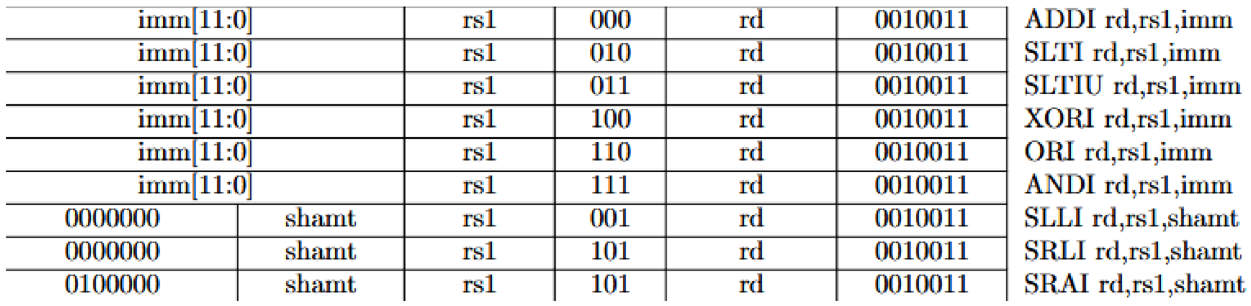
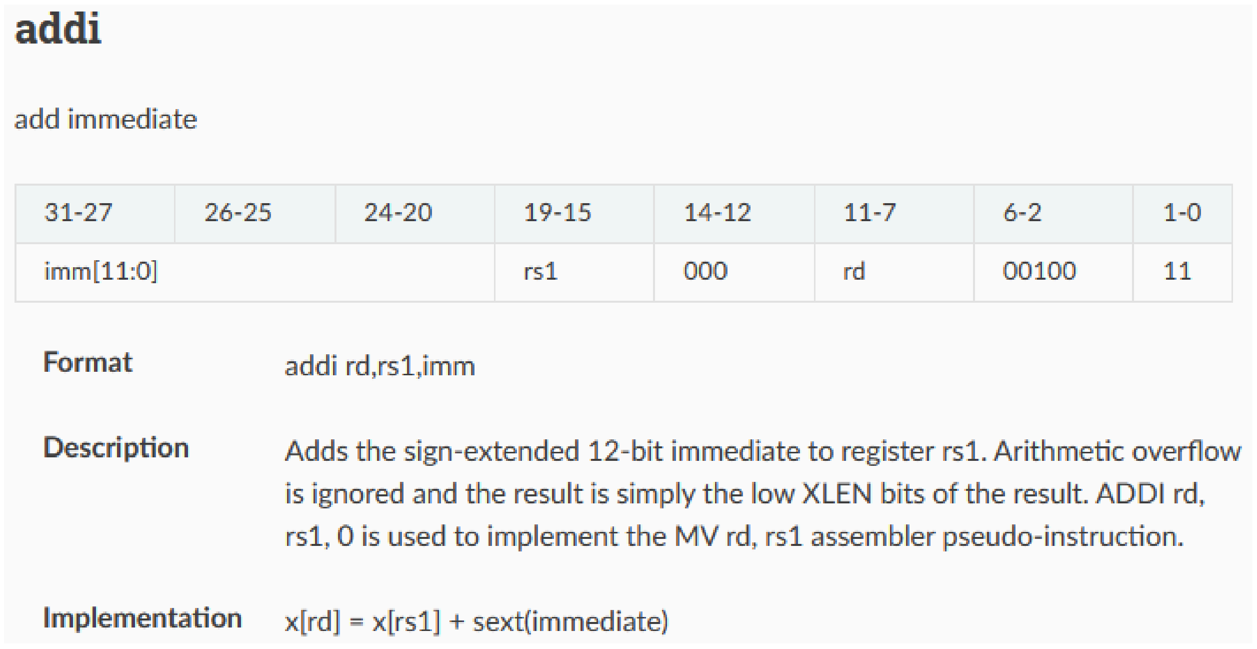
ECE586 Winter 2025
Team 07

Verification Test plan for Logical and Set Instructions



imm[11:0]		rs1	000	rd	0010011	ADDI rd,rs1,imm
imm[11:0]		rs1	010	rd	0010011	SLTI rd,rs1,imm
imm[11:0]		rs1	011	rd	0010011	SLTIU rd,rs1,imm
imm[11:0]		rs1	100	rd	0010011	XORI rd,rs1,imm
imm[11:0]		rs1	110	rd	0010011	ORI rd,rs1,imm
imm[11:0]		rs1	111	rd	0010011	ANDI rd,rs1,imm
0000000	shamt	rs1	001	rd	0010011	SLLI rd,rs1,shamt
0000000	shamt	rs1	101	rd	0010011	SRLI rd,rs1,shamt
0100000	shamt	rs1	101	rd	0010011	SRAI rd,rs1,shamt

1. **addi**



addi

add immediate

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	000	rd	00100	11

Format addi rd,rs1,imm

Description Adds the sign-extended 12-bit immediate to register rs1. Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result. ADDI rd, rs1, 0 is used to implement the MV rd, rs1 assembler pseudo-instruction.

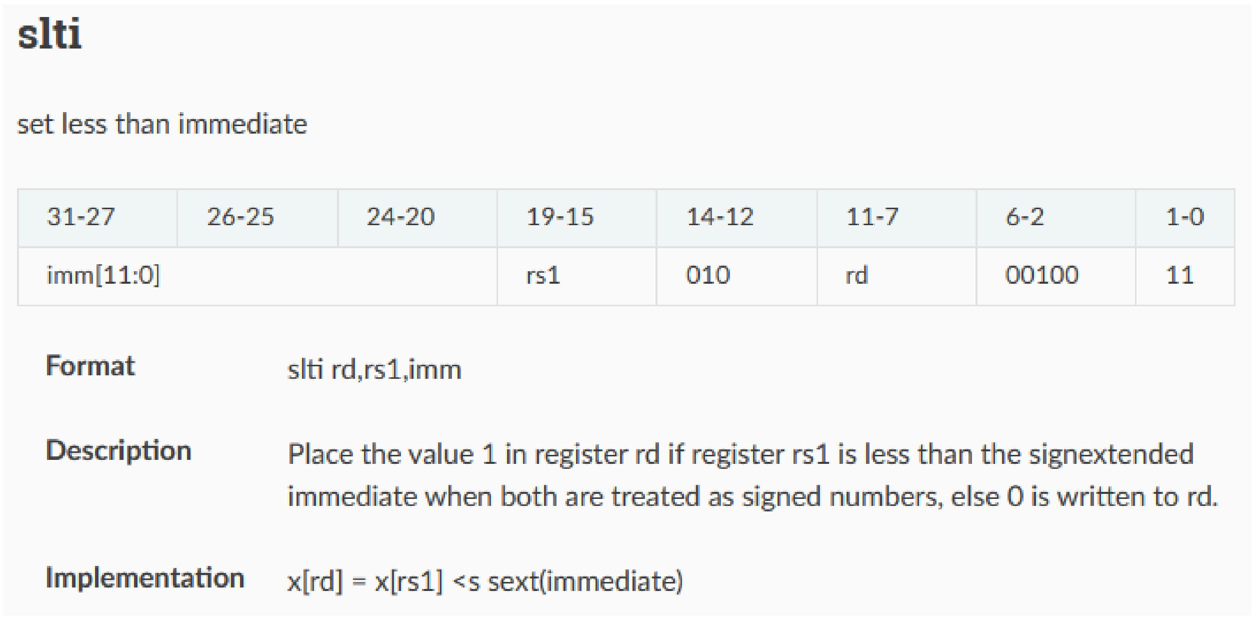
Implementation x[rd] = x[rs1] + sext(immediate)

- ☐ Basic testing
  - ☑ ~~Adding a negative number to a negative immediate~~
  - ☑ ~~Adding a positive number to a positive immediate~~
  - ☑ ~~Adding a negative number to a positive immediate~~
  - ☑ ~~Adding a positive number to a negative immediate~~
- ☐ ADDI rd, rs1, 0

- ☐ Is it overwriting rd =x0?

```
addi t0, t0, -10
addi t1, t0, -15
addi t1, t1, 30
addi t2, t1, 30

  0:    ff628293
  4:    ff128313
  8:    01e30313
  c:    01e30393
 10:    0000

Passed
```

2. slti



slti

set less than immediate

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	010	rd	00100	11

Format	slti rd,rs1,imm
Description	Place the value 1 in register rd if register rs1 is less than the signextended immediate when both are treated as signed numbers, else 0 is written to rd.
Implementation	x[rd] = x[rs1] <s sext(immediate)

- ☐ Basic Testing
  - ☑ ~~rs1 > imm~~
    - ☑ ~~rs1 is positive and imm is negative number~~
    - ☑ ~~rs1 is positive and imm is positive number~~
    - ☑ ~~rs1 is negative and imm is negative number~~
  - ☑ ~~rs1 < imm~~
    - ☑ ~~rs1 is negative and imm is positive number~~
    - ☑ ~~rs1 is positive and imm is positive number~~
    - ☑ ~~rs1 is negative and imm is negative number~~

