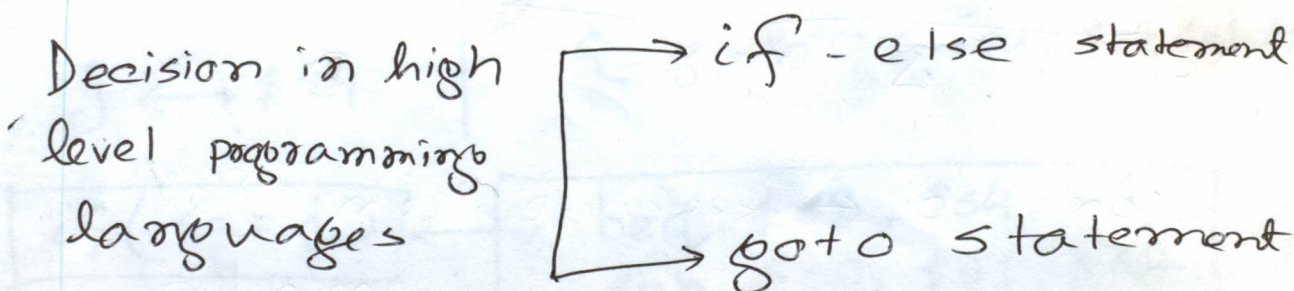
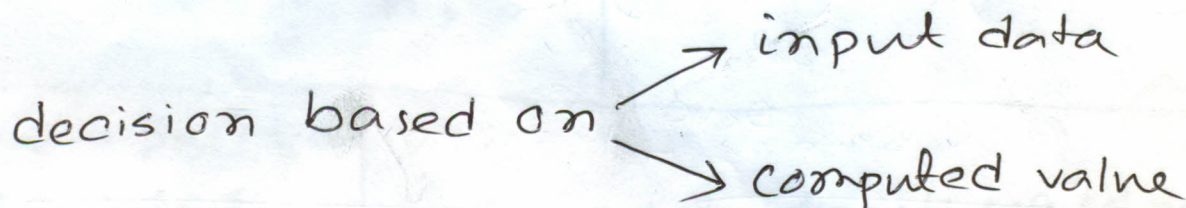


* Instructions for making decisions:

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



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
e	qual

~ beq reg1, reg2, L1
 if reg1 == reg2: j L1

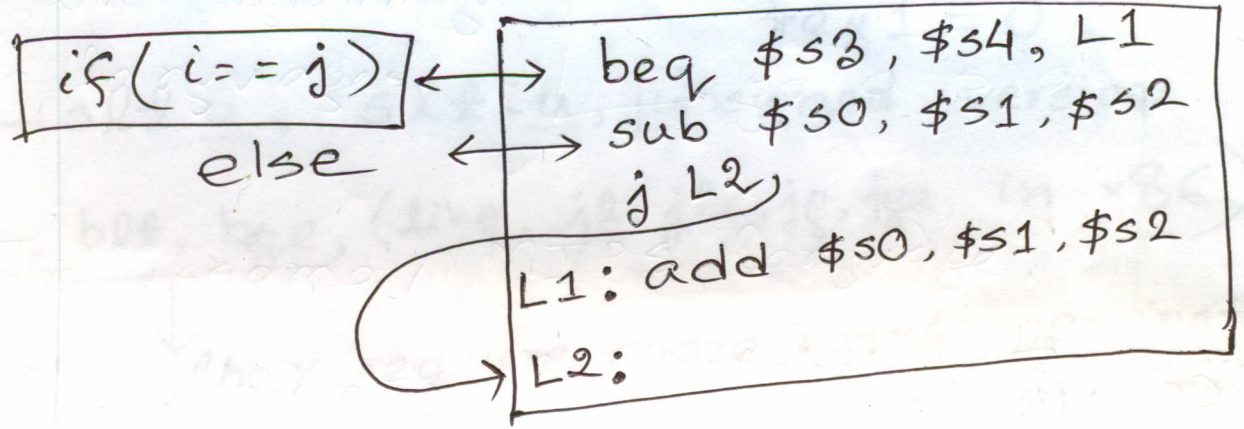
b	ranch
	if
n	ot
e	qual

~ 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

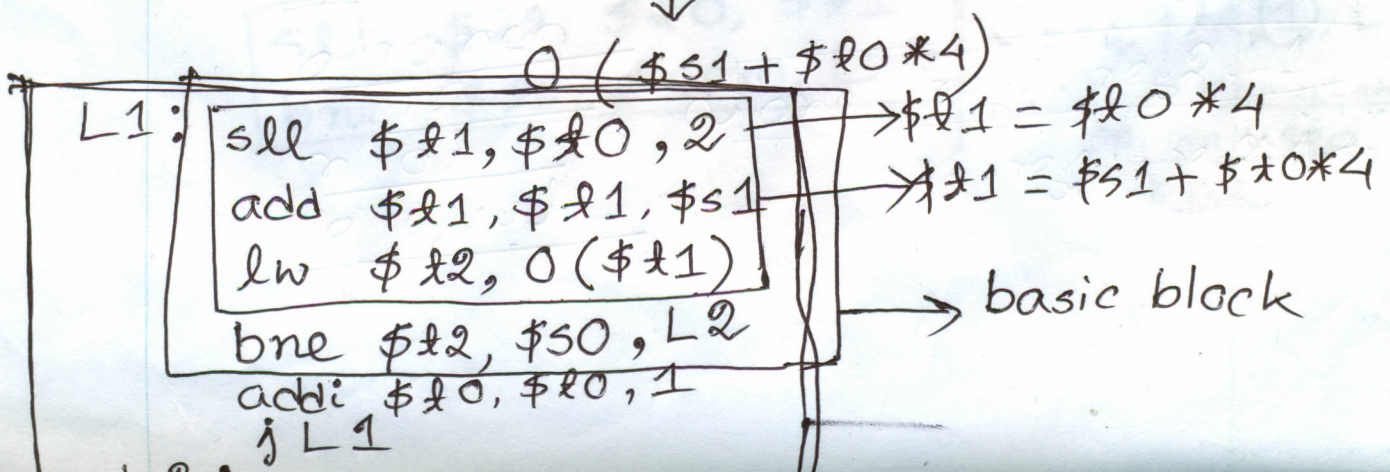
f = g + h ; ↔ add \$s0, \$s1, \$s2
f = g - h ; ↔ sub \$s0, \$s1, \$s2



C	Loop	MIPS
While (save[i] == k)	?	i ↔ \$t0, save ↔ \$s1 k ↔ \$s0,
i += 1;		

* Finding save[i],

$\$t0 * 4$ (\$s1) → only immediate value
lw reg1, [] (reg2)



More Conditional Operations

- `slt regd, regs, regt`

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

set	
on	
less	
than	

- `slti reg1, reg2, constant`

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

- `slt u, 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`
 immediate
 so, we used bne