RISCV-Busniss
Test Plan

Hector Cardenas
Riley Cox
Nathaniel Fraly
Stephen Weeks

Overview

Test cases will be mixed. For simple test cases, where allowable, tests can be written in C. Anything requiring corner cases should and will be tested in hand written assembly. Modules will, when applicable, be unit tested to verify single operations capabilities outside of integration testing. All test cases will use the tool chain provided in the assignment briefing.

NOTE: Bracket numbers relate to section in RISC-V ISA Manual

Overview
Register-Immediate [2.4.1]
- ADDI-Test
- SLTI-Test
- SLTIU-Test
- ANDI-Test
- ORI-Test
- XORI-Test
- SLLI-Test
- SRLI-Test
- SRAI-Test
- LUI-Test
- AUIPC-Test

Register-Register [2.4.2]
- ADD-Test
- SLT-Test
- SLTU-Test
- AND-Test
- OR-Test
- XOR-Test
- SLL-Test
- SRL-Test
- SUB-Test
- SRA-Test

Unconditional Jumps [2.5.1]
- JAL-Test
- JALR-Test

Conditional Branches [2.5.2]
- Branch-imm-Test
- BEQ-Test
- BNE-Test
- BLT-Test
- BLTU-Test
- BGE-Test
- BGEU-Test

Load and Store [2.6]
- LOAD-Test
- STORE-Test

Unit Test

Register-Immediate [2.4.1]

ADDI-Test

1. Zero added to a positive number
   Expectation: A number added to zero results in itself
   Outcome: A + 0 = A
2. Zero added to a negative number
   Expectation: A negative number added zero results in itself
   Outcome: -A + 0 = -A
3. Number added to its reciprocal
   Expectation: A number added to its reciprocal results in zero
   Outcome: A + -A = 0
4. A positive number added to a negative number
   Expectation: A positive added to a negative results in a number somewhere between the two
   Outcome: A + -B = C s.t. -B < C < A
5. A positive number added to a positive number
   Expectation: A positive added to a positive results in a new positive greater than the two
   Outcome: A + B = C s.t. A < C & B < C
6. A negative number added to a negative number
   Expectation: A negative number added to a negative number results in a new negative number less than both
   Outcome: -A + -B = -C s.t. -A > -C & -B > -C
7. Make sure sign extension is occurring appropriately
   Expectation: The immediate field is sign extended correctly
   Outcome: Tests above should switch so that a positive and negative get sign extended
8. Verify MV operation works as expected
   Expectation: The pseudo instruction MV uses `ADDI, rd, rs1, 0`, make sure rd ← rs1
   Outcome: MV op results in rs1 being stored in rd.

9. Verify NOP operation works as expected
   Expectation: The pseudo operation NOP performs as expected
   Outcome: x0 should not change AT ALL

SLTI-Test

1. Rs1 < sign_ext(immediate) [Both Positive]
   Expectation: rd is set to 1 when rs1 < imm
   Outcome: rd ← 1
2. Rs1 < sign_ext(immediate) [Both Negative]
   Expectation: rd is set to 1 when rs1 < imm
   Outcome: rd ← 1
3. Rs1 < sign_ext(immediate) [rs1 < 0 | imm > 0]
   Expectation: rd is set to 1 when rs1 < imm
   Outcome: rd ← 1
4. Rs1 < sign_ext(immediate) [rs1 = 0 | imm > 0]
   Expectation: rd is set to 1 when rs1 < imm
   Outcome: rd ← 1
5. Rs1 > sign_ext(immediate) [Both Positive]
   Expectation: rd is set to 0 when rs1 > imm
   Outcome: rd ← 0
6. Rs1 > sign_ext(immediate) [Both Negative]
   Expectation: rd is set to 0 when rs1 > imm
   Outcome: rd ← 0
7. Rs1 > sign_ext(immediate) [rs1 > 0 | imm < 0]
   Expectation: rd is set to 0 when rs1 > imm
   Outcome: rd ← 0
8. Rs1 > sign_ext(immediate) [rs1 > 0 | imm = 0]
   Expectation: rd is set to 0 when rs1 > imm
   Outcome: rd ← 0
9. Rs1 == immediate s.t. Rs1 =/= 0
   Expectation: rd is set to 0 when rs1 > imm
   Outcome: rd ← 0

