Usage of AND, OR, XOR, NOT

AND:

X	Y	X AND Y
0	0	0
1	0	0
0	1	0
1	1	1

USE : to chick any bit by change (0 to 1) or (1 to 0)

EX :

AX = F0F5 H, BX = 01FC H

AX	1	1	1	1	0	0	0	0	1	1	1	1	0	1	0	1
BX	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0
AX AND BX	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0

Example:

```
MOV AL, 'a'       ; AL = 01100001b
AND AL, 11011111b  ; AL = 01000001b  ('A')
RET
```

OR :

X	Y	X OR Y
0	0	0
1	0	1
0	1	1
1	1	1

USE : to select any bit and not change its value

EX :

AX = F0F5 H, BX = 01FC H

AX	1	1	1	1	0	0	0	0	1	1	1	1	0	1	0	1
BX	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0
AX OR BX	1	1	1	1	0	0	0	1	1	1	1	1	1	1	0	1

Example:

```
MOV AL, 'A'       ; AL = 01000001b
OR AL, 00100000b  ; AL = 01100001b  ('a')
RET
```

XOR :

X	Y	X XOR Y
0	0	0
1	0	1
0	1	1
1	1	0

USE : to exchange bit value (0 to 1) or (1 to 0)

EX : AX = F0F5 H, BX = 01FC H

AX	1	1	1	1	0	0	0	0	1	1	1	1	0	1	0	1
BX	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0
AX XOR BX	1	1	1	1	0	0	0	1	0	0	0	0	1	0	0	1

NOT

NOT D

X	NOT X
0	1
1	0

XCHG instruction

XCHG reg., reg

XCHG reg, [add

XCHG [add.], reg

EX: write an 8086 assembly language program to exchange two byte number stored in memory location started at 3000 with two byte number stored in memory location started at 4000 .

Answer:

Mov AX, [3000]

XCHG AX, [4000]

Mov [3000], AX

Q) Execute this four steps using XCHG instruction.

1. Load 12cdh into BX

2.copy the lower 8-bit from BX to AH

3. Load 10110111b to address memory contain in the BX reg.

4. Replace the data between BH and CL

Program Flow Control

Controlling the program flow is a very important thing, this is where your program can make decisions according to certain conditions.

- **Unconditional Jumps**

  The basic instruction that transfers control to another point in the program is **JMP**.

  The basic syntax of **JMP** instruction:

  JMP label

  To declare a *label* in your program, just type its name and add "**:**" to the end, label can be any character combination but it cannot start with a number, for example here are 3 legal label definitions:

```
label1:
label2:
a:
```

  Label can be declared on a separate line or before any other instruction, for example:

```
x1:
MOV AX, 1

x2: MOV AX, 2
```

Here is an example of **JMP** instruction:

```
ORG    100h

MOV    AX, 5          ; set AX to 5.
MOV    BX, 2          ; set BX to 2.

JMP    calc           ; go to 'calc'.

back:  JMP stop       ; go to 'stop'.

calc:
ADD    AX, BX         ; add BX to AX.
JMP    back           ; go 'back'.

stop:

RET                   ; return to operating system.

END                   ; directive to stop the compiler.
```

- **Short Conditional Jumps**

Unlike **JMP** instruction that does an unconditional jump, there are instructions that do a conditional jumps (jump only when some conditions are in act). These instructions are divided in three groups, first group just test single flag, second compares numbers as signed, and third compares numbers as unsigned.

