Experiment No 5
Exploring RISC-V Assembly: Shift Instructions, Conditional Branching, Jumps, and Control Structures

Objective

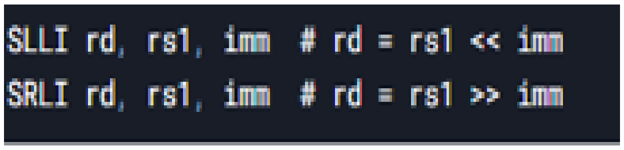
This lab aims to provide students with a comprehensive understanding of RISC-V assembly programming, focusing on:

- logical instructions with immediate constants.
- Conditional statements using beq and bne.
- Unconditional jumps using the j instruction.
- Control structures, including IF, IF-ELSE, while loops, and for loops.

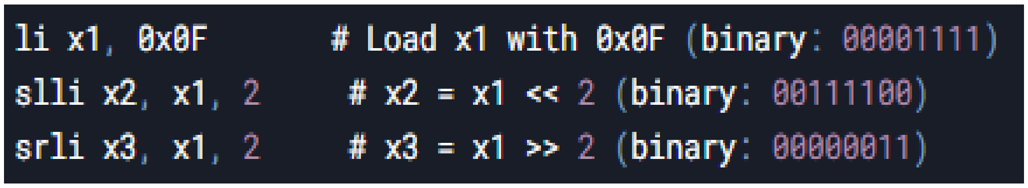
1. Shift Logical Instructions with Immediate Constants
Shift instructions are used to move bits within a register to the left or right. RISC-V provides the following shift logical instructions:

- **SLLI (Shift Left Logical Immediate)**: Shifts the bits in a register to the left by an immediate value, filling the rightmost bits with zeros.

- **SRLI (Shift Right Logical Immediate)**: Shifts the bits in a register to the right by an immediate value, filling the leftmost bits with zeros



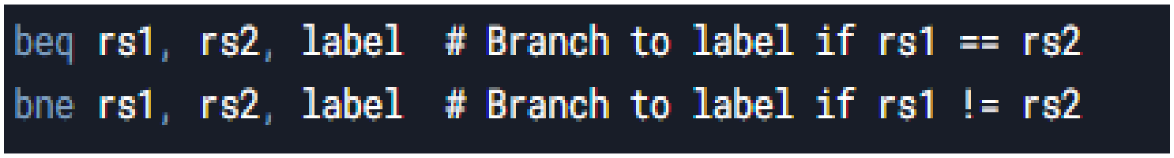
```
SLLI rd, rs1, imm  # rd = rs1 << imm
SRLI rd, rs1, imm  # rd = rs1 >> imm
```



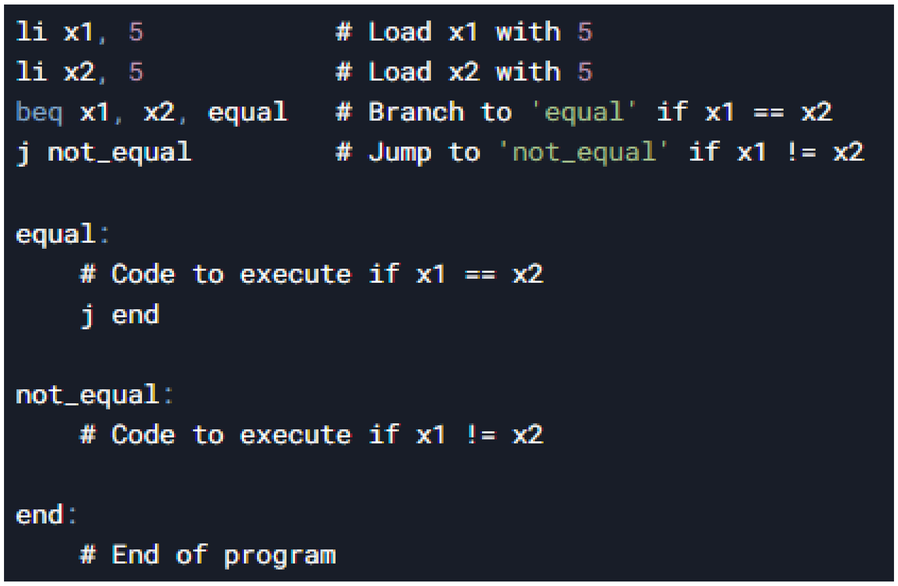
```
li x1, 0x0F       # Load x1 with 0x0F (binary: 00001111)
slli x2, x1, 2     # x2 = x1 << 2 (binary: 00111100)
srli x3, x1, 2     # x3 = x1 >> 2 (binary: 00000011)
```