ECE586 Winter 2025
Team 07

- [x] ~~rs1 = imm~~
    - [x] ~~rs1 is negative and imm is negative number~~
    - [x] ~~rs1 is positive and imm is positive number~~
    - [x] ~~rs1 is negative and imm is positive number~~
    - [ ] rs1 is positive and imm is negative number

```
addi t0, t0, 101
slti a0, t0, 100
slti a1, t0, -101
slti a2, t0, -100
addi t1, t1, -51
slti a3, t1, -101

Expected:
x5 (t0)  101     0x00000065    0b00000000000000000000000001100101
x6 (t1)  -51     0xffffffcd    0b11111111111111111111111111001101

   0:     06528293             addi     t0,t0,101
   4:     0642a513             slti     a0,t0,100
   8:     f9b2a593             slti     a1,t0,-101
   c:     f9c2a613             slti     a2,t0,-100
  10:     fcd30313             addi     t1,t1,-51
  14:     f9b32693             slti     a3,t1,-101
  18:     0000

Passed
```

```
addi t0, t0, 99
slti a0, t0, 200
addi t1, t1, -196
slti a1, t1, 11
slti a2, t1, -150

Expected:
x5 (t0)  99      0x00000063    0b00000000000000000000000001100011
x6 (t1)  -196    0xffffff3c    0b11111111111111111111111100111100
x10 (a0)         1         0x00000001    0b00000000000000000000000000000001
x11 (a1)         1         0x00000001    0b00000000000000000000000000000001
x12 (a2)         1         0x00000001    0b00000000000000000000000000000001

   0:     06328293
```

```
  4:    0c82a513
  8:    f3c30313
  c:    00b32593
 10:    f6a32613
 14:    0000
```

Passed

```
addi t0, t0, -99
slti a0, t0, -99
addi t1, t1, 99
slti a1, t0, 99
addi t2, t2, -196
slti a2, t2, 196
```

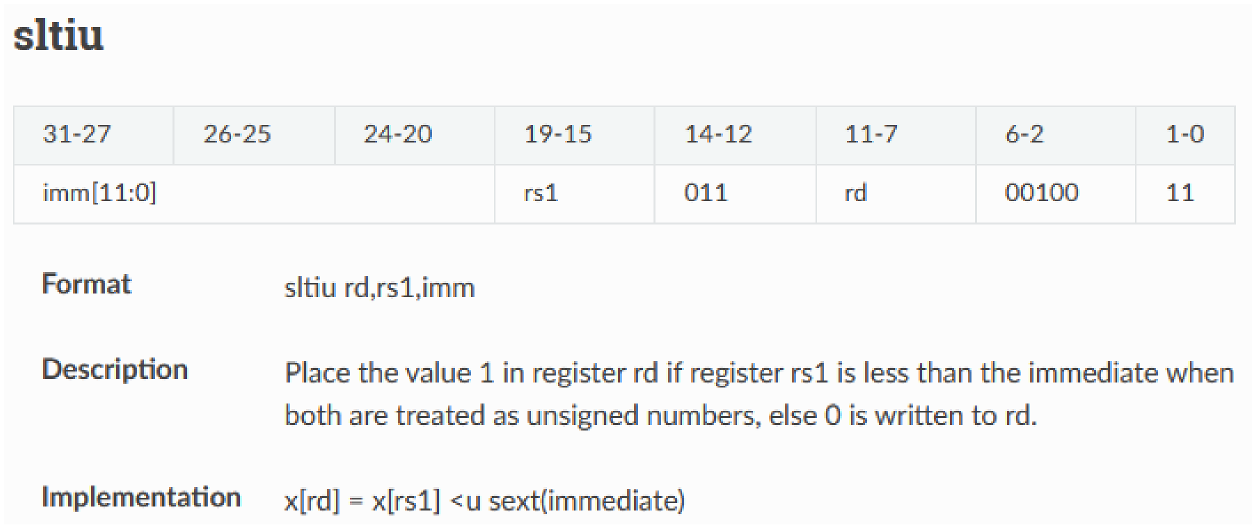
Expected:

```
x5 (t0)  -99    0xffffff9d    0b11111111111111111111111110011101
x6 (t1)  99     0x00000063    0b00000000000000000000000001100011
x7 (t2)  -196   0xffffff3c    0b11111111111111111111111100111100
x11 (a1)        1         0x00000001    0b00000000000000000000000000000001
x12 (a2)        1         0x00000001    0b00000000000000000000000000000001
```

```
  0:    f9d28293
  4:    f9d2a513
  8:    06330313
  c:    0632a593
 10:    f3c38393
 14:    0c43a613
 18:    000
```

Passed

3. sltiu



sltiu

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	011	rd	00100	11

Format	sltiu rd,rs1,imm
Description	Place the value 1 in register rd if register rs1 is less than the immediate when both are treated as unsigned numbers, else 0 is written to rd.
Implementation	x[rd] = x[rs1] <u sext(immediate)

- ☑ ~~Basic Tests~~
    - ☑ ~~rs1 > imm~~
        - ☑ ~~rs1 is negative and imm is negative number~~
        - ☑ ~~rs1 is positive and imm is positive number~~
        - ☑ ~~rs1 is negative and imm is positive number~~
        - ☑ ~~rs1 is positive and imm is negative number~~
    - ☐ rs1 < imm
        - ☐ rs1 is negative and imm is negative number
        - ☐ rs1 is positive and imm is positive number
        - ☐ rs1 is negative and imm is positive number
        - ☐ rs1 is positive and imm is negative number
    - ☐ rs1 = imm
        - ☐ rs1 is negative and imm is negative number
        - ☐ rs1 is positive and imm is positive number
        - ☐ rs1 is negative and imm is positive number
        - ☐ rs1 is positive and imm is negative number

```
addi t0, t0, 101
sltiu a0, t0, 100
sltiu a1, t0, -101
```

ECE586 Winter 2025
Team 07

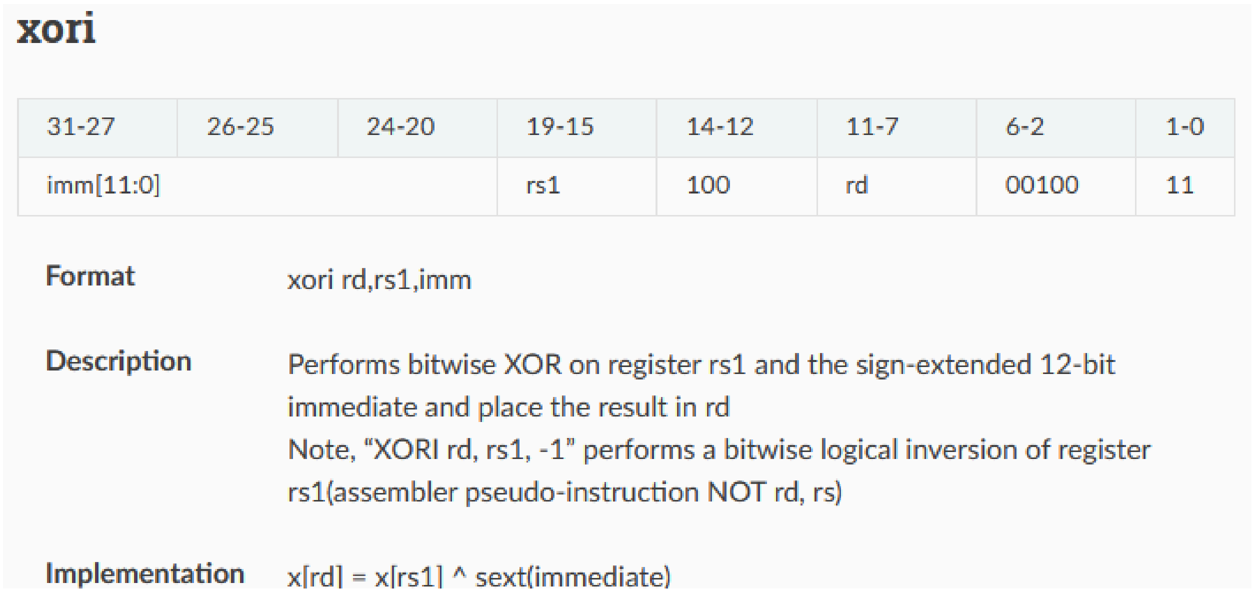
```
sltiu a2, t0, -100
addi t1, t1, -51
sltiu a3, t1, -101

Expected:
x5 (t0)  101     0x00000065   0b00000000000000000000000001100101
x6 (t1)  -51     0xffffffcd   0b11111111111111111111111111001101
x11 (a1)         1     0x00000001   0b00000000000000000000000000000001
x12 (a2)         1     0x00000001   0b00000000000000000000000000000001

   0:   06528293         addi    t0,t0,101
   4:   0642b513         sltiu   a0,t0,100
   8:   f9b2b593         sltiu   a1,t0,-101
   c:   f9c2b613         sltiu   a2,t0,-100
  10:   fcd30313         addi    t1,t1,-51
  14:   f9b33693         sltiu   a3,t1,-101
  18:   0000

Passed
```

```
addi t0, t0, 99
sltiu a0, t0, 200
addi t1, t1, -196
sltiu a1, t1, 11
sltiu a2, t1, -150

Expected:

x5 (t0)  99      0x00000063   0b00000000000000000000000001100011
x6 (t1)  -196    0xffffff3c   0b11111111111111111111111100111100
x10 (a0)         1     0x00000001   0b00000000000000000000000000000001
x12 (a2)         1     0x00000001   0b00000000000000000000000000000001

   0:   06328293         addi    t0,t0,99
   4:   0c82b513         sltiu   a0,t0,200
   8:   f3c30313         addi    t1,t1,-196
   c:   00b33593         sltiu   a1,t1,11
  10:   f6a33613         sltiu   a2,t1,-150
  14:   0000

Passed
```

```
addi t0, t0, -99
sltui a0, t0, -99
```

```
addi t1, t1, 99
sltiu a1, t0, 99
addi t2, t2, -196
sltiu a2, t2, 196

Expected:
x5 (t0)  -99     0xffffff9d     0b11111111111111111111111110011101
x6 (t1)  99      0x00000063     0b00000000000000000000000001100011
x7 (t2)  -196    0xffffff3c     0b11111111111111111111111100111100

   0:     f9d28293          addi     t0,t0,-99
   4:     f9d2b513          sltiu    a0,t0,-99
   8:     06330313          addi     t1,t1,99
   c:     0632b593          sltiu    a1,t0,99
  10:     f3c38393          addi     t2,t2,-196
  14:     0c43b613          sltiu    a2,t2,196
  18:     000

Passed
```

