 2: imm > 0

Setup:

- Memory:
    - Address 0x4 = 0x55555555
- Registers:
    - Register 5 = 0
    - Register 6 = 0

Input: 0x00428303

Decoded: LW rd=r6, rs1=r5, imm=4

Expected Output: register[6] = sign-extended word at memory[register[5]] = 0x55555555

Test Case 3: imm < 0

Setup:

- Memory:
    - Address 0x0 = 0xAAAAAAAA
- Registers:
    - Register 5 = 4
    - Register 7 = 0

Input: 0xFFC28383

Decoded: LW rd=7, rs1=5, imm=-4

Expected Output: register[6] = sign-extended word at memory[register[5]] = 0xAAAAAAAA

13. Load Byte Unsigned (LBU)

-

Test Case 1: imm = 0

Setup:

- Memory:
  - Address 0x0 = 0xAAAAAAAA
- Registers:
  - Register 5 = 0

Input: 0x00028283

Decoded: LB rd=r5, rs1=r5, imm=0

Expected Output: register[5] = zero-extended byte at memory[register[5]] = 0x000000AA

Test Case 2: imm > 0

Setup:

- Memory:
  - Address 0x4 = 0x55555555
- Registers:
  - Register 5 = 0
  - Register 6 = 0

Input: 0x00428303

Decoded: LB rd=r6, rs1=r5, imm=4

Expected Output: register[6] = zero-extended byte at memory[register[5]] = 0x00000055

Test Case 3: imm < 0

Setup:

- Memory:
  - Address 0x0 = 0xAAAAAAAA
- Registers:
  - Register 5 = 4
  - Register 7 = 0

Input: 0xFFC28383

Decoded: LB rd=7, rs1=5, imm=-4

Expected Output: register[6] = zero-extended byte at memory[register[5]] = 0x0000000AA

14. Load Halfword Unsigned (LHU)

Test Case 1: imm = 0

Setup:

- Memory:
  - Address 0x0 = 0xAAAAAAAA
- Registers:
  - Register 5 = 0

Input: 0x00028283

Decoded: LB rd=r5, rs1=r5, imm=0

Expected Output: register[5] = zero-extended half-word at memory[register[5]] = 0x0000AAAA

Test Case 2: imm > 0

Setup:

- Memory:
  - Address 0x4 = 0x55555555
- Registers:
  - Register 5 = 0
  - Register 6 = 0

Input: 0x00428303

Decoded: LB rd=r6, rs1=r5, imm=4

Expected Output: register[6] = zero-extended half-word at memory[register[5]] = 0x00005555

Test Case 3: imm < 0

Setup:

- Memory:
  - Address 0x0 = 0xAAAAAAAA
- Registers:
  - Register 5 = 4
  - Register 7 = 0

Input: 0xFFC28383

Decoded: LB rd=7, rs1=5, imm=-4

Expected Output: register[6] = zero-extended half-word at memory[register[5]] = 0x0000AAAA

15. JALR (Jump and Link Register)

Test Case 1:

- **Input:** 0x000280E7
- **Decoded:** JALR rd=1, rs1=5, imm=0
- **Expected Output:** register[1] = return address, PC = register[5]

16. ECALL & EBREAK

Test Case 1 (ECALL):

- **Input:** 0x00000073
- **Expected Output:** "ECALL instruction executed"

Test Case 2 (EBREAK):

- **Input:** 0x00100073
- **Expected Output:** "EBREAK instruction executed"

S - TYPE

1. **Store Byte (SB)**

Test Case 1: imm = 0

Setup:

- Memory:
  - Word starting at Address 0x0 = 0x00
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xAAAAAAAA

Input: 0x00628023

Decoded: LB rs1=r5, rs2=6, imm=0

Expected Output: Word starting at memory[imm(rs1)] = 0x000000AA

Test Case 2: imm > 0

Setup:

- Memory:
  - Word starting at Address 0x4 = 0x00000000
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xBBBBBBBB