2. Conditional Statements (beq and bne)

Conditional statements allow the program to make decisions based on certain conditions. In RISC-V, conditional branching is achieved using the beq (branch if equal) and bne (branch if not equal) instructions.



```
beq rs1, rs2, label  # Branch to label if rs1 == rs2
bne rs1, rs2, label  # Branch to label if rs1 != rs2
```



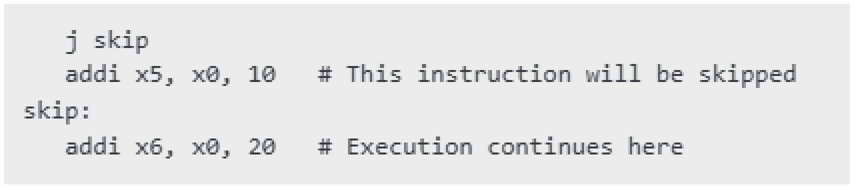
```
li x1, 5             # Load x1 with 5
li x2, 5             # Load x2 with 5
beq x1, x2, equal    # Branch to 'equal' if x1 == x2
j not_equal          # Jump to 'not_equal' if x1 != x2

equal:
    # Code to execute if x1 == x2
    j end

not_equal:
    # Code to execute if x1 != x2

end:
    # End of program
```

beq x1, x2, equal: If x1 equals x2, the program branches to the equal label.

j not_equal: If the condition is not met, the program jumps to the not equal label.

3. Unconditional Jumps (j Instruction)

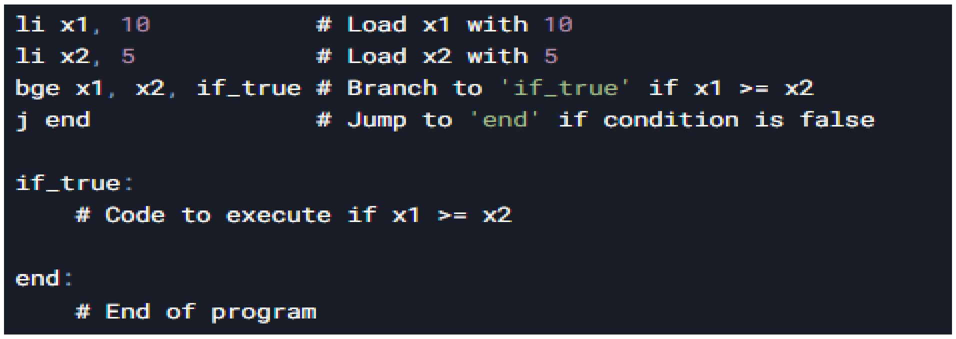
Unconditional jumps are used to transfer control to a specific part of the program without any condition.



```
   j skip
   addi x5, x0, 10   # This instruction will be skipped
skip:
   addi x6, x0, 20   # Execution continues here
```