4. xori



xori

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	100	rd	00100	11

Format	xori rd,rs1,imm
Description	Performs bitwise XOR on register rs1 and the sign-extended 12-bit immediate and place the result in rd Note, “XORI rd, rs1, -1” performs a bitwise logical inversion of register rs1(assembler pseudo-instruction NOT rd, rs)
Implementation	x[rd] = x[rs1] ^ sext(immediate)

- ☐ Basic tests
    - ☐ Simple xor operation

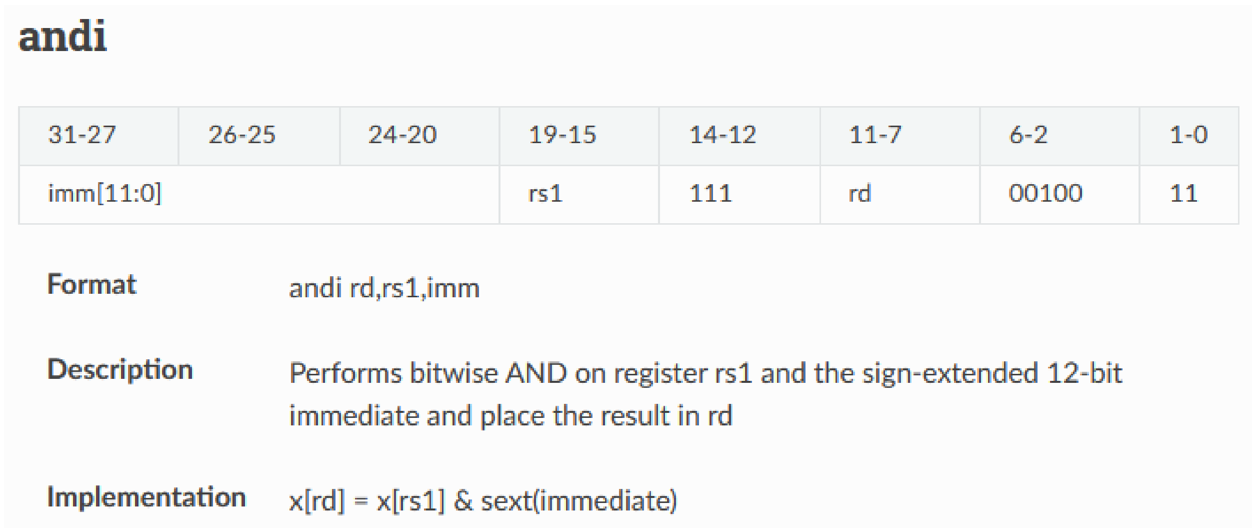
- ☐ immediate is sign extended 1
- ☐ Immediate is sign extended 0

☐ Logical inversion of rs1

- ☐ XORI rd, rs1, -1

```
addi t0, t0, -1
xori a0, t0, 2730
addi t1, t1, -2730
xori a1, t1, 2730
addi t2, t2, 1023
xori a2, t2, -1

Expected:
x5 (t0)  -1       0xffffffff 0b11111111111111111111111111111111
x6 (t1)  1365     0x00000555    0b00000000000000000000010101010101
x7 (t2)  1023     0x000003ff    0b00000000000000000000001111111111
x10 (a0)          -1366   0xfffffaaa    0b11111111111111111111101010101010
x12 (a2)          -1024   0xfffffc00    0b11111111111111111111110000000000

   0:   fff28293        addi    t0,t0,-1
   4:   5552c513                xori    a0,t0,1365
   8:   55530313                addi    t1,t1,1365
   c:   55534593                xori    a1,t1,1365
  10:   3ff38393                addi    t2,t2,1023
  14:   fff3c613        not     a2,t2
  18:   000

Passed
```

5. andi



andi

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	111	rd	00100	11

Format	andi rd,rs1,imm
Description	Performs bitwise AND on register rs1 and the sign-extended 12-bit immediate and place the result in rd
Implementation	x[rd] = x[rs1] & sext(immediate)

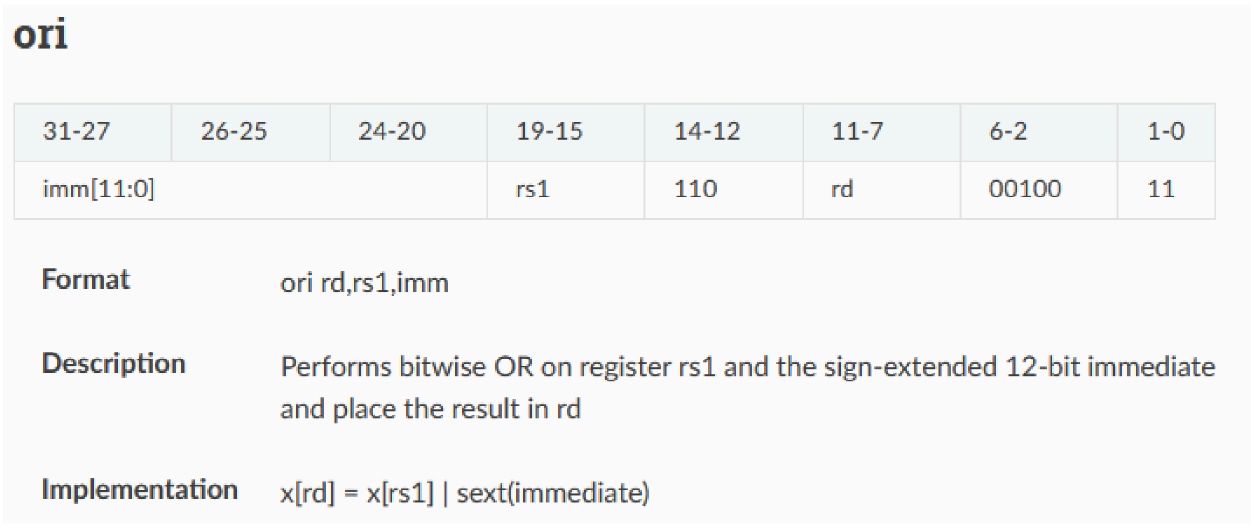
ECE586 Winter 2025
Team 07

- ☐ Basic tests
    - ☐ Simple and operation
        - ☐ immediate is (negative) sign extended 1
        - ☐ Immediate is (positive) sign extended 0
    - ☐ Immediate = 0x000
        - ☐ Sign extended 1
        - ☐ Sign extended 0
    - ☐ Immediate =0xFFF
        - ☐ Sign extended 1
        - ☐ Sign extended 0

```
 addi t0, t0, -1953
andi a0, t0, -1825
addi t1, t1, 1810
andi a1, t1, 1794
andi a2, t0, 1825
andi a3, t1, -1794

Expected:
        x5 (t0)  -1953   0xfffff85f      0b11111111111111111111100001011111
        x6 (t1)  1810    0x00000712      0b00000000000000000000011100010010
        x10 (a0)         -1953   0xfffff85f      0b11111111111111111111100001011111
        x11 (a1)         1794    0x00000702      0b00000000000000000000011100000010
        x12 (a2)         1       0x00000001      0b00000000000000000000000000000001
        x13 (a3)         18      0x00000012      0b00000000000000000000000000010010

  0:    85f28293                 addi    t0,t0,-1953
  4:    8df2f513                 andi    a0,t0,-1825
  8:    71230313                 addi    t1,t1,1810
  c:    70237593                 andi    a1,t1,1794
 10:    7212f613                 andi    a2,t0,1825
 14:    8fe37693                 andi    a3,t1,-1794
 18:    0000

Passed
```

```
addi t0, t0, -1
andi a0, t0, -1807
andi a1, t0, 1807

addi t1, t1, 0
andi a2, t1, -1807
andi a3, t1, 1807

Expected:
        x5 (t0)  -1      0xffffffff 0b11111111111111111111111111111111
        x10 (a0)         -1807   0xfffff8f1      0b11111111111111111111100011110001
        x11 (a1)        1807    0x0000070f    0b00000000000000000000011100001111


  0:    fff28293        addi    t0,t0,-1
  4:    8f12f513                andi    a0,t0,-1807
  8:    70f2f593                andi    a1,t0,1807
  c:    00030313                mv      t1,t1
 10:    8f137613                andi    a2,t1,-1807
 14:    70f37693                andi    a3,t1,1807
 18:    0000

Passed
```

6. ori



ori

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	110	rd	00100	11

Format ori rd,rs1,imm

Description Performs bitwise OR on register rs1 and the sign-extended 12-bit immediate and place the result in rd

Implementation x[rd] = x[rs1] | sext(immediate)

