



**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.
```

→ take input.

→ 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. (2)

→ jump move from one place to another.

→ program flow ↓ suddenly it went up and you control program flow.

Jump is an instruction 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

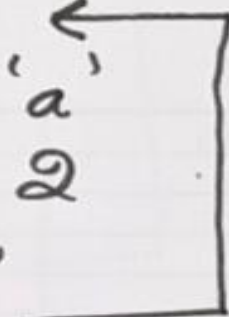
→ We used label where we want to go.
(Need space, location, address to jump).

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

Syntax:-

`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.

② Conditional jump

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 equals 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 comparison we can't decide.

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

Compare :- Subtract operand1 from operand2 but does not store the result; only changes the flags.

→ In Subtraction If result is Zero, handled by ZF,
ZF is 1 if result is zero.

- In subtraction: two operands, one is sub from other, result not stored, destination value is changed.
- On compare we don't want to store the result.
- Flag register change if result is Zero, JE check it and jump according to it. Changing of flag we jump according to it.

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 result is zero, flag register is On. It then jump, if both equal it go to flag register ZF

jump IF ZF = 1

- Backend ~~the code~~ all opcodes check Bits of flag register.
(opcode of conditional jump)
- JE check zero flag.

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

Al = 5
DI = 3

Not performed jump, Go downward
JE 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

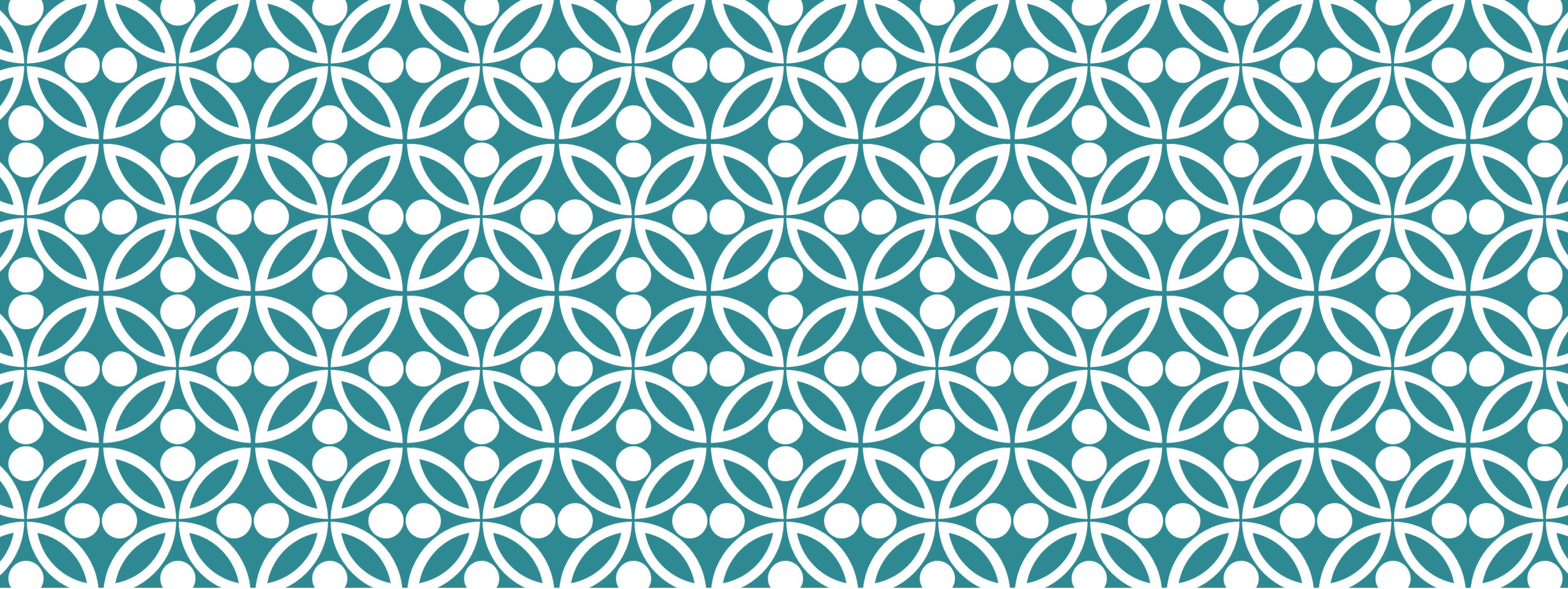
jump if greater or equal, jump if above or equal.

Optional

- If you want to test flag register what happens 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$'  
msg1 db 'number is not equal$'
```

```
  .code
```

```
main proc
```

```
mov ax, @data
```

```
mov ds, ax
```

```
mov dl, '3'
```

```
mov ah, 1
```

```
int 21h
```

} to access variable fast,
heap memory initialize.

→ send value to dl

→ 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 (11)

→ 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 will
mov ah, 4ch
int 21h
So it exit on this point.

if it equals
Print that slumbee is equal
& exit.

] program end.