10. Rs1 == immediate s.t. Rs1 == 0
    Expectation: rd is set to 0 when rs1 > imm
    Outcome: rd ← 0

SLTIU-Test

1. Rs1 < immediate [Both Positive]
   Expectation: rd is set to 1 when rs1 < imm
   Outcome: rd ← 1
2. Rs1 < immediate [Both Negative] (this is an unsigned operation)
   Expectation: unsigned negative number s.t. rs1 < imm
   Outcome: rd ← 1
3. Rs1 < immediate s.t. rs1 > 0 & imm < 0
   Expectation: unsigned negative number s.t. rs1 < imm
   Outcome: rd ← 1
4. Rs1 > immediate [Both Positive]
   Expectation: rd is set to 0 when rs1 > imm
   Outcome: rd ← 0
5. Rs1 > immediate [Both negative]
   Expectation: rd is set to 0 from signed → unsigned s.t. rs1 > imm
   Outcome: rd ← 0
6. Rs1 > immediate s.t. rs1 < 0 & imm > 0
   Expectation: unsigned negative number for rs1 s.t. rs1 > imm
   Outcome: rd ← 0
7. Rs1 == immediate s.t. rs1 =/= 0
   Expectation: rd is set to 0 because rs1 == imm
   Outcome: rd ← 0
8. Rs1 == immediate s.t. rs1 = 0
   Expectation: rd is set to 0 because rs1 == imm
   Outcome: rd ← 0
9. The SEQZ rd, rs command when rs is zero
   Expectation: rd is set to 1 because rs == 0
   Outcome: rd ← 1

10. The SEQZ rd, rs command when rs is not zero
    Expectation: rd is set to 0 because rs =/= 0
    Outcome: rd ← 0

ANDI-Test

1. AND A with 0x000
   Expectation: A & 0x000 = 0
   Outcome: rd ← 0
2. AND A with 0xFFF
   Expectation: A & 0xFFF = A
   Outcome: rd ← A
3. AND A with 0xAAA
   Expectation: A & 0xAAA results in an expected calculated value
   Outcome: rd ← calculated result
4. AND A with 0x555
   Expectation: A & 0x555 results in an expected calculated value
   Outcome: rd ← calculated result

ORI-Test

1. OR A with 0x000
   Expectation: A | 0x000 = A
   Outcome: rd ← A
2. OR A with 0xFFF
   Expectation: A | 0xFFF = 0xFFF
   Outcome: rd ← 0xFFF
3. OR A with 0xAAA
   Expectation: A | 0xAAA results in an expected calculated value
   Outcome: rd ← calculated result
4. OR A with 0x555
   Expectation: A | 0x555 results in an expected calculated value
   Outcome: rd ← calculated result

XORI-Test

1. XOR A with ~A
   Expectation: A ^ ~A = 0xFFF
   Outcome: rd ← 0xFFF
2. XOR A with A
   Expectation: A ^ A = 0x000
   Outcome: rd ← 0
3. Verify that NOT operation works as expected
   Expectation: The pseudo instruction NOT uses ‘XORI rd, rs1, -1’
   Outcome: rd ← ~rs1
4. XOR A with 0xAAA
   Expectation: A ^ 0xAAA results in an expected calculated value
   Outcome: rd ← calculated result
5. XOR A with 0x555
   Expectation: A ^ 0x555 results in an expected calculated value
   Outcome: rd ← calculated result

