Recap !

a is in %eax

b is in %edx

```
cmpl   %edx, %eax
setl   %al
```

set if lesser

if (a < b)

%al = 1 (True)

else

%al = 0 (False)

Same for the Zero flag.

ZF = 1 ⇒ The last operation yielded a zero

ZF = 0 ⇒ The last operation yielded something other than zero.

```
movl   $0x1, %eax
decl   %eax
```

If I check the Zero flag now, it will be 1

```
setz   %al
```

Scaled Index Operand

(, %eax , 4)

Imm (E_b , E_a , S)

Imm (, E_a , S)

can only be 1, 2, 4 or 8.

and so on.

Jump instruction

① Unconditional Jumps

- jmp label
- jmp * operand
- jmp * (operand)

---

② ~~Condition~~

Conditional Jumps

		Jump condition
① je Label (or) jz Label	ZF	If ZF is 1, Jump!
② jne (or) Label jnz	(~ZF)	If ZF is 0, Jump!
③ js Label	SF	
④ jns Label	~SF	

Signed Jump → jg, jge, jl, jle

Unsigned Jump → ja, jae, jb, jbe.

If Statements

C

```
if (a == b)
{
    a--; //
}
```

Assumptions

a, b are integers

a is in %eax

b is in %ebx

Assembly

```
        b       a
cmpl   %ebx , %eax
jne    skip_decr : //(~ZF)
decl   %eax
skip_decr :
```

If a == b

then a − b = 0

So ZF would be set to 1

So ~ZF = 0. (False)

Aside: Note

ZF is set (ZF = 1)

ZF is cleared (ZF = 0)

For loops

C

sum, i are integers.

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

Assembly

```
movl  $10, %ecx
movl  $0, %eax     ← sum is in %eax.
movl  $1, %edx     ← i is in %edx.
jmp   Label_1
Label_2:
addl  %edx, %eax   // sum = sum + i
incl  %edx         // i++
Label_1:
cmpl  %ecx, %edx   // 10, i
jle   Label_2      jump when i <= 10
```

jne 20 <main + 0x20>.

How do we the target that we jump to ?

__: 75 04 // jne 20

__:

__: Go to the offset (or address) of the next instruction and add 0x04 to it.

Relative Encoding

Encode the difference between the address of the target instruction and the address of the instruction immediately following the jump.

Can be 1, 2, or 4 bytes.

Absolute encoding

We give the exact 4-byte address to specify the target.

Relative > Absolute ?

① Compact — we may only need 1 or 2 bytes.

② Object code can be shifted to anywhere in memory without worrying about the address.

Relative encoding examples (Show code on screen)

① 8048 3ae : 75 04 jne 8048 3b4

+ 2

8048 3b0 :

+ 04

② 8048 28f : 74 05 je 8048296

8048291 :

05

0000 0101

③ 804 8357 : 72 e7 jb 8048340

804 8359 :

+ 19

1110 0111

16 | 25
1 - 9

-128 + 64 + 32 + 4 + 2 + 1

= -25_b = -19_{16}