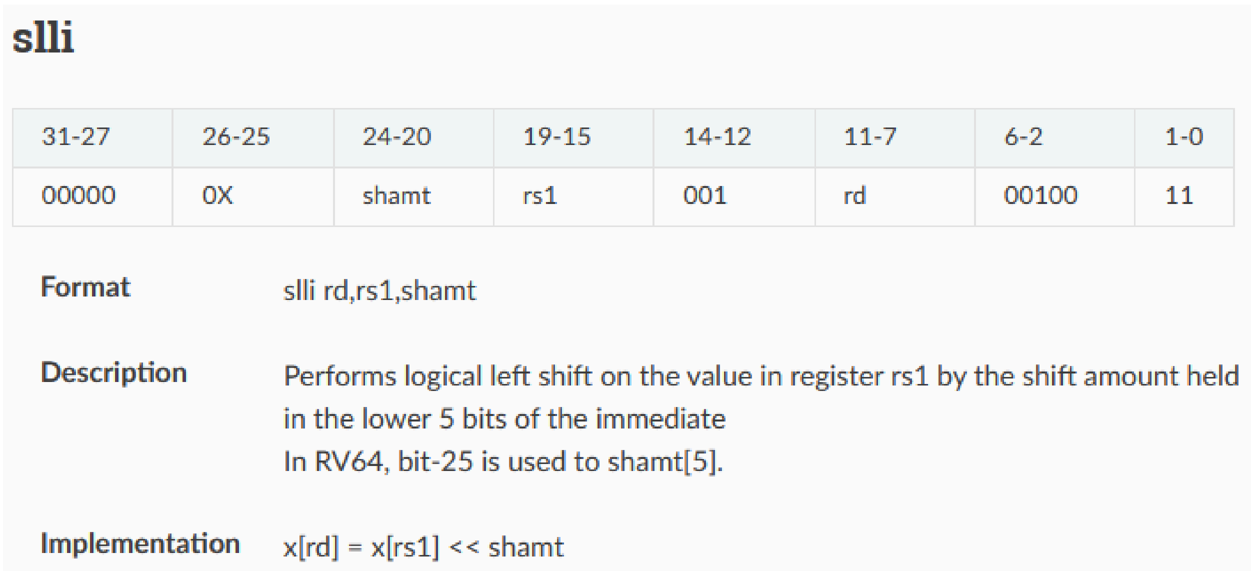
- ☐ Basic tests
    - ☐ Simple or operation
    - ☐ immediate is sign extended 1
    - ☐ Immediate is sign extended 0
    - ☐ Immediate = 0x0000
        - ☐ Sign extended 1
        - ☐ Sign extended 0
    - ☐ Immediate = 0xFFFF
        - ☐ Sign extended 1
        - ☐ Sign extended 0

ECE586 Winter 2025
Team 07

```
 addi t0, t0, -1953
ori a0, t0, -1825
addi t1, t1, 1810
ori a1, t1, 1794
ori a2, t0, 1825
ori a3, t1, -1794

Expected:
x5 (t0)  -1953  0xfffff85f    0b11111111111111111111100001011111
x6 (t1)  1810   0x00000712    0b00000000000000000000011100010010
x10 (a0)        -1825  0xfffff8df    0b11111111111111111111100011011111
]x11 (a1)       1810   0x00000712    0b00000000000000000000011100010010
]x12 (a2)       -129   0xffffff7f    0b11111111111111111111111101111111
]x13 (a3)       -2     0xfffffffe    0b11111111111111111111111111111110

   0:    85f28293            addi    t0,t0,-1953
   4:    8df2e513            ori     a0,t0,-1825
   8:    71230313            addi    t1,t1,1810
   c:    70236593            ori     a1,t1,1794
  10:    7212e613            ori     a2,t0,1825
  14:    8fe36693            ori     a3,t1,-1794
  18:    0000

Passed
```

```
addi t0, t0, -1
ori a0, t0, -1807
ori a1, t0, 1807

addi t1, t1, 0
ori a2, t1, -1807
ori a3, t1, 1807

Expected:
x5 (t0)  -1      0xffffffff 0b11111111111111111111111111111111
x10 (a0)         -1     0xffffffff 0b11111111111111111111111111111111
x11 (a1)         -1     0xffffffff 0b11111111111111111111111111111111
x12 (a2)         -1807  0xfffff8f1    0b11111111111111111111100011110001
x13 (a3)         1807   0x0000070f    0b00000000000000000000011100001111

   0:    fff28293        addi    t0,t0,-1
   4:    8f12e513                ori     a0,t0,-1807
   8:    70f2e593                ori     a1,t0,1807
   c:    00030313                mv      t1,t1
  10:    8f136613                ori     a2,t1,-1807
  14:    70f36693                ori     a3,t1,1807
  18:    000

Passed
```

7. slli



slli

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	0X	shamt	rs1	001	rd	00100	11

Format slli rd,rs1,shamt

Description Performs logical left shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate
In RV64, bit-25 is used to shamt[5].

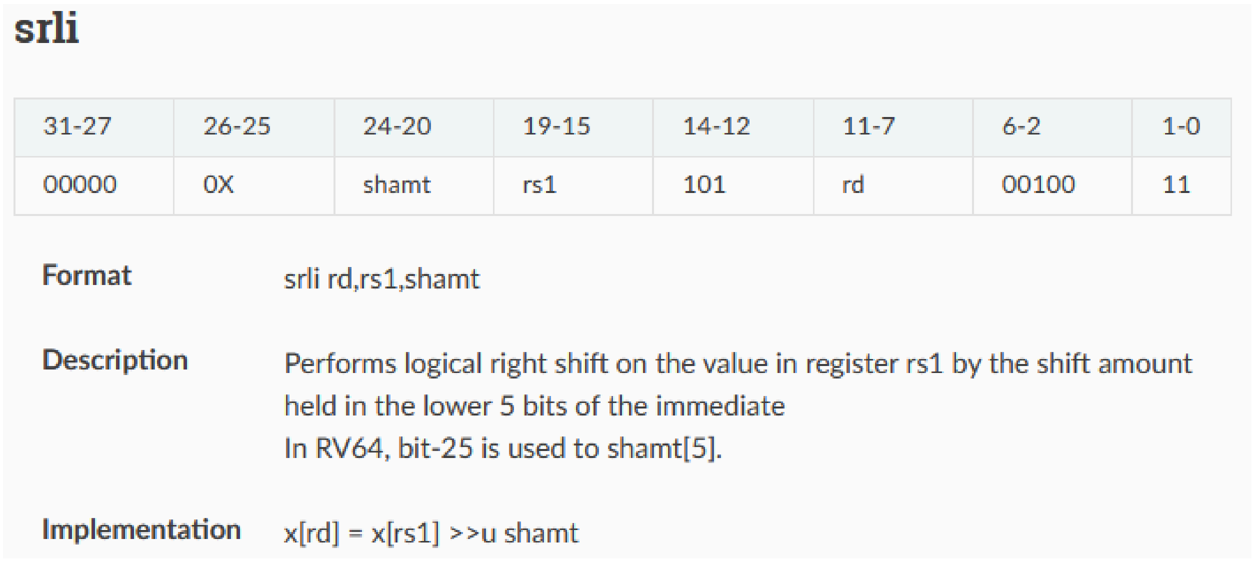
Implementation x[rd] = x[rs1] << shamt

- ☑ ~~Basic operations~~
    - ☑ ~~Normal random srl~~
    - ☑ ~~Sll 0 (shamt=0)~~
    - ☑ ~~Sll 32 (shamt=32)~~

```
addi t0, t0, -1
slli t1, t0, 0
slli t2, t0, 1
slli t3, t0, 31
slli t4, t0, 32

Expected:
x5 (t0)     -1              0xffffffff          0b11111111111111111111111111111111
x6 (t1)     -1              0xffffffff          0b11111111111111111111111111111111
x7 (t2)     -2              0xfffffffe          0b11111111111111111111111111111110
x28 (t3)   -2147483648      0x80000000    0b10000000000000000000000000000000
x29 (t4)  -1        0xffffffff      0b11111111111111111111111111111111

   0:     fff28293
   4:     00029313
   8:     00129393
   c:     01f29e13
  10:     02029e93
```

```
Passed
```

8. srli



srli

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	0X	shamt	rs1	101	rd	00100	11

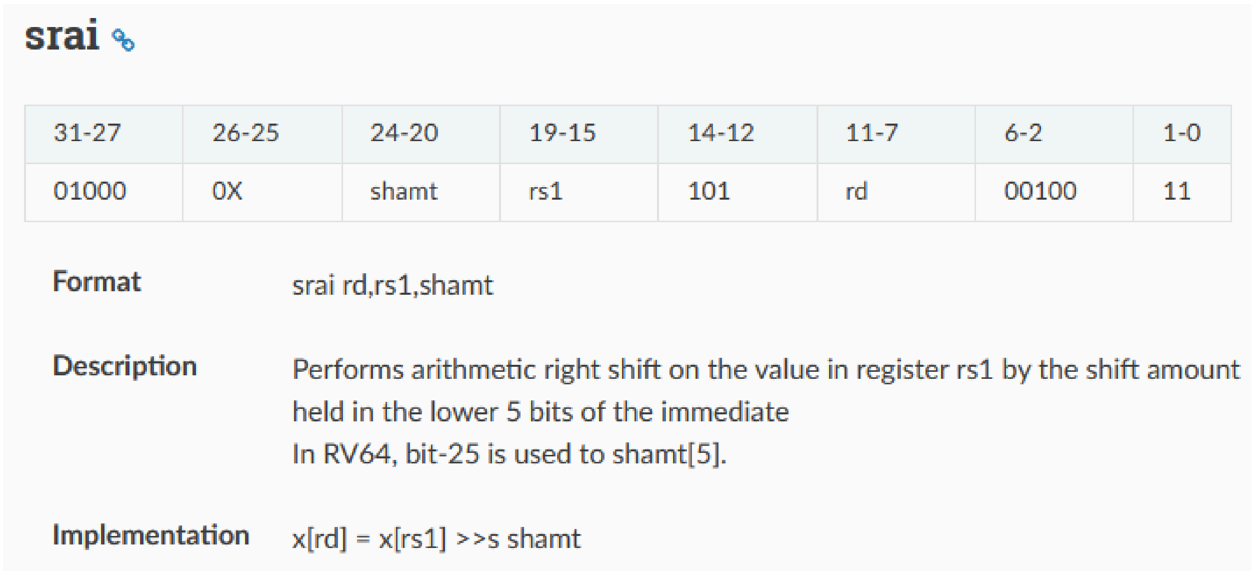
Format	srli rd,rs1,shamt
Description	Performs logical right shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate In RV64, bit-25 is used to shamt[5].
Implementation	x[rd] = x[rs1] >>u shamt