SLLI-Test

1. Shift left so that a number is all 0s
   Expectation: Shift a non-zero number (preferably -1) s.t. the result is all 0’s
   Outcome: rd ← 0
2. Shift left 0 and it should not change
   Expectation: Shift when rs1 is zero and make sure it stays zero
   Outcome: rd ← 0
3. Shift a negative number left
   Expectation: Shift a negative number s.t. the lower bits of the shamt are zero
   Outcome: rd ← (-A << shamt)
4. Shift a positive number left
   Expectation: Shift a positive number s.t. the result is the lower bits of the shamt are zero
   Outcome: rd ← (A << shamt)

5. Shift a number by a value greater than 31
   Expectation: When storing a value > 31 we only shift by the lower 5 bits
   Outcome: rd ← (A << imm[4:0])
6. Shift a number by a negative value
   Expectation: When storing a negative value, shift by the lower 5 bits of that amount
   Outcome: rd ← (A << -imm[4:0])
7. Shift left 0 by a non-zero number and it should not change
   Expectation: Shift of zero should always be zero
   Outcome: rd ← 0
8. Shift left a non-zero number by 0 and it should not change
   Expectation: Shift by zero should not change a number
   Outcome: rd ← (A << 0)

SRLI-Test

1. Shift right so that a number is all 0s
   Expectation: Shift a non-zero number (preferably -1) s.t. The result is all 0's
   Outcome: rd ← 0
2. Shift right 0 and it should not change
   Expectation: Shift when rs1 is zero and make sure it stays zero
   Outcome: rd ← 0
3. Shift a negative number right
   Expectation: Shift a negative number s.t. the result is the upper bits are zero
   Outcome: rd ← (-A >> shamt)
4. Shift a positive number right
   Expectation: Shift a positive number s.t. the result is the upper bits are zero
   Outcome: rd ← (A >> shamt)
5. Shift a number by a value greater than 31
   Expectation: Should shift using the lower 5 bits of the immediate

Outcome: rd ← (A >> imm[4:0])

6. Shift a number by a negative value
   Expectation: When storing a negative value, shift by the lower 5 bits of that amount
   Outcome: rd ← (A << -imm[4:0])
7. Shift right 0 by a non-zero number and it should not change
   Expectation: Shift of zero should always be zero
   Outcome: rd ← 0
8. Shift right a non-zero number by 0 and it should not change
   Expectation: Shift by zero should not change a number
   Outcome: rd ← (A >> 0)

SRAI-Test

1. Shift right 0 and it should not change
   Expectation: Shift when rs1 is zero and make sure its stays zero
   Outcome: rd ← 0
2. Shift a positive number right
   Expectation: Shift a positive number right s.t. the result is the upper bits are zero
   Outcome: rd ← (A >> (0)shamt)
3. Shift a negative number right
   Expectation: Shift a negative number right s.t. the result is the upper bits are 1's
   Outcome: rd ← (A >> (1)shamt)
4. Shift a number by a value greater than 31
   Expectation: Should shift using the lower 5 bits of the immediate
   Outcome: rd ← (A >> imm[4:0])
5. Shift right 0xFFFFFFFF and it should not change
   Expectation: Shit when rs1 is all 1's and make sure it stays all 1's
   Outcome: rd ← 0xFFFFFFFF

6. Shift a number by a negative value
   Expectation: When storing a negative value, shift by the lower 5 bits of that amount
   Outcome: rd ← (A >>-imm[4:0]) with sign preservation
7. Shift right 0 by a non-zero number and it should not change
   Expectation: Shift of zero should always be zero
   Outcome: rd ← 0
8. Shift right a non-zero number by 0 and it should not change
   Expectation: Shift by zero should not change a number
   Outcome: rd ← (A >> 0)