Input: 0x00628223

Decoded: LB rs1=r5, rs2=6, imm=4

Expected Output: Word starting at memory[imm(rs1)] = 0x000000BB

Test Case 3: imm < 0

Setup:

- Memory:
  - Word starting at Address 0x0 = 0x00000000
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xCCCCCCCC

Input: 0xFE628E23

Decoded: LB rs1=r5, rs2=6, imm=-4

Expected Output: Word starting at memory[imm(rs1)] = 0x000000CC

2. **Store Byte (SH)**

Test Case 1: imm = 0

Setup:

- Memory:
  - Word starting at Address 0x0 = 0x00
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xAAAAAAAA

Input: 0x00629023

Decoded: LB rs1=r5, rs2=6, imm=0

Expected Output: Word starting at memory[imm(rs1)] = 0x0000AAAA

Test Case 2: imm > 0

Setup:

- Memory:
  - Word starting at Address 0x4 = 0x00000000
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xBBBBBBBB

Input: 0x00629223

Decoded: LB rs1=r5, rs2=6, imm=4

Expected Output: Word starting at memory[imm(rs1)] = 0x0000BBBB

Test Case 3: imm < 0

Setup:

- Memory:
  - Word starting at Address 0x0 = 0x00000000
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xCCCCCCCC

Input: 0xFE629E23

Decoded: LB rs1=r5, rs2=6, imm=-4

Expected Output: Word starting at memory[imm(rs1)] = 0x0000CCCC

3. **Store Byte (Sw)**

Test Case 1: imm = 0

Setup:

- Memory:
  - Word starting at Address 0x0 = 0x00
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xAAAAAAAA

Input: 0x0062A023

Decoded: LB rs1=r5, rs2=6, imm=0

Expected Output: Word starting at memory[imm(rs1)] = 0xAAAAAAAA

Test Case 2: imm > 0

Setup:

- Memory:
  - Word starting at Address 0x4 = 0x00000000
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xBBBBBBBB

Input: 0x0062A223

Decoded: LB rs1=r5, rs2=6, imm=4

Expected Output: Word starting at memory[imm(rs1)] = 0xBBBBBBBB

Test Case 3: imm < 0

Setup:

- Memory:
  - Word starting at Address 0x0 = 0x00000000
- Registers:
  - Register 5 = 0x00
  - Register 6 = 0xCCCCCCCC

Input: 0xFE62AE23

Decoded: LB rs1=r5, rs2=6, imm=-4

Expected Output: Word starting at memory[imm(rs1)] = 0xCCCCCCCC

U - TYPE

17. LUI (Load Upper Immediate)

Test Case 1:

- **Input:** 0x00010537
- **Decoded:** LUI rd=10, imm=0x00010
- **Expected Output:** register[10] = 0x00010000 (imm shifted left by 12 bits)

Test Case 2 (Large Immediate):

- **Input:** 0xABCDE537
- **Decoded:** LUI rd=10, imm=0xABCDE
- **Expected Output:** register[10] = 0xABCDE000

Test Case 3 (Zero Immediate):

- **Input:** 0x00000537
- **Decoded:** LUI rd=10, imm=0x00000
- **Expected Output:** register[10] = 0x00000000

18. AUIPC (Add Upper Immediate to PC)

Test Case 1:

- **Input:** 0x0001017
- **Decoded:** AUIPC rd=10, imm=0x00010
- **Expected Output:** register[10] = 0x00010000 + 0x1000 = 0x101000

Test Case 2 (Large Immediate):

- **Input:** 0xABCDE517
- **Decoded:** AUIPC rd=10, imm=0xABCDE
- **Expected Output:** register[10] = 0xABCDE000 + 0x2000 = 0xABCDE2000

Test Case 3 (Zero Immediate):

- **Input:** 0x00000517
- **Decoded:** AUIPC rd=10, imm=0x00000
- **Expected Output:** register[10] = 0x4000

B - TYPE

1. **BEQ**

Test Case 1: (equal)
Input: 00208463
Decoded: BEQ x1, x2, 0x8
Expected output : PC = PC + 0x8, Branch is taken