- ✓ ~~Basic operations~~
    - ✓ ~~Normal random srl~~
    - ✓ ~~Srl 0 (shamt=0)~~
    - ✓ ~~Srl 32 (shamt=32)~~

```
addi t0, t0, -1
srli t1, t0, 0
srli t2, t0, 1
srli t3, t0, 31
srli t4, t0, 32

Expected:
x5 (t0)  -1          0xffffffff 0b11111111111111111111111111111111
x6 (t1)  -1          0xffffffff 0b11111111111111111111111111111111
x7 (t2)  2147483647  0x7fffffff      0b01111111111111111111111111111111
x28 (t3)        1    0x00000001      0b00000000000000000000000000000001
x29 (t4)        -1   0xffffffff 0b11111111111111111111111111111111

  0:    fff28293
  4:    0002d313
  8:    0012d393
  c:    01f2de13
 10:    0202de93
```

```
Passed
```

9. srai



srai

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
01000	0X	shamt	rs1	101	rd	00100	11

Format	srai rd,rs1,shamt
Description	Performs arithmetic right shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate In RV64, bit-25 is used to shamt[5].
Implementation	x[rd] = x[rs1] >>s shamt

- [x] ~~Basic operations~~
    - [x] ~~Sra positive number~~
        - [x] ~~Normal random sra~~
        - [x] ~~Sra 0 (shamt=0)~~
        - [x] ~~Sra 32 (shamt=32)~~
    - [x] ~~Sra negative number~~
        - [x] ~~Normal random sra~~
        - [x] ~~Sra 0 (shamt=0)~~
        - [x] ~~Sra 32 (shamt=32)~~

```
addi t0, t0, -1
srai t1, t0, 0
srai t2, t0, 1
srai t3, t0, 31
srai t4, t0, 32

Expected:
x5 (t0)  -1        0xffffffff 0b11111111111111111111111111111111
x6 (t1)  -1        0xffffffff 0b11111111111111111111111111111111
x7 (t2)  -1        0xffffffff 0b11111111111111111111111111111111
x28 (t3)           -1        0xffffffff 0b11111111111111111111111111111111
```

```
x29 (t4)        -1      0xffffffff 0b11111111111111111111111111111111

   0:     fff28293
   4:     4002d313
   8:     4012d393
   c:     41f2de13
  10:     4202de93
  14:   0000
```

Passed

```
addi t0, t0, 2047
srai t1, t0, 0
srai t2, t0, 1
srai t3, t0, 7
srai t4, t0, 31
srai t5, t0, 32
```

Expected:

```
x5 (t0)  2047   0x000007ff    0b00000000000000000000011111111111
x6 (t1)  2047   0x000007ff    0b00000000000000000000011111111111
x7 (t2)  1023   0x000003ff    0b00000000000000000000001111111111
x28 (t3)        15      0x0000000f    0b00000000000000000000000000001111
x29 (t4)        0       0x00000000    0b00000000000000000000000000000000
x30 (t5)        2047    0x000007ff    0b00000000000000000000011111111111

   0:     7ff28293
   4:     4002d313
   8:     4012d393
   c:     4072de13
  10:     41f2de93
  14:     4202df13
  18:     0000
```

Passed

ECE586 Winter 2025
Team 07

Arithmetic Test Cases:

ADD

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

```
 #--------------------------------------
# Helper: Load immediate macro (manually expanded)
# --------------------------------------
# Load -5 into x3: (since -5 fits in 12-bit immediate)
   addi x3, x0, -5        # x3 = -5

# Load 10 into x4:
   addi x4, x0, 10        # x4 = 10

# Load 7 into x5:
   addi x5, x0, 7         # x5 = 7

# Load -7 into x6:
   addi x6, x0, -7        # x6 = -7

# Load 25 into x8:
   addi x8, x0, 25        # x8 = 25

# --------------------------------------
# Test 1: Both operands are zero (x1=0, x2=0)
# Result -> x10
# --------------------------------------
   addi x1, x0, 0         # x1 = 0
   addi x2, x0, 0         # x2 = 0
   add x10, x1, x2        # x10 = 0 + 0 = 0

# --------------------------------------
# Test 2: Signed addition (-5 + 10)
# Result -> x11
# --------------------------------------
   add x11, x3, x4        # x11 = -5 + 10 = 5

# --------------------------------------
# Test 3: Adding a register with its negative (7 + -7)
# Result -> x12
# --------------------------------------
```

```
    add x12, x5, x6         # x12 = 7 + (-7) = 0

# --------------------------------------
# Test 4: Adding zero with a non-zero register (0 + 25)
# Result -> x13
# --------------------------------------
    addi x7, x0, 0          # x7 = 0
    add x13, x7, x8         # x13 = 0 + 25 = 25

# --------------------------------------
# End of tests: Results stored in x10-x13
# x10 = 0, x11 = 5, x12 = 0, x13 = 25
# --------------------------------------
```

```
   0:    ffb00193
   4:    00a00213
   8:    00700293
   c:    ff900313
  10:    01900413
  14:    00000093
  18:    00000113
  1c:    00208533
  20:    004185b3
  24:    00628633
  28:    00000393
  2c:    008386b3
```

SUB

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

SLL

- Test with random values and shift amounts within the range of 0 to 31
- Input value is a positive number, and the shift amount is 1
- Input value is zero and the shift amount is zero
- Test with some input value and shift amount equal to 31
- Input value is all ones, and the shift amount is zero

- Tests to check whether the lower 5 bits are getting extracted from the rs2

```
# ---------------------------------------
# Load values using ADDI & LUI
# ---------------------------------------

# Random value: 0x12345678
lui x5, 0x12345          # x5 = 0x12345000
addi x5, x5, 0x678       # x5 = 0x12345678 (legal immediate: 0x678 ≤ 2047 decimal)

# Positive value: 0x15 (21 in decimal)
addi x6, x0, 21

# Zero value
addi x7, x0, 0

# Input with all ones: 0xFFFFFFFF (use LUI + ADDI with -1)
lui x8, 0xFFFFF          # x8 = 0xFFFFF000
addi x8, x8, -1          # x8 = 0xFFFFFFFF

# Random shift amount: 0x1F (31 decimal)
addi x9, x0, 31

# ---------------------------------------
# Test 1: Random value (0x12345678) shifted by 4
# Expected: x10 = 0x23456780
# ---------------------------------------
addi x11, x0, 4          # Shift amount = 4
sll x10, x5, x11

# ---------------------------------------
# Test 2: Positive value (21) shifted by 1
# Expected: x12 = 42
# ---------------------------------------
addi x11, x0, 1
sll x12, x6, x11

# ---------------------------------------
# Test 3: Zero value shifted by zero
# Expected: x13 = 0
# ---------------------------------------
addi x11, x0, 0
sll x13, x7, x11

# ---------------------------------------
# Test 4: Random value shifted by 31
# Expected: x14 = 0x80000000
# ---------------------------------------
```

```
sll x14, x5, x9

# ---------------------------------------
# Test 5: All ones (0xFFFFFFFF) shifted by zero
# Expected: x15 = 0xFFFFFFFF
# ---------------------------------------
addi x11, x0, 0
sll x15, x8, x11

# ---------------------------------------
# Test 6: Check lower 5 bits extraction from rs2
# Shift amount in x17: 0xFF (binary: 11111111) → lower 5 bits = 31
# Expected: x16 = x6 << 31 (verify only lower 5 bits used)
# ---------------------------------------
addi x17, x0, -1        # x17 = 0xFFFFFFFF (rs2 with all ones)
sll x16, x6, x17

# ---------------------------------------
# End: Results stored in x10 - x16
# ---------------------------------------
```

```
0:  123452b7
4:  67828293
8:  01500313
c:  00000393
10: fffff437
14: fff40413
18: 01f00493
1c: 00400593
20: 00b29533
24: 00100593
28: 00b31633
2c: 00000593
30: 00b396b3
34: 00929733
38: 00000593
3c: 00b417b3
40: fff00893
44: 01131833
```

SLT

- Tested with basic values.
- Following are the updated test cases with different values (results unchanged):

- rs1 = +45 and rs2 = -10. Sets rd = 0
- rs1 = -50 and rs2 = -20. Sets rd = 1
- rs1 = -60 and rs2 = +25. Sets rd = 1
- rs1 = +30 and rs2 = +15. Sets rd = 0
- rs1 = -5 and rs2 = 0. Sets rd = 1
- rs1 = 0 and rs2 = 0. Sets rd = 0

```
# Test case 1: rs1 = +45, rs2 = -10, sets rd = 0
addi x3, x0, 45     # Load immediate +45 into x3 (rs1)
addi x4, x0, -10    # Load immediate -10 into x4 (rs2)
slt  x5, x3, x4     # slt sets x5 (rd) to 0 (x3 is not less than x4)

# Test case 2: rs1 = -50, rs2 = -20, sets rd = 1
addi x6, x0, -50    # Load immediate -50 into x6 (rs1)
addi x7, x0, -20    # Load immediate -20 into x7 (rs2)
slt  x8, x6, x7     # slt sets x8 (rd) to 1 (x6 is less than x7)

# Test case 3: rs1 = -60, rs2 = +25, sets rd = 1
addi x9, x0, -60    # Load immediate -60 into x9 (rs1)
addi x10, x0, 25    # Load immediate +25 into x10 (rs2)
slt  x11, x9, x10   # slt sets x11 (rd) to 1 (x9 is less than x10)

# Test case 4: rs1 = +30, rs2 = +15, sets rd = 0
addi x12, x0, 30    # Load immediate +30 into x12 (rs1)
addi x13, x0, 15    # Load immediate +15 into x13 (rs2)
slt  x14, x12, x13  # slt sets x14 (rd) to 0 (x12 is not less than x13)

# Test case 5: rs1 = -5, rs2 = 0, sets rd = 1
addi x15, x0, -5    # Load immediate -5 into x15 (rs1)
addi x16, x0, 0     # Load immediate 0 into x16 (rs2)
slt  x17, x15, x16  # slt sets x17 (rd) to 1 (x15 is less than x16)

# Test case 6: rs1 = 0, rs2 = 0, sets rd = 0
addi x18, x0, 0     # Load immediate 0 into x18 (rs1)
addi x19, x0, 0     # Load immediate 0 into x19 (rs2)
slt  x20, x18, x19  # slt sets x20 (rd) to 0 (x18 is not less than x19)
```

```
0:  02d00193
4:  ff600213
8:  0041a2b3
c:  fce00313
10: fec00393
14: 00732433
18: fc400493
1c: 01900513
```

```
20: 00a4a5b3
24: 01e00613
28: 00f00693
2c: 00d62733
30: ffb00793
34: 00000813
38: 0107a8b3
3c: 00000913
40: 00000993
44: 01392a33
```

