JUMP, CONDITIONAL JUMP, UNCONDITIONAL JUMP

Instructions to print a one time

```
mov dl, 'a'
mov ah, 2
int 21h
```

. Program jump and run again, change its position.

OR.

```
mov ah, 1
int 21h.
mov dl, 3
mov ah, 4ch
int 21h.
```

mov ah, 1 int 21h. → take input.

mov dl, 3 → compare input with dl.

If you want to compare your input with dl if both equal it again take I/P from user.

If jump not equal it run downward.

→ It jump only according to condition, we write.
→ jump move from one place to another.
→ Program flow ↓ suddenly it went upward
you control program flow.

Jump is an instuction to control the program flow. (With condition or not).

Types of Jump :- (i) Unconditional jump.
(ii) Conditional jump.

① Unconditional jump :-

```
mov dl, 'a'
mov ah, 2
int 21h
```

send a to dl and print it
again want to jump, how to do it

→ We used <u>label</u> where we want to go.
(Need space, location, address to jump).

- jump to label without any condition (Unconditional jump).

<u>Syntax</u> :-

$\boxed{\text{jmp Label}}$

```
L1:
mov dl, 'a'
mov ah, 2
int 21h
jmp L1
```

- Label present in any place in program, it jump to that label.

(ii) <u>Conditional jump</u>

jump to label when condition occur.

Syntax :-

Opcode Label (opcode means variety of conditions, =, >, <, <=, etc.

1) for example

```
L1 :
mov ah, 1
int 21h
mov dl, 3
JE L1
```

For this program, we take input & compare if both equal it jump. So the condition is equal to JE (jump equal to). and write label name.

Problem, Compare to be done before jump, so the input is equal or not. We have to compare input & dl in this case. Without comparision we can't decide.

- After comparision result we decide next step.
- Compare is needed before jump.
- Compare means we have to subtraction method).

<u>Compare</u> :- Subtracts operand1 from operand2 but does not store the result; only changes the flags.

- In Subtraction If result is zero, handled by ZF, <u>ZF is 1 if result is zero.</u>
- In subtraction two operands, one is sub from other, result not stored, destination value is changed.
- In compare we don't want to store the result.
- Flag register change if result is zero, JE check it and jump according to it. <u>changing of flag we jump according to it.</u>

Compare Syntax :-

```
Cmp reg, reg
Cmp reg, constant
Cmp reg, [memory add.]
```

```
cmp dl, al
cmp dl, '3'
cmp dl, [si]
```

```
L1:
mov ah, 1
int 21h
mov dl, 3
Cmp al, dl
JE L1
```

jump if ZF = 1

- input save in al, 3 is store in dl
- If input is 3, compare value of al & dl, both 3, subtract resullt is zero, flag register is On. It then jump, if both equal it go to flag register ZF

jump IF ZF = 1

- Backend ~~this~~ all opcodes check bits of flag register. (opcode of conditional jump)
- In JE check zero flag.

Example of 5 comes in al & 3 is in dl.

AL = 5 Not performed jump, Go downward
DL = 3 JE L1 line is ignored.

Opcodes (Conditions a/c to nature of program)

1. JE, JZ jump if equal, jump if zero.
2. JNE, JNZ jump if not equal, jump if not zero.

(When al = 5, dl = 3) work here.

③ JL , JB — jump if less, Jump if below.

④ JLE, JBE — jump if less or equal, jump if below or equal.

⑤. JG, JA — jump if greater , jump if above.

⑥ JGE, JAE — jump if greater or equal, jump if above or equal.

(Optional)

- If you want to test flag register what happen when 1 is in on Carry flag.
- Direct jump.

JC, JP, JA, JS, JT, JI, JD, JO, JZ

- If subtraction performed, what parity add on result, (9)
  if write jump parity it jump;
- Opcode learned.

LAB

Program to print the input number is equal or not to given number in program.

→ One number is placed in register, 2nd number input from user, if equal print message equal else not equal.

```
dosseg
.model small
.stack 100h
.data
msg1 db 'number is equal$'
msg1 db 'number isot equal$'
.code
main proc
mov ax, @data
mov ds, ax
mov dl, '3'
mov ah, 1
int 21h
```

} to access variable fast, heap memory initialize.

mov dl, '3' → send value to dl

mov ah, 1 → when input is taken, its ASCII code is there. So we use '3', In this way dl get ascii code

```
cmp al, dl
je L1
mov dx, offset msg2
mov ah, 9
int 21h
mov ah, 4ch
int 21h
L1:
mov dx, offset msg1
mov ah, 9
int 21h
mov ah, 4ch
int 21h
main endp
end main
```

→ compare dl and al

→ If not equal this line ignored (je L1), no jump go downward.

Print msg2,

to exit from this section and avoid autorun of L1, we write

```
mov ah, 4ch
int 21h
```

So it exit on this point.

If it equals

Print that number is equal & exit.

] program end.