LUI-Test

1. Load a positive number into a register
   Expectation: Be able to load a 20 bit positive number into a register that isn’t a special register (x0 - x4, x8)
   Outcome: rd ← immediate value
2. Load a negative number into a register
   Expectation: Be able to load a 20 bit negative number into a register that isn’t a special register (x0 - x4, x8)
   Outcome: rd ← immediate value
3. Load a number larger than 20 bits
   Expectation: It should only load the lower 20 bits
   Outcome: rd ← lower load 20 bits
   NOTE: THIS PROBABLY DOESN’T MATTER

AUIPC-Test

1. Add a positive number to pc and load into a register
   Expectation: Be able to take a 20 bit positive number add to pc and load it into a register that isn’t a special register
   Outcome: rd ← pc + (imm[31:12] << 12)
2. Add a negative number to pc and load into a register
   Expectation: Be able to take a 20 bit negative number add to pc and load it into a register that isn't a special register

Outcome: rd ← pc + (imm[31:12] << 12)

3. Add a number larger than 20 bits to the pc
   Expectation: Be able to take the lower 20 bits and add to pc then load it into a register that isn't a special register
   Outcome: rd ← pc + (imm[31:12] << 12)

Register-Register [2.4.2]

ADD-Test

1. Zero added to a positive number
   Expectation: A number added to zero results in itself
   Outcome: A + 0 = A
2. Zero added to a negative number
   Expectation: A negative number added zero results in itself
   Outcome: -A + 0 = -A
3. Number added to its reciprocal
   Expectation: A number added to its reciprocal results in zero
   Outcome: A + -A = 0
4. A positive number added to a negative number
   Expectation: A positive added to a negative results in a number somewhere between the two
   Outcome: A + -B = C s.t. -B < C < A
5. A positive number added to a positive number
   Expectation: A positive added to a positive results in a new positive greater than the two
   Outcome: A + B = C s.t. A < C & B < C
6. A negative number added to a negative number
   Expectation: A negative number added to a negative number results in a new negative number less than both
   Outcome: -A + -B = -C s.t. -A > -C & -B > -C

SLT-Test

1. Rs1 < rs2 [Both Positive]
   Expectation: rd is set to 1 when rs1 < rs2
   Outcome: rd ← 1
2. Rs1 < rs2 [Both Negative]
   Expectation: rd is set to 1 when rs1 < rs2
   Outcome: rd ← 1
3. Rs1 < rs2 [rs1 < 0 | rs2 > 0]
   Expectation: rd is set to 1 when rs1 < rs2
   Outcome: rd ← 1
4. Rs1 < rs2 [rs1 = 0 | rs2 > 0]
   Expectation: rd is set to 1 when rs1 < rs2
   Outcome: rd ← 1
5. Rs1 > rs2 [Both Positive]
   Expectation: rd is set to 0 when rs1 > rs2
   Outcome: rd ← 0
6. Rs1 > rs2 [Both Negative]
   Expectation: rd is set to 0 when rs1 > rs2
   Outcome: rd ← 0
7. Rs1 > rs2 [rs1 > 0 | rs2 < 0]
   Expectation: rd is set to 0 when rs1 > rs2
   Outcome: rd ← 0
8. Rs1 > rs2 [rs1 > 0 | rs2 = 0]
   Expectation: rd is set to 0 when rs1 > rs2
   Outcome: rd ← 0
9. Rs1 == rs2 s.t. Rs1 =/= 0
   Expectation: rd is set to 0 when rs1 > rs2
   Outcome: rd ← 0
10. Rs1 == rs2 s.t. Rs1 == 0
    Expectation: rd is set to 0 when rs1 > rs2
    Outcome: rd ← 0