Test Case 2: (not equal)
Input: 00208463
Decoded: BEQ x1, x2, 0x8
Expected output: PC = PC + 4, branch not taken

Test Case 3 (zero values)
Input: 00208463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

2. **BNE**

Test Case 1: (equal)
Input: 00209463
Decoded: BEQ x1, x2, 0x8
Expected output : PC = PC + 0x8, Branch is taken

Test Case 2: (not equal)
Input: 00209463
Decoded: BEQ x1, x2, 0x8
Expected output: PC = PC + 4, branch not taken

Test Case 3 (zero values)
Input: 00209463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

3. **BLT**

Test Case 1: (a<b)
Input: 0020c463
Decoded: BEQ x1, x2, 0x8
Expected output : PC = PC + 0x8, Branch is taken

Test Case 2: (b>a)
Input: 0020c463
Decoded: BEQ x1, x2, 0x8
Expected output: PC = PC + 4, branch not taken

Test Case 3 (zero values)
Input: 0020c463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

4. **BGE**

Test Case 1: (equal)
Input: 0020d463
Decoded: BGE x1, x2, 0x8
Expected output : PC = PC + 0x8, Branch is taken

Test Case 2: (not equal)
Input: 0020d463
Decoded: BEQ x1, x2, 0x8

Expected output: PC = PC + 4, branch not taken

Test Case 3 (zero values)
Input: 0020d463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

5. **BLTU**

Test Case 1: (a<b)
Input: 0020e463
Decoded: BEQ x1, x2, 0x8
Expected output : PC = PC + 0x8, Branch is taken

Test Case 2: (a>b)
Input: 0020e463
Decoded: BEQ x1, x2, 0x8
Expected output: PC = PC + 4, branch not taken

Test Case 3 (a<b diff MSB)
Input: 0020e463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

Test Case 4 (a>b diff MSB)
Input: 0020e463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

Test Case 5 (a=b diff MSB)
Input: 0020e463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

6. **BGEU**

Test Case 1: (a<b)
Input: 0020f463
Decoded: BEQ x1, x2, 0x8
Expected output : PC = PC + 0x8, Branch is taken

Test Case 2: (a>b)
Input: 0020f463
Decoded: BEQ x1, x2, 0x8
Expected output: PC = PC + 4, branch not taken

Test Case 3 (a<b diff MSB)
Input: 0020f463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

Test Case 4 (a>b diff MSB)
Input: 0020f463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

Test Case 5 (a=b diff MSB)
Input: 0020f463
Decoded: BEQ x1, x2, 0x8
Expected output: branch taken, both variables are zero

R - TYPE

1. ADD

Test Case 1:

Input: `0x00B50633`

Decoded: `ADD a2, a0, a1`

Pre-state: `a0 = 0xA, a1 = 0x5`

Expected Output: `a2 = 0x0F`

Test Case 2(Overflow):

Input: 0x01C30533

Decoded: Add a0, t1, t3

Pre-State: t1 = 0x7FFFFFFF, t3 = 0x1

Expected Output: a0 = 0x80000000

Test Case 3(Negative):

Input: `0x00B50633`

Decoded: `ADD a2, a0, a1`

Pre-state: `a0 = 0xA, a1 = 0xFFFFFFFB`

Expected Output: `a2 = 0x05`

2. SUB

Test Case 1:

Input: `0x40B50633`

Decoded: `SUB x12, x10, x11`

Pre-state: `x10 = 0x15`, `x11 = 0x05`

Expected Output: `x12 = 0x10` (0x15 - 0x05)

Test Case 2(Overflow):

Input: 0x41C385B3

Decoded: sub a1, t2, t3

Pre-State: t2 = 0x80000000, t3 = 0x1

Expected Output: a0 = 0x7FFFFFFF

Test Case 3(Negative):

Input: `0x00B50633`

Decoded: `ADD a2, a0, a1`

Pre-state: `a0 = 0xA, a1 = 0xFFFFFFFB`

Expected Output: `a2 = 0x0F`

3. **OR**

Test Case 1:

