**Test Plan**

**R type Instruction**

Test case 1 : Basic functionality

ADDI x5, x0, 0x10 # x5 = 0x10

ADDI x6, x0, 0x5 # x6 = 0x5

ADD x7, x6, x5 # x7 = x6 + x5 (0x10 + 0x5)

SUB x7, x6, x5 # x7 = x6 - x5 (0x10 – 0x5)

SLL x7, x6, x5 # x7 = x6 << (x5 & 0x1F) (Logical shift left)

SLT x7, x6, x5 # x7 = (x6 < x5) ? 1 : 0 (Signed comparison)

SLTU x7, x6, x5 # x7 = (x6 < x5) ? 1 : 0 (Unsigned comparison)

XOR x7, x6, x5 # x7 = x6 ^ x5 (Bitwise XOR)

SRL x7, x6, x5 # x7 = x6 >> (x5 & 0x1F) (Logical shift right)

SRA x7, x6, x5 # x7 = x6 >> (x5 & 0x1F) (Arithmetic shift right)

OR x7, x6, x5 # x7 = x6 | x5 (Bitwise OR)

AND x7, x6, x5 # x7 = x6 & x5 (Bitwise AND)

| **Instruction** | **Expected Output (Hex)** | **Expected Output (Decimal)** |
| --- | --- | --- |
| ADD x7, x6, x5 | 0x15 | 21 |
| SUB x7, x6, x5 | 0xFFFFFFF5 | -11 |
| SLL x7, x6, x5 | 0x00050000 | 327680 |
| SLT x7, x6, x5 | 0x1 | 1 |
| SLTU x7, x6, x5 | 0x1 | 1 |
| XOR x7, x6, x5 | 0x15 | 21 |
| SRL x7, x6, x5 | 0x0 | 0 |
| SRA x7, x6, x5 | 0x0 | 0 |
| OR x7, x6, x5 | 0x15 | 21 |
| AND x7, x6, x5 | 0x0 | 0 |

Test Case 2 :

Overflow condition for ADD  
ADDI x5, x0, 0x7FFF\_FFFF # x5 = 2147483647 (Max positive value)

ADDI x6, x0, 0x1 # x6 = 1

ADD x7, x5, x6 # x7 = x5 + x6 (Expected: 0x80000000, which is -2147483648 in signed 2's complement)

Expected result : This results in an overflow, but no exception is raised.

Test Case 3 :

Overflow condition for SUB

ADDI x5, x0, 0x80000000 # x5 = 0x80000000 (-2147483648, Min signed 32-bit value)

ADDI x6, x0, 0x00000001 # x6 = 0x00000001 (1)

SUB x7, x5, x6 # x7 = x5 - x6 (Expected: 0x7FFFFFFF, which is +2147483647 in signed 2's complement)

Test case 4 :

Differentiating between SRA and SRL

ADDI x5, x0, 0xFFFFFFFF # x5 = 0xFFFFFFFF (-1 in two's complement)

ADDI x6, x0, 4 # x6 = 4 (shift amount)

# Perform Logical Shift Right (SRL)

SRL x7, x5, x6 # x7 = x5 >> x6 (Logical shift, fills with 0s)

# Perform Arithmetic Shift Right (SRA)

SRA x8, x5, x6 # x8 = x5 >> x6 (Arithmetic shift, fills with sign bit)

**Expected Output**

* SRL x7, x5, x6: **Logical Shift Right**
  + 0xFFFFFFFF >> 4 results in 0x0FFFFFFF (fills with 0s).
* SRA x8, x5, x6: **Arithmetic Shift Right**
  + 0xFFFFFFFF >> 4 results in 0xFFFFFFFF (fills with 1s, maintaining the sign bit).

Test case 5 :

Differentiate SLTU and SLT

# Initialize registers with values that show the difference between SLT and SLTU

ADDI x5, x0, -1 # x5 = 0xFFFFFFFF (-1 in signed, large positive in unsigned)

ADDI x6, x0, 1 # x6 = 0x00000001 (1 in both signed and unsigned)

SLT x7, x5, x6 # Signed comparison: (-1 < 1) -> x7 = 1

SLTU x8, x5, x6 # Unsigned comparison: (0xFFFFFFFF < 0x00000001) -> x8 = 0