4. Control Structures in RISC-V Assembly

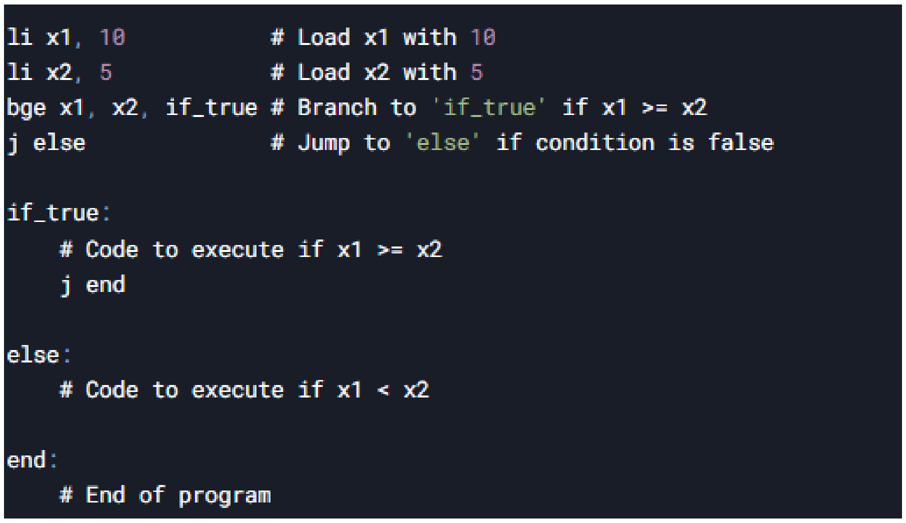
Control structures are used to control the flow of execution in a program. In RISC-V, these structures are implemented using conditional and unconditional jumps.
IF Statement: An if statement executes a block of code if a condition is true.



```
li x1, 10            # Load x1 with 10
li x2, 5             # Load x2 with 5
bge x1, x2, if_true # Branch to 'if_true' if x1 >= x2
j end                # Jump to 'end' if condition is false

if_true:
    # Code to execute if x1 >= x2

end:
    # End of program
```

IF-ELSE Statements in RISC-V

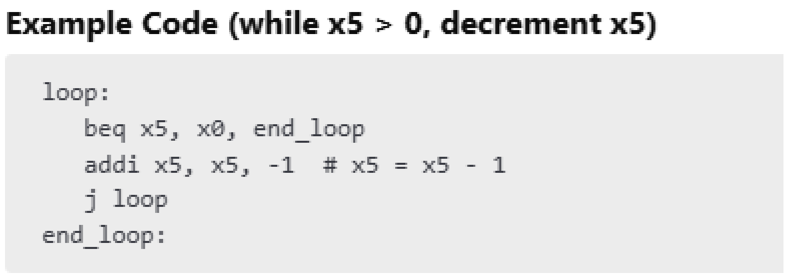
An IF-ELSE statement executes one block of code if a condition is true and another block if the condition is false.



```
li x1, 10            # Load x1 with 10
li x2, 5             # Load x2 with 5
bge x1, x2, if_true # Branch to 'if_true' if x1 >= x2
j else               # Jump to 'else' if condition is false

if_true:
    # Code to execute if x1 >= x2
    j end

else:
    # Code to execute if x1 < x2

end:
    # End of program
```

5. WHILE Loop in RISC-V

A while loop repeats a block of code as long as a condition is true. Using beq to implement a while loop.



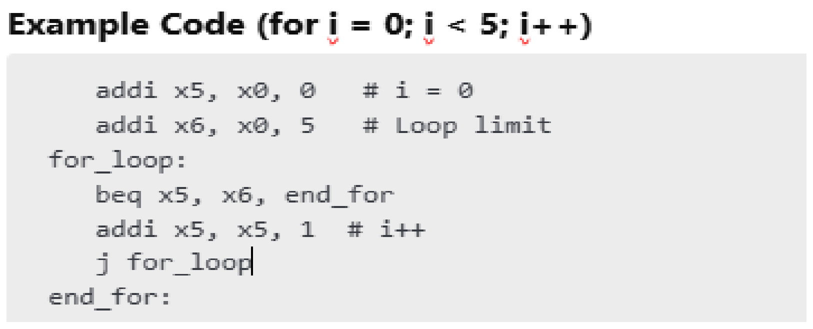
Example Code (while x5 > 0, decrement x5)

```
loop:
   beq x5, x0, end_loop
   addi x5, x5, -1  # x5 = x5 - 1
   j loop
end_loop:
```

Uses a while loop to decrement x10 until it reaches 0.

6. FOR Loop in RISC-V

A for loop repeats a block of code a specific number of times. using beq to implement a for loop.



Example Code (for i = 0; i < 5; i++)

```
   addi x5, x0, 0   # i = 0
   addi x6, x0, 5   # Loop limit
for_loop:
   beq x5, x6, end_for
   addi x5, x5, 1  # i++
   j for_loop
end_for:
```

Lab Tasks:

Task 1:

Write a RISC-V assembly program that performs the following operations:

- Load the immediate value **4** into register **t0**.
- Perform a **logical left shift** on **t0** by **1** bit and store the result in **t2**.
- Perform a **logical right shift** on **t0** by **1** bit and store the result in **t3**.
- Perform an **arithmetic right shift** on **t0** by **1** bit and store the result in **t4**.