SLTU-Test

1. Rs1 < Rs2 [Both Positive]
   Expectation: rd is set to 1 when rs1 < rs2
   Outcome: rd ← 1
2. Rs1 < Rs2 [Both Negative] (this is an unsigned operation)
   Expectation: unsigned negative number s.t. rs1 < rs2
   Outcome: rd ← 1
3. Rs1 < Rs2 s.t. rs1 > 0 & rs2 < 0
   Expectation: unsigned negative number s.t. rs1 < rs2
   Outcome: rd ← 1
4. Rs1 > Rs2 [Both Positive]
   Expectation: rd is set to 0 when rs1 > rs2
   Outcome: rd ← 0
5. Rs1 > Rs2 [Both negative]
   Expectation: rd is set to 0 from signed → unsigned s.t. rs1 > rs2
   Outcome: rd ← 0
6. Rs1 > Rs2 s.t. rs1 < 0 & rs2 > 0
   Expectation: unsigned negative number for rs1 s.t. rs1 > rs2
   Outcome: rd ← 0
7. Rs1 == Rs2 s.t. rs1 =/= 0
   Expectation: rd is set to 0 because rs1 == rs2
   Outcome: rd ← 0
8. Rs1 == Rs2 s.t. rs1 = 0
   Expectation: rd is set to 0 because rs1 == rs2
   Outcome: rd ← 0
9. The SNEZ rd, rs command when rs is zero
   Expectation: rd is set to 0 because rs == 0
   Outcome: rd ← 0
10. The SNEZ rd, rs command when rs is not zero
    Expectation: rd is set to 1 because rs =/= 0
    Outcome: rd ← 1

AND-Test

1. AND A with 0x000
   Expectation: A & 0x000 = 0
   Outcome: rd ← 0
2. AND A with 0xFFF
   Expectation: A & 0xFFF = A
   Outcome: rd ← A
3. AND A with 0xAAA
   Expectation: A & 0xAAA results in an expected calculated value
   Outcome: rd ← calculated result
4. AND A with 0x555
   Expectation: A & 0x555 results in an expected calculated value
   Outcome: rd ← calculated result

OR-Test

1. OR A with 0x000
   Expectation: A | 0x000 = A
   Outcome: rd ← A
2. OR A with 0xFFF
   Expectation: A | 0xFFF = 0xFFF
   Outcome: rd ← 0xFFF
3. OR A with 0xAAA
   Expectation: A | 0xAAA results in an expected calculated value
   Outcome: rd ← calculated result
4. OR A with 0x555
   Expectation: A | 0x555 results in an expected calculated value
   Outcome: rd ← calculated result

XOR-Test

1. XOR A with ~A
   Expectation: A ^ ~A = 0xFFF
   Outcome: rd ← 0xFFF
2. XOR A with A

Expectation: A ^ A = 0x000
Outcome: rd ← 0

3. XOR A with 0xAAA
   Expectation: A ^ 0xAAA results in an expected calculated value
   Outcome: rd ← calculated result
4. XOR A with 0x555
   Expectation: A ^ 0x555 results in an expected calculated value
   Outcome: rd ← calculated result

SLL-Test

1. Shift left so that a number is all 0s
   Expectation: Shift a non-zero number (preferably -1) s.t. the result is all 0's
   Outcome: rd ← 0
2. Shift left 0 and it should not change
   Expectation: Shift when rs1 is zero and make sure it stays zero
   Outcome: rd ← 0
3. Shift a negative number left
   Expectation: Shift a negative number s.t. the lower bits of the shamt are zero
   Outcome: rd ← (-A << shamt)
4. Shift a positive number left
   Expectation: Shift a positive number s.t. the result is the lower bits of the shamt are zero
   Outcome: rd ← (A << shamt)
5. Shift a number by a value greater than 31
   Expectation: When storing a value > 31 we only shift by the lower 5 bits
   Outcome: rd ← (A << rs2[4:0])
6. Shift a number by a negative value
   Expectation: When storing a negative value, shift by the lower 5 bits of that amount
   Outcome: rd ← (A << -rs2[4:0])