SLTU

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

XOR

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

SRL

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

SRA

- Tested with basic shifting
- Tested with MSB=1 value
- Tested for the following updated values (results unchanged):
    - rs1 = -64 and rs2 = 3
    - rs1 = +56 and rs2 = 2
    - rs1 = -48 and rs2 = 5. Lower 5 bits are taken and treated as a positive decimal shift amount

    - rs1 = +40 and rs2 = 6. Lower 5 bits are taken and treated as a positive decimal shift amount
    - rs1 = +32 and rs2 = 1. Lower 5 bits are taken and treated as a positive decimal shift amount

OR

- Test for one register is ‘b0
- Test for one register is ‘b1
- One register ‘ba and other is ‘b4

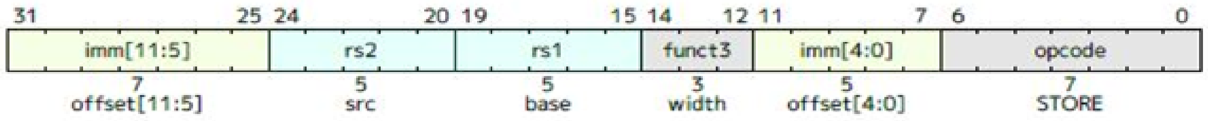
AND

- One register all zero and other is all 1s
- One register all 1s and other is all zero

ECE586 Winter 2025
Team 07

Verification plan for Loads and Stores, and Upper immediate instructions:

Stores:



1) Store byte(sb):

Format: sb rs2, offset$_{12}$(rs1)

Basic Testing:

1) **Byte extraction:** Only the least significant byte of the source register should be stored.
2) **Address Calculation:** Ensure the correct memory address is computed.
3) **Memory Update:** Only update the corresponding byte in memory without modifying other bytes in the word.

2) Store half(sh):

Format: sh rs2, offset$_{12}$(rs1)

Basic Testing:

1) **Byte extraction:** Only the least significant 2 bytes of the source register should be stored.
2) **Address Calculation:** Ensure the correct memory address is computed.
3) **Memory Update:** Ensure that only the intended bytes are modified, and other bytes remain unchanged.
4) In **little-endian** mode (default in RISC-V), the lower byte is stored first at the computed address, followed by the higher byte.

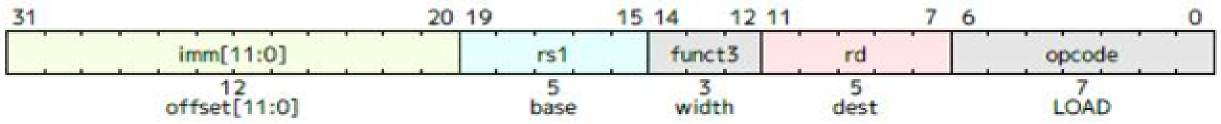
3) Store word(sw):

Format: sw rs2, offset$_{12}$(rs1)

Basic Testing:

1) **Memory Update:** Ensure that the bytes are modified in their corresponding address.

Loads:



1) Load byte(lb):

Format: lb rd, offset$_{12}$(rs1)

Basic Tesing:

- Load only a byte into the destination register from the correct address.
- Check for sign extension.

2) Load half(lh):

Format: lh rd, offset$_{12}$(rs1)

Basic Tesing:

- Load consecutive 2 bytes from the given address.
- Check for sign extension and proper concatenation of bytes.

3) Load word(lw):

Format: lw rd, offset$_{12}$(rs1)

Basic Tesing:

- Load four bytes from the consecutive memory address.
- Check for proper concatenation of bytes.

4) Load byte Unsigned(lbu):

Format: lbu rd, offset$_{12}$(rs1)

Basic Tesing:

- Load only a byte into the destination register from the correct address.
- Check for extension. The MSB bytes should be 0.

5) Load half(lhu):

Format: lhu rd, offset$_{12}$(rs1)

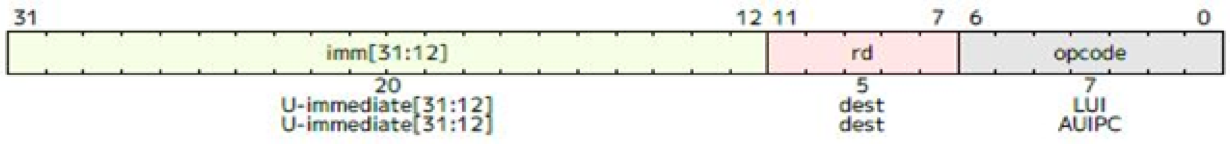
Basic Tesing:

- Load consecutive 2 bytes from the given address.
- Check for sign extension and proper concatenation of bytes. MSB bytes should be 0.

Assemby Code for Stores and Loads:

```
    .section .text
         .align 2
         .globl  _start

_start:
         addi sp, sp, -16
         sw s0, 12(sp)
         addi s0, sp, 16
         li a1, 0x00000078
         sb a1, 0(sp)
         li a2, 0x000000a8
         sb a2, 1(sp)
         li a3, 0x00005678
         sh a3, 2(sp)
         li a4, 0x0000a678
         sh a4, 4(sp)
         li a5, 0x12345678
         sw a5, 6(sp)
         lb s1, 0(sp)
         lb s2, 1(sp)
         lbu s3, 0(sp)
         lbu s4, 1(sp)
         lh s5, 2(sp)
         lh s6, 4(sp)
         lhu s7, 2(sp)
         lhu s8, 4(sp)
         lw s9,6(sp)
         lw s0, 12(sp)
         addi sp, sp, 16
.word 0
```

Upper Immediates:



1) Load Upper Immediate(lui):

Format: lui rd, imm

Basic Testing:

- Check for the storing of immediate value in the upper 20 bits of destination register.
- The LSB 12 bits of destination register should be 0.

2) Add upper immediate PC(auipc):

Format: auipc rd, imm

Basic Testing:

- The immediate value should be added to the current pc value and stored in destination register.

Assembly code for Upper Immediates:

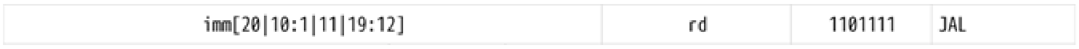
```
_start:
        lui t1, 0x12345
        auipc t2, 0x12345
.word 0
```

ECE586 Winter 2025
Team 07

Verification Plan for Jump and Branch Instructions:

1. **JAL (Jump and link):**

Opcode Map:



imm[20|10:1|11|19:12]	rd	1101111	JAL

Testcases:

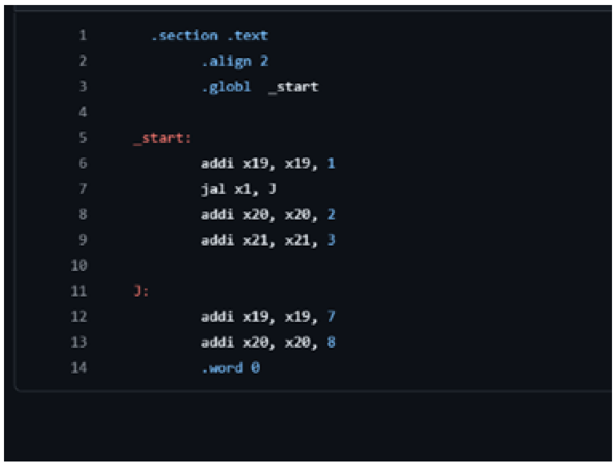
a. Forward Immediate Jump:
Verify that when using a forward immediate label, the JAL instruction transfers control to the correct forward address.

b. Backward Immediate Jump:
Ensure that a backward immediate label causes the JAL instruction to branch backward to the correct target.

c. Return Address Update:
Confirm that the JAL instruction correctly saves the return address (the address immediately following the jump) in the appropriate register.

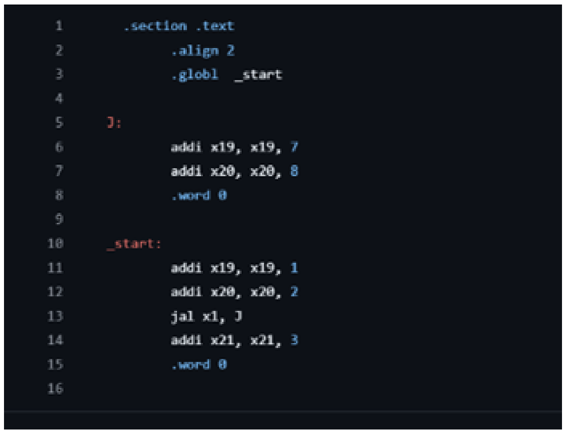
Assembly Code:

Simple forward jump:



```
    .section .text
        .align 2
        .globl  _start

  _start:
        addi x19, x19, 1
        jal x1, J
        addi x20, x20, 2
        addi x21, x21, 3

  J:
        addi x19, x19, 7
        addi x20, x20, 8
        .word 0
```

Backward jump:



```
  .section .text
        .align 2
        .globl  _start

J:
        addi x19, x19, 7
        addi x20, x20, 8
        .word 0

_start:
        addi x19, x19, 1
        addi x20, x20, 2
        jal x1, J
        addi x21, x21, 3
        .word 0
```

2. <u>JALR (Jump and link register)</u>:

Opcode Map:

imm[11:0]	rs1	000	rd	1100111	JALR

Testcases:

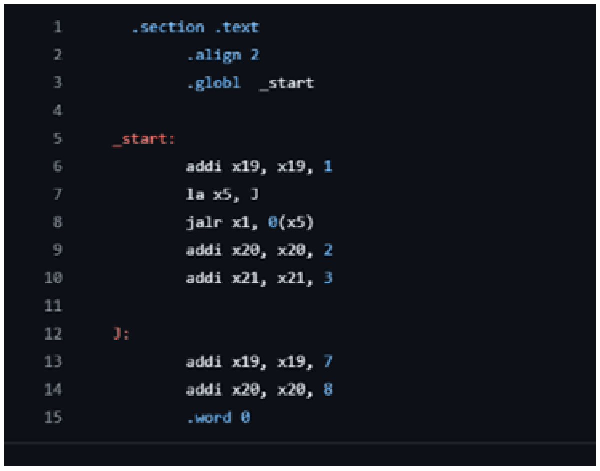
a. Forward Immediate Jump:
Confirm that with a forward immediate label, the JALR instruction computes and jumps to the correct target address.

b. Backward Immediate Jump:
Ensure that using a backward immediate label correctly results in a jump to the appropriate backward target address.

c. Return Address Update:
Validate that the JALR instruction properly stores the address of the following instruction into the return address register.