Task 2:

Write a RISC-V assembly program to demonstrate **conditional branching** using beq, bne, instructions. Perform the following tasks:

Conditional Branching using beq (Branch if Equal):

- Load two immediate values into registers **s0 and s1.**
- Perform a **logical left shift** on **s1** by **2** bits.
- Compare **s0** and **s1** using beq. If they are equal, branch to the label target.
- If the branch is taken, skip the next two instructions.
- At the target label, add **s0** to **s1** and store the result in **s1**.

<u>Conditional Branching using bne (Branch if Not Equal)</u>:

- Repeat the same steps as in beq but use bne instead.
- If s0 and s1 are not equal, jump to the target label.
- Verify that, in this case, the branch is not taken because s0 == s1.

Task 3:

Write a RISC-V assembly program to demonstrate **unconditional branching** using the j (jump) instruction. Perform the following tasks:

- Use the **j instruction** to **unconditionally jump** to the label target.
- Ensure that all instructions between the j instruction and the target label are **skipped** (not executed).
- At the target label, perform an addition operation:
- Add the value in register **s0** to **s1** and store the result in **s1**.

Task 4:

Write a RISC-V assembly program that converts a **C code** into an **Assembly cod**e by using an **IF and IF-ELSE statements**. Perform the following tasks:

<u>C Code IF Statement:</u>

if (apples = = oranges) // C code check i==j

f = g + h;

apples = oranges − h;

<u>Convert the above C code into Assembly code:</u>

- Compare the values of apples (s0) and oranges (s1) using bne.
- If they are not equal, skip the next instruction and jump to L1.
- If they are equal, execute f = g + h by adding s3 (g) and s4 (h) and storing the result in s2 (f).
- At label L1, execute apples = oranges - h by subtracting s4 (h) from s1 (oranges) and storing the result in s0 (apples).

<u>C Code IF-ELSE Statement:</u>

if (apples = = oranges)

f = g + h;

else

```
  apples = oranges − h;
```

Convert the above C code into Assembly code:

- Compare s0 (apples) and s1 (oranges) using bne.
- if they are not equal, jump to L1 (ELSE block).
- If they are equal, execute f = g + h.
- Use the j instruction to skip the ELSE block and jump to L2.
- At label L1, execute apples = oranges - h (ELSE block).
- Label L2 marks the end of the IF-ELSE structure.

Task 5:

Write a RISC-V assembly program to implement a **while loop** that calculates the power of **2** such that **2^x = 128.**

While loop : High-Level Code

```
// determines the power

// of x such that 2^x = 128

int pow = 1;

int x = 0;

do {

 pow = pow * 2;

 x = x + 1;

} while (pow!= 128);
```

Convert the above C code into Assembly code:

Follow these steps:

- Set pow = 1 in s0.
- Set x = 0 in s1.
- Store the constant 128 in t0 (used for comparison).

1. **Loop Execution:**

- Multiply pow by 2 using a left shift operation (slli).
- Increment x by 1 using addi.

- Use the bne instruction to check if pow != 128.
- If pow is not equal to 128, repeat the loop.
- If pow equals 128, exit the loop and proceed to the done label.

Task 6:

Write a RISC-V assembly program to implement a **for loop** that adds numbers from **0 to 9** and stores the sum in a register. Follow these steps:

For loop : High-Level Code:

```
// add the numbers from 0 to 9

int sum = 0;

int i;

for (i = 0; i < 10; i = i + 1) {

   sum = sum + i; }
```

Convert the above C code into Assembly code:

Follow these steps:

Initialize Variables:

- Set sum = 0 in **s1**.
- Set i = 0 in **s0**.
- Store the constant 10 in **t0** (loop condition).

Loop Execution:

- Check if i >= 10 using bge. If **true**, exit the loop and jump to done.
- If **false**, execute sum = sum + i.
- Increment i by **1**.
- Jump back to the beginning of the loop to check the condition again.

Loop Termination:

- When I reach **10**, the loop exits and proceed to done.