Shift Operation

↳ follows I-type
but modified

slli $\underbrace{x11}_{rd}$, $\underbrace{x19}_{rs}$, $\underbrace{4}_{imm}$

31 … 0

6	6	5	3	5	7

31 … 0

funct6	imm	rs1	funct3	rd	opcode

$\underbrace{\text{funct6 imm}}$
immediate broken down into 2 fields.

Why ?? ⟹ If you shift a 64 bit value more than 63 bits what happens?

Shift Left

↳ shift left and fill the positions with 0

⟹ We can perform multiplication by 2^i using slli.

slli $\underbrace{x11}$, x19, 4
↓
the value that will be stored in x11 is basically,

val in x_{19} * 2^4

Shift Right

↳ shift right and fill the positions with 0

⟹ We can perform division. by 2^i using srli.

Srli $\underbrace{x11}$, x19, 4
↓
the value that will be stored in x11 is basically,

val in x_{19} / 2^4

And ⟹ Bit Musking

and $\underbrace{x_9}_{Rd}$, $\underbrace{x_{10}}_{Rs1}$, $\underbrace{x_{11}}_{Rs2}$

x_{10} = 0000 ..... 0000 1100 1100

x_{11} = 0000 ..... 0000 0000 1100

x_9 = 0000 .... 0000 0000 1100

only these two bits should remain as it is, rest 0.

OR ⟹ Include Bits

and $\underbrace{x_9}_{Rd}$, $\underbrace{x_{10}}_{Rs1}$, $\underbrace{x_{11}}_{Rs2}$

x_{10} = 0000 ..... 0000 1100 0000

x_{11} = 0000 ..... 0000 0011 0000

x_9 = 0000 .... 0000 1111 0000

You want to set these bits to 1 and rest should remain as it is.

XOR ⟹ Can work as Buffer / Not

XOR $\underbrace{x_9}_{Rd}$, $\underbrace{x_{10}}_{Rs1}$, $\underbrace{x_{11}}_{Rs2}$

A	B	$A \oplus B$
0	0	0
1	0	1
0	1	1
1	1	0

⟹ XOR with 0 = Buffer

⟹ XOR with 1 = Not

Decision Making

* It is commonly represented in programming languages using the

(i) If statement

(ii) goto statements (label)

RISC-V includes two decision making instructions.

(if statement with a go to)

Conditional Branches

testing a value, based on the test result allows for a transfer of control to a new address in the program

beq rs1, rs2, L1

Branch if equal

label

Explanation: Go to the statement labeled "L1"; if the values in rs1=rs2

bne rs1, rs2, L1

Branch If not equal

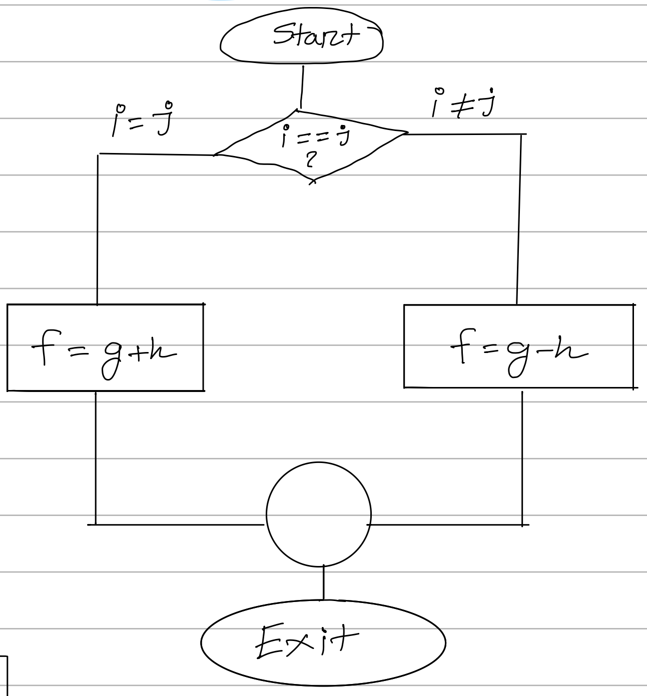
label

Explanation: Go to the statement labeled "L1"; if the values in rs1 != rs2

Given Code

```
If (i==j):
    f= g+h
else:
    f=g-h
```

Flow Chart:



RISC-V assembly code:

```
bne   x22, x23, Else
add   x19, x20, x21
beq   x0, x0, Exit
Else:
sub   x19, x20, x21
Exit:
```

(unconditional Branch) → beq

$f = x_{19}$

$g = x_{20}$

$h = x_{21}$

$i = x_{22}$

$j = x_{23}$

Conditional Jumps:

	Instruction	Syntax	operation
=	beq	beq rs1, rs2, L1	rs1 == rs2
!=	bne	bne rs1, rs2, L2	rs1 != rs2
<	blt	blt rs1, rs2, L3	rs1 < rs2
>=	bge	bge rs1, rs2, L4	rs1 >= rs2

Loop

```
while   (save [i] == k)
            i = i+1
            a = a+1
```

i = X22
k = X24
a = X23
Save, base = X25

```
Loop:
  ld   X7, 0[X25]          ✗ Static Wrong

  BNE  X7, X24, Exit

  Addi X22, X22, 1  }
                    }  Repeat ?
  Addi X23, X23, 1  }

  Beq  X0, X0, loop


Exit:
```

```
Loop:
  SLLi  X8, X22, 3
  Add   X8, X25, X8
  LD    X7, 0[X8]          ✓ Correct

  BNE  X7, X24, Exit

  Addi X22, X22, 1  }
                    }  Repeat ?
  Addi X23, X23, 1  }

  Beq  X0, X0, loop


Exit:
```