SRL-Test

1. Shift right so that a number is all 0s
   Expectation: Shift a non-zero number (preferably -1) s.t. The result is all 0’s
   Outcome: rd ← 0
2. Shift right 0 and it should not change
   Expectation: Shift when rs1 is zero and make sure it stays zero
   Outcome: rd ← 0
3. Shift a negative number right
   Expectation: Shift a negative number s.t. the result is the upper bits are zero
   Outcome: rd ← (-A >> shamt)
4. Shift a positive number right
   Expectation: Shift a positive number s.t. the result is the upper bits are zero
   Outcome: rd ← (A >> shamt)
5. Shift a number by a value greater than 31
   Expectation: Should shift using the lower 5 bits of rs2
   Outcome: rd ← (A >> rs2[4:0])
6. Shift a number by a negative value
   Expectation: When storing a negative value, shift by the lower 5 bits of that amount
   Outcome: rd ← (A << -rs2[4:0])

SUB-Test

1. Subtract two positive numbers from each other
   Expectation: A - B = C
   Outcome: rd ← C
2. Subtract two negative numbers
   Expectation: -A - (-B) = C
   Outcome: rd ← C
3. Subtract pos/neg from each other s.t. rs1 > 0 & rs2 < 0
   Expectation: A - (-B) = C s.t. C > A & C > B
   Outcome: rd ← C

4. Subtract pos/neg from each other s.t. rs1 < 0 & rs2 > 0
   Expectation: -A - B = C s.t C < A & C < -B
   Outcome: rd ← C
5. Subtract a number from its reciprocal
   Expectation: A - (-A) = 2A
   Outcome: rd ← 2A
6. Subtract a positive number from zero
   Expectation: A - 0 = A
   Outcome: rd ← A
7. Subtract a negative number from zero
   Expectation: -A - 0 = -A
   Outcome: rd ← -A

SRA-Test

1. Shift right 0 and it should not change
   Expectation: Shift when rs1 is zero and make sure its stays zero
   Outcome: rd ← 0
2. Shift a positive number right
   Expectation: Shift a positive number right s.t. the result is the upper bits are zero
   Outcome: rd ← (A >> (0)shamt)
3. Shift a negative number right
   Expectation: Shift a negative number right s.t. the result is the upper bits are 1's
   Outcome: rd ← (A >> (1)shamt)
4. Shift a number by a value greater than 31
   Expectation: Should shift using the lower 5 bits of rs2
   Outcome: rd ← (A >> rs2[4:0])
5. Shift right 0xFFFFFFFF and it should not change
   Expectation: Shit when rs1 is all 1's and make sure it stays all 1's
   Outcome: rd ← 0xFFFFFFFF

6. Shift a number by a negative value
    Expectation: When storing a negative value, shift by the lower 5 bits of that amount
    Outcome: rd ← (A << -rs2[4:0]) with sign preservation

Unconditional Jumps [2.5.1]

JAL-Test

1. Should not jump to a 2-byte aligned address
    Expectation: When the LSB-1 is not zero it should trap
    Outcome: An error is given and we exit or hang
2. Should jump positively up in memory
    Expectation: I should jump up in memory address hierarchy
    Outcome: End up at a higher memory location in the PC
3. Should jump negatively down in memory
    Expectation: I should jump down in memory address hierarchy
    Outcome: End up at a lower memory location in the PC

JALR-Test

1. Should not jump to a 2-byte aligned address
    Expectation: When the LSB-1 is not zero it should trap
    Outcome: An error is given and we exit or hang
2. Should jump positively up in memory
    Expectation: I should jump up in memory address hierarchy
    Outcome: End up at a higher memory location in the PC
3. Should jump negatively down in memory
    Expectation: I should jump down in memory address hierarchy
    Outcome: End up at a lower memory location in the PC

Conditional Branches [2.5.2]

Branch-imm-Test

1. Providing an unaligned imm [imm % 4 != 0]
   Expectation: When the bottom two bits are not 0 we should trap
   Outcome: An error is given and we exit or hang
2. Providing a positive imm value [imm[MSB] = 0]
   Expectation: We branch to a higher PC
   Outcome: PC ← PC + sign-extended imm
3. Providing a negative imm value [imm[MSB] = 1]
   Expectation: We branch to a lower PC
   Outcome: PC ← PC + sign-extended imm

