⊛ Instructions for making decisions:

difference between a computer & a simple calculator is it's ability to take decisions.

decision based on → input data
decision based on → computed value

Decision in high level programming languages → if - else statement
Decision in high level programming languages → goto statement

In assembly MIPS language,

branch to a labeled instruction,

- unconditionally, jump

  j L1

- conditionally, (if a condition is true) [otherwise, continue sequentially],

b ranch
if
eq ual

~ beq reg1, reg2, L1

if reg1 == reg2: j L1

b ranch
if
n ot
e qual

~ bne reg1, reg2, L1

if reg1 != reg2: j L1

C | MIPS

```
if (i==j)
        f=g+h;
else
        f=g-h;
```

?

f ↔ $s0
g ↔ $s1
⋮
j ↔ $s4

f=g+h; ↔ add $s0, $s1, $s2

f=g-h; ↔ sub $s0, $s1, $s2

```
if(i==j)  ↔   beq $s3, $s4, L1
else      ↔   sub $s0, $s1, $s2
              j L2
          L1: add $s0, $s1, $s2
          L2:
```

Loop

C | MIPS

```
while (save[i]==K)
      i+=1;
```

?

i ↔ $t0,
K ↔ $s0,
save ↔ $s1

⊛ Finding save[i],

$t0*4 ($s1) → only immediate value

lw reg1, □ (reg2)

↕

0 ($s1+$t0*4)

```
L1: sll  $t1, $t0, 2      → $t1 = $t0*4
    add  $t1, $t1, $s1    → $t1 = $s1+$t0*4
    lw   $t2, 0($t1)
    bne  $t2, $s0, L2     → basic block
    addi $t0, $t0, 1
    j L1
L2:
```

ore Conditional Operations

S	et
	on
l	ess
t	han

```
- slt   regd, regs, regt

      if (regs < regt)
                  regd = 1;
      else
                  regd = 0;

- slti  reg1, reg2, constant

      if (reg2 < constant)
                  reg1 = 1;
      else
                  reg1 = 0;
```

- sltu, sltiu, unsigned version

→ No, blt, bge, (like, jl, jle, jg, jge, in x86)

↓ they require more work per instruction, extra clock cycle per instruction or stretched clock cycle

└→ smaller is faster

Then, how to implement them,

blt ⟷ slt + beq

Example, if ($t0 < $t1) branch to L

```
slt  $t2, $t0, $t1
bne  $t2, $zero, L     ⟷   beq $t2, 1, L
```

1 ↓ immediate

so, we used bne,