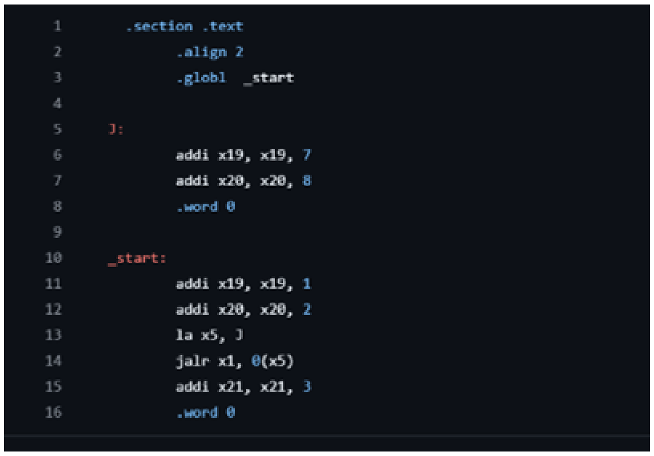
Assembly Code:

Simple forward jump:



```
    .section .text
        .align 2
        .globl  _start

_start:
        addi x19, x19, 1
        la x5, J
        jalr x1, 0(x5)
        addi x20, x20, 2
        addi x21, x21, 3

J:
        addi x19, x19, 7
        addi x20, x20, 8
        .word 0
```

Backward jump:



```
    .section .text
        .align 2
        .globl  _start

J:
        addi x19, x19, 7
        addi x20, x20, 8
        .word 0

_start:
        addi x19, x19, 1
        addi x20, x20, 2
        la x5, J
        jalr x1, 0(x5)
        addi x21, x21, 3
        .word 0
```

3. <u>BEQ (Branch if equal)</u>:

Opcode Map:

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

Testcases:

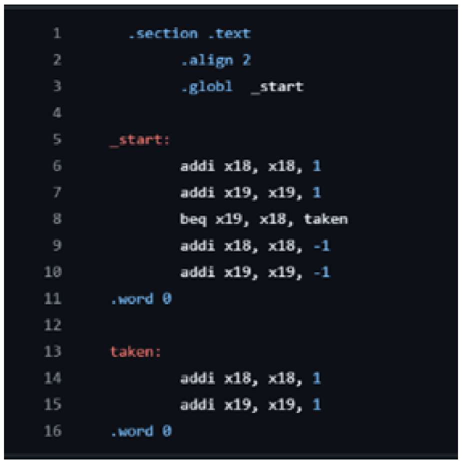
a. Simple Branching:
Verify that when the BEQ condition is satisfied, the instruction correctly transfers control to the designated target.

b. Multiple Branching:
Confirm that a sequence of BEQ instructions (Forward and Backward branches) branches correctly when the equality condition holds for multiple tests.

c. Branch not taken:
Validate that the PC falls through for the branch not taken condition.

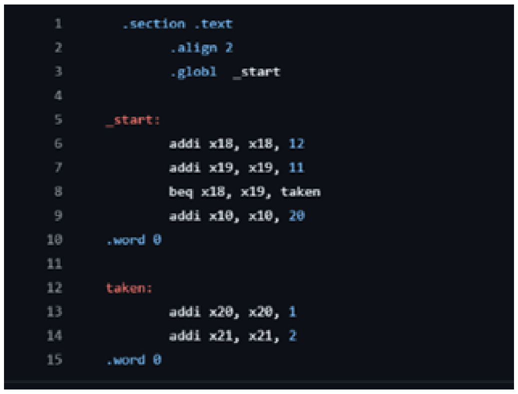
Assembly Code:

Forward branch (BEQ):



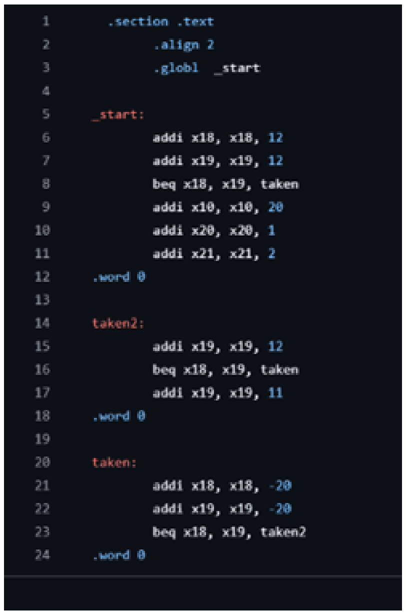
```
  .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, 1
        addi x19, x19, 1
        beq x19, x18, taken
        addi x18, x18, -1
        addi x19, x19, -1
.word 0

taken:
        addi x18, x18, 1
        addi x19, x19, 1
.word 0
```

Branch not taken (Fall through):



```
  .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, 12
        addi x19, x19, 11
        beq x18, x19, taken
        addi x10, x10, 20
.word 0

taken:
        addi x20, x20, 1
        addi x21, x21, 2
.word 0
```

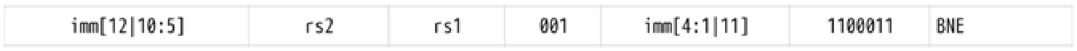
Multiple Branching (forward and backward with +ve and -ve values):



```
  .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, 12
        addi x19, x19, 12
        beq x18, x19, taken
        addi x10, x10, 20
        addi x20, x20, 1
        addi x21, x21, 2
.word 0

taken2:
        addi x19, x19, 12
        beq x18, x19, taken
        addi x19, x19, 11
.word 0

taken:
        addi x18, x18, -20
        addi x19, x19, -20
        beq x18, x19, taken2
.word 0
```

4. BNE (Branch if not equal):

Opcode Map:



imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE

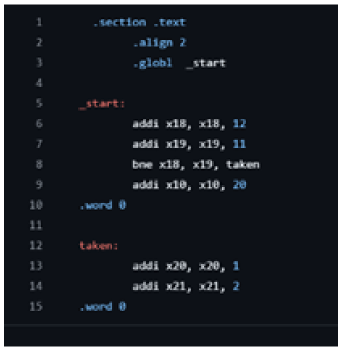
Testcases:

a. Simple Branching:
 Verify that when the BNE condition is satisfied, the instruction correctly transfers control to the designated target.

b. Multiple Branching:
 Confirm that a sequence of BNE instructions (Forward and Backward branches) branches correctly when the equality condition holds for multiple tests.

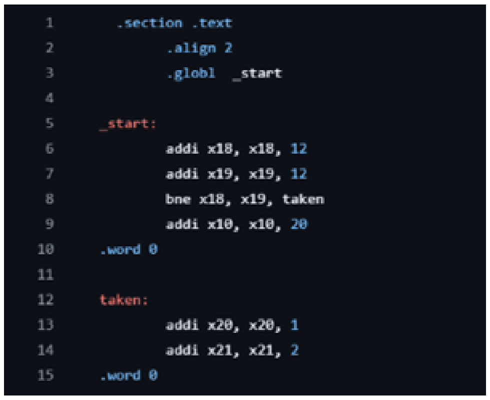
c. Branch not taken:
 Validate that the PC falls through for the branch not taken condition.