BEQ-Test

4. Rs1 < rs2 [same sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
5. Rs1 > rs2 [same sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
6. Rs1 = rs2
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
7. Rs1 < rs2 [different sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
8. Rs1 > rs2 [different sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4

BNE-Test

1. Rs1 < rs2 [same sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
2. Rs1 > rs2 [same sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
3. Rs1 = rs2
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
4. Rs1 < rs2 [different sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
5. Rs1 > rs2 [different sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm

BLT-Test

1. Rs1 < rs2 [same sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
2. Rs1 > rs2 [same sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
3. Rs1 = rs2
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
4. Rs1 < rs2 [different sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
5. Rs1 > rs2 [different sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4

BLTU-Test

1. Rs1 < rs2 [same sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
2. Rs1 > rs2 [same sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
3. Rs1 = rs2
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
4. Rs1 < rs2 [different sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
5. Rs1 > rs2 [different sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm

BGE-Test

1. Rs1 < rs2 [same sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
2. Rs1 > rs2 [same sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
3. Rs1 = rs2
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
4. Rs1 < rs2 [different sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
5. Rs1 > rs2 [different sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm

BGEU-Test

1. Rs1 < rs2 [same sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4
2. Rs1 > rs2 [same sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
3. Rs1 = rs2
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
4. Rs1 < rs2 [different sign]
   Expectation: We branch to target
   Outcome: PC ← PC + sign-extended imm
5. Rs1 > rs2 [different sign]
   Expectation: We do not branch to target
   Outcome: PC ← PC + 4

Load and Store [2.6]

LOAD-Test

6. Should be able to properly load a word
   Expectation: Load a word from an aligned memory location
   Outcome: rd ← mem[word]
7. Should be able to properly load a positive half-word
   Expectation: Load a positive (MSB = 0) half-word from an aligned memory location
   Outcome: rd ← mem[HW] with positive sign extension
8. Should be able to properly load a negative half-word
   Expectation: Load a negative (MSB = 1) half-word from an aligned memory location
   Outcome: rd ← mem[HW] with negative sign extension

9. Should be able to properly load a positive byte
    Expectation: Load a positive (MSB = 0) byte from an aligned memory location
    Outcome: rd ← mem[byte] with positive sign extension
10. Should be able to properly load a negative byte
    Expectation: Load a negative (MSB = 1) byte from an aligned memory location
    Outcome: rd ← mem[byte] with negative sign extension
11. STRETCH: handle exception for misaligned memory
    Expectation: A trap should happen when loading misaligned memory
    Outcome: rd ← mem[misaligned] raises a trap

STORE-Test

1. Should store a word into the proper memory location
    Expectation: A known word value is written to an expected memory location
    Outcome: mem[address+3 : address] ← rs2
2. Should store a half-word into the proper memory location
    Expectation: A known half-word value is written to an expected memory location
    Outcome: mem[address+1 : address] ← rs2[15:0]
3. Should store a byte into the proper memory location
    Expectation: A known byte value is written to an expected memory location
    Outcome: mem[address] ← rs2[7:0]
4. We do not write to memory location outside the bounds of the data type
    Expectation: Do not overwrite memory not meant for that data size
    Outcome: Never write more than the needed data length

Unit Test

Unit testing will focus around functionality of each sub operation class. All class operations should be well tested, within reason, to expected functionality. Unit testing can take 2 forms: actual C-code test in the unit suite creating instructions and executing them, or .mem files with expected value output. The latter should come with expected results and behavior so test results are quickly and easily verified.

Goal of unit testing is to verify functionality and fix bugs EARLY. This is to ease debug time and allow for early detection of bugs for the integration team. **NOTE: Bugs found during integration testing should also be converted to unit testing as this is an obvious edge case that was found.**