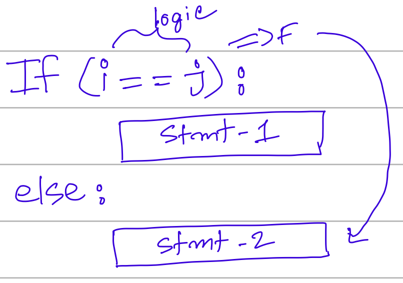
Decision Making

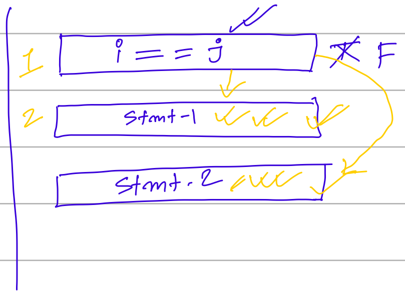
* It is commonly represented in programming languages using the
  - (i) If statement
  - (ii) goto statements (label)

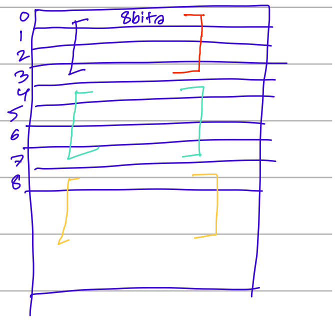
0 => add x20, x22, x21 => [32 bits]

4 => sub x21, x21, x19 => ["]

8 => slli x21, x21, 4 => ["]







RISC-V includes two decision making instructions.

(if statement with a go to)

beq rs1, rs2, L1

Branch if equal — label

Conditional Branches

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

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

bne rs1, rs2, L1

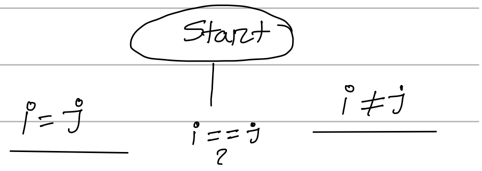
Branch If not equal — label

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

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

Flow Chart:



Given Code

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

```
IF (i==j):
    f=g+h
```

```
Beq (x22, x23) If → ✗   → False
Beq x0, x0, Exit
IF:
Add x19, x20, x21 ✗
Exit:
```

```
                         True
                   False
BNE (x22, x23), Exit
Add x19, x20, x21

Exit:
```

(i ≠ j)

RISC-V assembly code:

i ≠ j ⇒ False

```
bne   x22, x23, Else
add   x19, x20, x21      2
Beq   x0, x0, Exit=3          (unconditional Branch)
      ↓    ↓
      0    0
Else:                          JMP □
Sub   x19, x20, x21
Exit:
```

$f = x_{19}$

$g = x_{20}$

$h = x_{21}$

$i = x_{22}$

$j = x_{23}$

Loop

T → loop ✓

while (save[i] == k) → False ↳ Loop Break

a = a + 1 ✓

i = i + 1 ✓

i = X22

k = X24

a = X23

save => X25

Base

save [5 | 5 | 5 | 3 | ...]
0 1 2 3

save[1] => 8[X25]

1×8

save[0]

offset

```
Loop:
    ld    X9, 0[X25]      -1
    BNE   X9, X24, Exit   -2
    Addi  X23, X23, 1     -3
    Addi  X22, X22, 1     -4
    Beq   X0, X0, Loop
Exit:
```

WRONG

i/X22	k/X24	a/X23
∅	5	∅
1		1

Base: 0000

save[2] => index × 8

index × 2^3

save = [5 | 5 | 5 | 3 | ....]
0 1 2 3 ....

Offset

ld rd reg, imm [reg]

```
Loop:
    Slli  X10, X22, 3     ] offset
    Add   X10, X10, X25   ] offset + Base
    ld    X9, 0[X10]
    BNE   X9, X24, Exit   -2
    Addi  X23, X23, 1     -3
    Addi  X22, X22, 1     -4
    Beq   X0, X0, Loop
Exit:
```

i
0 OK
1
2

$1 \times 2^3 = 8$

offset

i[X22]	k[X24]	a[X23]	X10	X9
0	5	0	0	~~8~~
1		1	000∅	5
			8	
			0008	