Assembly Code:
Forward branch (BNE):



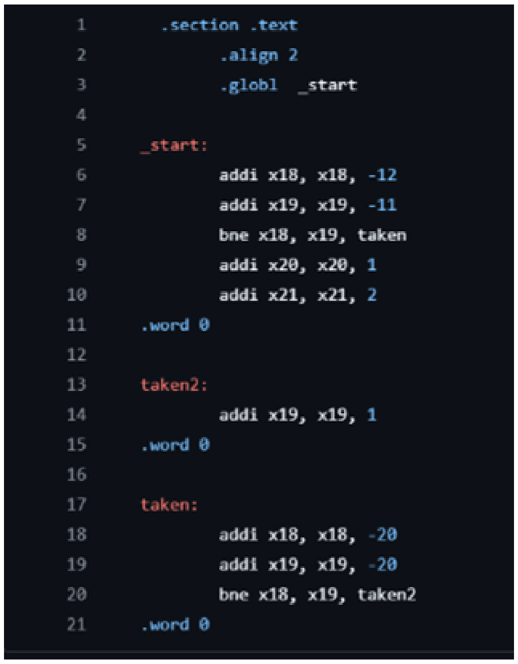
```
    .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, 12
        addi x19, x19, 11
        bne x18, x19, taken
        addi x10, x10, 20
.word 0

taken:
        addi x20, x20, 1
        addi x21, x21, 2
.word 0
```

Branch not taken (Fall through):



```
    .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, 12
        addi x19, x19, 12
        bne x18, x19, taken
        addi x10, x10, 20
.word 0

taken:
        addi x20, x20, 1
        addi x21, x21, 2
.word 0
```

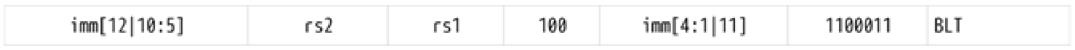
Multiple Branching (forward and backward with +ve and -ve values):



```
    .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, -12
        addi x19, x19, -11
        bne x18, x19, taken
        addi x20, x20, 1
        addi x21, x21, 2
.word 0

taken2:
        addi x19, x19, 1
.word 0

taken:
        addi x18, x18, -20
        addi x19, x19, -20
        bne x18, x19, taken2
.word 0
```

5. BLT (Branch if less than):

Opcode Map:



imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT

Testcases:

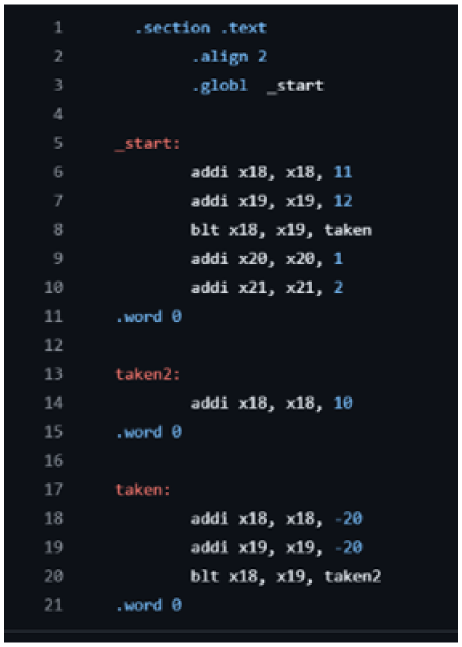
a. Bidirectional Branching:
Confirm that the BLT instruction enables the PC to branch both forwards and backwards as needed.

b. Equal Operands:
Verify that no branch occurs when both operands are identical.

c. Second Operand Greater:
Ensure that when the first operand is less than the second, the branch is correctly taken.

d. First Operand Greater:
Confirm that when the first operand is greater than the second, the branch is not executed.

Assembly Code:
Covers less than (branch taken), +ve and -ve values and forward and backward branching:



```
  .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, 11
        addi x19, x19, 12
        blt x18, x19, taken
        addi x20, x20, 1
        addi x21, x21, 2
.word 0

taken2:
        addi x18, x18, 10
.word 0

taken:
        addi x18, x18, -20
        addi x19, x19, -20
        blt x18, x19, taken2
.word 0
```

6. <u>BGE (Branch if greater than or equal to)</u>:

Opcode Map:

imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE

Testcases:

a. Bidirectional Branching:
Verify that the BGE instruction supports branching both forward and backward.

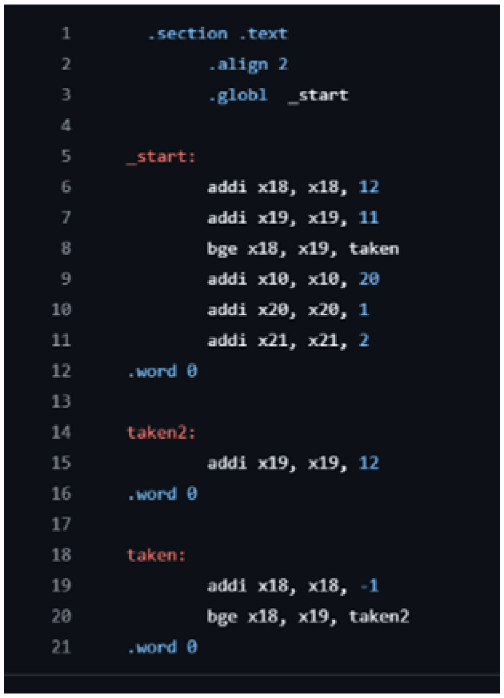
b. Equal Operands:
Confirm that when both operands are identical, the branch is taken, as the condition (greater or equal) holds.

c. Second Operand Greater:
Ensure that if the second operand is larger than the first, the branch is not executed.

d. First Operand Greater:
Validate that when the first operand exceeds the second, the branch is correctly taken.

Assembly Code:

Covers greater than and equal to (branch taken), +ve and -ve values and forward and backward branching:



```
  .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, 12
        addi x19, x19, 11
        bge x18, x19, taken
        addi x10, x10, 20
        addi x20, x20, 1
        addi x21, x21, 2
.word 0

taken2:
        addi x19, x19, 12
.word 0

taken:
        addi x18, x18, -1
        bge x18, x19, taken2
.word 0
```

7. <u>BLTU (Branch if less than unsigned)</u>:

Opcode Map:

imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU

Testcases:

a. Basic Branching Test:
Confirm that the BLTU instruction executes a branch correctly in a straightforward unsigned comparison.

b. Negative Comparison Case 1:
Verify that when rs1 is -1 and rs2 is 1, the branch is not taken, as -1 (0xFFFFFFFF) is greater than 1 when viewed unsigned.

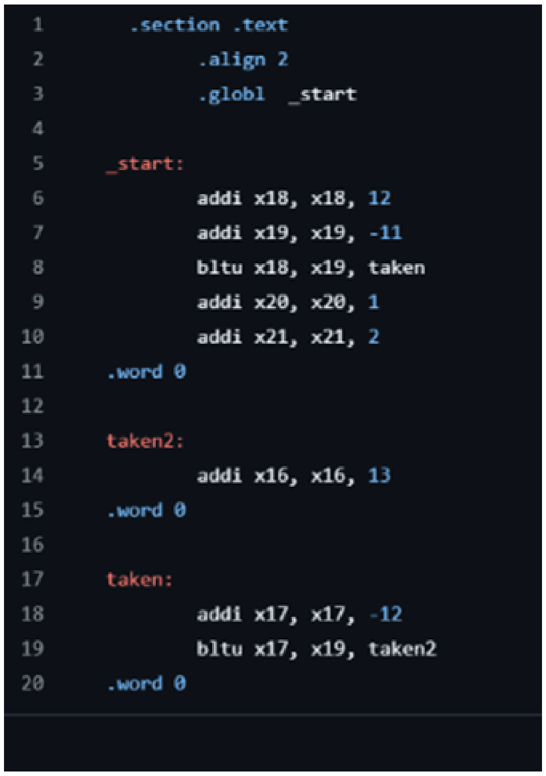
c. Negative Comparison Case 2:
 Ensure that when both rs1 and rs2 are -1, the operands are equal in the unsigned domain, so no branch occurs.

d. Zero Comparison:
 Check that with both rs1 and rs2 set to 0, no branch is performed because the operands are identical.

e. Non-Trivial Negative Comparison:
 Confirm that when rs1 is -50 and rs2 is -20, the unsigned comparison results in rs1 being less than rs2, and the branch is taken.

Assembly Code:

Covers less than (branch taken), +ve and -ve values and forward and backward branching:

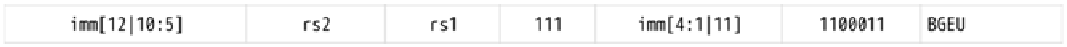


```
    .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, 12
        addi x19, x19, -11
        bltu x18, x19, taken
        addi x20, x20, 1
        addi x21, x21, 2
.word 0

taken2:
        addi x16, x16, 13
.word 0

taken:
        addi x17, x17, -12
        bltu x17, x19, taken2
.word 0
```

8. BGEU (Branch if greater than or equal to unsigned):

Opcode Map:

ECE586 Winter 2025
Team 07



imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

Testcases:

a. Greater Condition Test:
   Verify that when rs1 is unsigned greater than rs2, the branch is correctly taken.

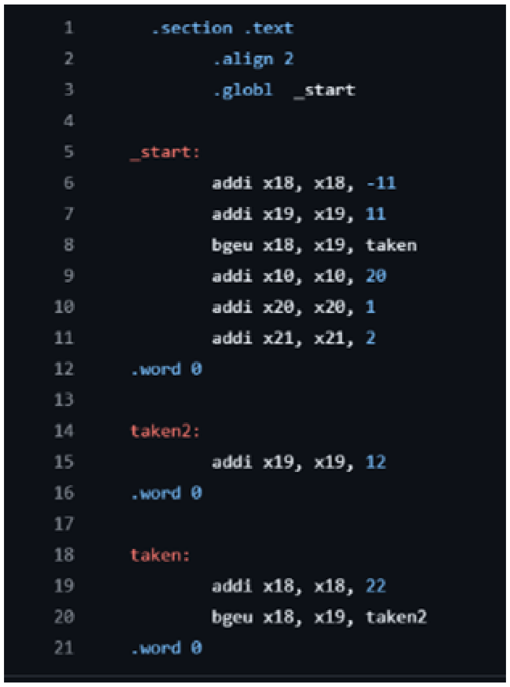
b. Equal Condition Test:
   Confirm that the branch is executed when both operands are equal, satisfying the "greater or equal" requirement.

c. Negative Operand Comparison:
   Ensure that even when rs1 is negative, if its unsigned value is still greater than rs2, the branch is taken.

d. Bidirectional Branching:
   Check that the instruction correctly handles branches in both forward and backward directions.

Assembly Code:

Covers greater than and equal to (branch taken), +ve and -ve values and forward and backward branching:



```
  .section .text
        .align 2
        .globl  _start

_start:
        addi x18, x18, -11
        addi x19, x19, 11
        bgeu x18, x19, taken
        addi x10, x10, 20
        addi x20, x20, 1
        addi x21, x21, 2
.word 0

taken2:
        addi x19, x19, 12
.word 0

taken:
        addi x18, x18, 22
        bgeu x18, x19, taken2
.word 0
```