Input: `0x00B50733`

Decoded: `OR x14, x10, x11`

Pre-state: `x10 = 0b1101`, `x11 = 0b1011`

Expected Output: `x14 = 0b1111` (0xD | 0xB = 0xF)

4. **AND**

Test Case 1:

Input: `0x00B578cc`

Decoded: `AND a6,a0,a1`

Pre-state: `a0= 0x5, a1 = 0xA`

Expected Output: `a6 = 0x0`

5. **SLL**

Test Case 1:

Input: `0x00B51633`

Decoded: sll a2, a0, a1

Pre-state: `a0 = 0xA, a1 = 0xFFFFFFFB`

Expected Output: `a2 = 0x140`

Test Case 2(negative):

Input: `0x00B51633`

Decoded: sll a2, a0, a1

Pre-state `a0 = 0xA, a1 = 0xFFFFFFFB`

Expected Output: `a2 = 0x50000000`

6. **SRL**

Test Case 1:

Input: `0x00B556B3`

Decoded: `srl a3, a0, a1`

Pre-state: `a0 = 0x5, a1= 0xA`

Expected Output: `a3 = 0`

Test Case 2(negative):

Input: `0x00B556B3`

Decoded: srl a2, a0, a1

Pre-state: `a0 = 0xA, a1 = 0xFFFFFFFB`

Expected Output: `a2 = 0x0`

7. **SRA**

Test Case 1:

Input: `0x40B55733`

Decoded: `sra a4, a0, a1`

Pre-state: `a0 = 0xA, a1 =0x5`

Expected Output: `a4 = 0`

Test Case 2(negative):

Input: `0x40B55733`

Decoded: sra a2, a0, a1

Pre-state: `a0 = 0xA, a1 = 0xFFFFFFFB`

Expected Output: `a2 = 0x0`

8. **SLT**

Test Case 1:

Input: `0x00B527B3`

Decoded: `slt a5, a0, a1`

Pre-state: `a0 = 0xA, a1 =0x5`

Expected Output: `a5 = 0`

Test Case 2(negative):

Input: `0x00B527B3`

Decoded: slt a2, a0, a1

Pre-state: `a0 = 0xA, a1 = 0xFFFFFFFB`

Expected Output: `a2 = 0x0`

9. SLTU

Test Case 1:

Input: `0x00A5B833`

Decoded: `sltu a6, a1, a0`

Pre-state: `a1 =0xA, a0 = 0x5`

Expected Output: `a6 = 1`

Test Case 2(negative):

Input: `0x00B527B3`

Decoded: sltu a2, a0, a1

Pre-state: `a0 = 0xA, a1 = 0xFFFFFFFB`

Expected Output: `a2 = 0x0`

10. XOR

Test Case 1:

Input:0x01CE4633

Decoded: xor a1, t2, t2

Prestate: t2=0x1

Expected output: a1= 0x0

Test Case 2:

Input: 0x00B54733

Decoded: xor a1, t1, t2

Prestate: t1 = 0xA, t2 = 0x5

Expected output: a1 = 0xF

J-Type

1. JAL

Test Case 1:

Input:0x02C000EF

Decoded: jal x1, target1

Prestate: x1= 0x0, imm = 0x2C
Expected output: PC = 0x2C

2. JALR

Test Case 1:

Input:0x00008067
Decoded: jalr x1, x1, 0
Prestate: x1= 0x0, ,x2 = 0x4 imm = 0x0
Expected output: PC = 0x4

Test Case 2:

Input:0x000101E7
Decoded: jalr x3, 0(x2)
Prestate: x3= 0x14, ,x2 = 0x34, imm = 0x0
Expected output: PC = 0x34

Test Case 3:

Input:0x004303E7
Decoded: jalr x7, 4(x6)
Prestate: x7= 0x24, ,x2 = 0x3C, imm = 0x4
Expected output: PC = 0x40