Jump instructions that test single flag

Instruction	Description	Condition	Opposite Instruction
JZ , JE	Jump if Zero (Equal).	ZF = 1	JNZ, JNE
JC , JB, JNAE	Jump if Carry (Below, Not Above Equal).	CF = 1	JNC, JNB, JAE
JS	Jump if Sign.	SF = 1	JNS

JO	Jump if Overflow.	OF = 1	JNO
JPE, JP	Jump if Parity Even.	PF = 1	JPO
JNZ , JNE	Jump if Not Zero (Not Equal).	ZF = 0	JZ, JE
JNC , JNB, JAE	Jump if Not Carry (Not Below, Above Equal).	CF = 0	JC, JB, JNAE
JNS	Jump if Not Sign.	SF = 0	JS
JNO	Jump if Not Overflow.	OF = 0	JO
JPO, JNP	Jump if Parity Odd (No Parity).	PF = 0	JPE, JP

Jump instructions for signed numbers

Instruction	Description	Condition	Opposite Instruction
JE , JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE , JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JG , JNLE	Jump if Greater (>). Jump if Not Less or Equal (not <=).	ZF = 0 and SF = OF	JNG, JLE
JL , JNGE	Jump if Less (<). Jump if Not Greater or Equal (not >=).	SF <> OF	JNL, JGE
JGE , JNL	Jump if Greater or Equal (>=). Jump if Not Less (not <).	SF = OF	JNGE, JL
JLE , JNG	Jump if Less or Equal (<=). Jump if Not Greater (not >).	ZF = 1 or SF <> OF	JNLE, JG

<> - sign means not equal.

Jump instructions for unsigned numbers

Instruction	Description	Condition	Opposite Instruction
JE , JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE , JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JA , JNBE	Jump if Above (>). Jump if Not Below or Equal (not <=).	CF = 0 and ZF = 0	JNA, JBE
JB , JNAE, JC	Jump if Below (<). Jump if Not Above or Equal (not >=). Jump if Carry.	CF = 1	JNB, JAE, JNC
JAE , JNB, JNC	Jump if Above or Equal (>=). Jump if Not Below (not <). Jump if Not Carry.	CF = 0	JNAE, JB
JBE , JNA	Jump if Below or Equal (<=). Jump if Not Above (not >).	CF = 1 or ZF = 1	JNBE, JA

Generally, when it is required to compare numeric values **CMP**instruction is used (it does the same as **SUB** (subtract) instruction, but does not keep the result, just affects the flags).

The logic is very simple, for example:
it's required to compare 5 and 2,
5 - 2 = 3
the result is not zero (Zero Flag is set to 0).

Another example:
it's required to compare 7 and 7,
7 - 7 = 0
the result is zero! (Zero Flag is set to 1 and **JZ** or **JE** will do

the jump).

Here is an example of **CMP** instruction and conditional jump:

```
include emu8086.inc

ORG    100h

MOV    AL, 25     ; set AL to 25.
MOV    BL, 10     ; set BL to 10.

CMP    AL, BL     ; compare AL - BL.

JE     equal      ; jump if AL = BL (ZF = 1).

PUTC   'N'        ; if it gets here, then AL <> BL,
JMP    stop       ; so print 'N', and jump to stop.

equal:            ; if gets here,
PUTC   'Y'        ; then AL = BL, so print 'Y'.

stop:

RET               ; gets here no matter what.

END
```

example:

```
include emu8086.inc

ORG    100h

MOV    AL, 25     ; set AL to 25.
MOV    BL, 10     ; set BL to 10.

CMP    AL, BL     ; compare AL - BL.


JNE    not_equal  ; jump if AL <> BL (ZF = 0).
JMP    equal
not_equal:


; let's assume that here we
; have a code that is assembled
; to more then 127 bytes...
```

```
PUTC   'N'       ; if it gets here, then AL <> BL,
JMP    stop      ; so print 'N', and jump to stop.

equal:           ; if gets here,
PUTC   'Y'       ; then AL = BL, so print 'Y'.

stop:

RET              ; gets here no matter what.

END
```

example:

```
ORG    100h

; unconditional jump forward:
; skip over next 2 bytes,
JMP $2
a DB 3    ; 1 byte.
b DB 4    ; 1 byte.

; JCC jump back 7 bytes:
; (JMP takes 2 bytes itself)
MOV BL,9
DEC BL      ; 2 bytes.
CMP BL, 0   ; 3 bytes.
JNE $-7

RET

END
```