Loop (jmp, loop)

Infinite Finite

M, Immediate
R, Immediate
M, R
R, M

Unconditional (Loop) Conditional loop (jmp)

```
mov ecx, 10
L1:
  ;; ecx = ecx - 1
  if (ecx == 0)
    break;
  return
loop L1
```

```
L1:
  cmp
  jmp L1
```

```
if (x == 0)
  cout << "zero"
else
  cout << "A number"
```

cmp __, __
D S

```
.data
x byte 10
.code
  cmp __, __
  jz LTrue
```

Date:
M T W T F S S

Cmp

* Ye cmp instruction updates the values of Flag Register, not the value of Destination operand.

```
.data
    num1    DWORD ?
    num2    DWORD ?
    msg1    BYTE  "Message", 0
.code
    mov edx, OFFSET msg1   ; String ka offset hamesha edx ko dein
    call ReadString
```

```
.data
    x byte 10
msg1 Byte  "Print zero", 0
msg2 Byte  "A number", 0
.code
    cmp x, 0 ;
    jz LTrue
    mov edx, offset msg2:
    call Write String
    jmp L1
LTrue:
    mov edx, offset msg1
    call Write String
L1:
```

```
.data
    x byte ?
msg1 BYTE "How many times you want to print?"
.code
    mov eax, 0
    mov edx, offset msg1
    call Write String
    call Read Hex ; eax = 10
    mov edx, 0
L1: cmp edx, eax
    jnz LTrue
    jmp Lexit
LTrue: inc edx, edx=1 (i++)
    jmp L1
Lexit:
    Iterator hai,
```

Date
M T W T F S S

```
mov ecx, 0
L1:  mov edx, msg1
     call WriteString
loop L1 ⟶ if (ecx == 0)
            break;
          else continue;
```

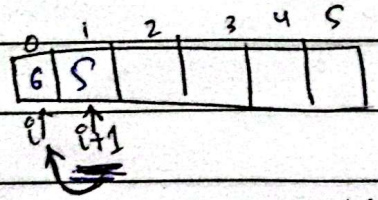
runs infinite times.

```
.data   loop1 DWORD ?        count DWORD 1
        x byte ?             msg1 BYTE "*", 0
.code   main PROC
        mov eax, 0
        call ReadHex      ; eax = 4 ← input
        mov ecx, eax      ; ecx = 4
```

```
*
* *
* * *
* * * *
```

```
L1:
        mov loop1, ecx        loop1 ← 4
        mov ecx, count        count = 1
L2:     mov edx, offset msg1
        call WriteString
        loop L2
        inc count
        mov ecx, loop1
        call Crlf ; moves to next line
        loop L1
        Crlf
        main endp
        End main
```

Q.



x ← user input

```
mov bl, [arr + 1]      ; bl = 5
```

5 + 6 + 5 + 5 + 5 + 5 + 5

```
bl → ecx
ecx--;  add [arr], bl
```

Date:
M T W T F S S

```
arr  BYTE 6h,5h,10h, 20h
mov  bl, [arr + 1]
mov  ecx , bl
L1:
  add  [arr], bl
  mov  bh,
```

```
arr    BYTE  5,3,4,2.
result BYTE  ?
mov al, [arr]
mov bl, [arr+1]
xor c1,c1      } to clear c1 [counter], bh,
xor bh, bh     } and dl
xor dl, dl     }
L1:
  add dl, al
  inc c1
  cmp c1, bl
  jnz loop L1
  mov [result], dl
```

Unsigned (+ve) Signed (+ve/-ve)

```
if (x == y)
  Print ("x == y")
elseif (x > y)
  print ("x > y")
Else  print ("x < y")
```

ZF = 1 destination == source.

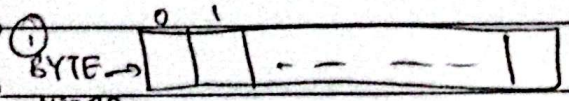
CMP RESULTS	ZF	CF
destination < source	0	1
destination > source	0	0
destination == source	1	0

```
.data
msg1  BYTE "Destination is equal to source",0
msg2  BYTE "Destination is ~~greater~~ less than source",0
msg3  BYTE "~~Destination~~ source is greater than ~~Destination~~ source",0

main  PROC
CMP  x, eax
JE   Equal
JNE  x, eax
```

• code

```
main PROC
    cmp  x, eax
→   je   Lequal
    cmp  x, eax
→   jc   Lless
    jmp  LGreater
Lequal:  mov edx, OFFSET msg1
         jmp  Lprint
Lless:   mov edx, OFFSET msg2
         jmp  Lprint
LGreater: mov edx, OFFSET msg3
Lprint:  Call WriteString
```



② x ← input

③ Make sure x > 0 x must be greater than zero.

④ if (x == arr[x]) x = is the index

count += 1

else if (x < arr[x])

count < += 1

else

count >+= 1

```
Cmp  x, eax
jc   Lless
Lless: mov edx, OFFSET msg1   → "x<0"
       jmp  Lprint
Lprint: Call writestring
```

⑤ Print

Date
M T W T F S S

```
.data
    x SDW -9
.code
    main PROC
    cmp x, eax
    jz LEqual
    cmp x, eax
sign ← jsf LGreater        (Cond1)
flag   jmp LCond2.
    LGreater (Cond1) cmp x, eax
        jof LGreater
        jmp Lless
```

```
    jmp LCond2
LCond2  cmp x, eax
        jof LGreater
        jmp Lless
```

Signed			Unsigned		
SF ≠ OF	jL/jLE	D < S	0	1	jB/jBE
SF = OF	jG/jGE	D > S	0	0	jAE/jA ~~jNBE~~ jump above ~~jump if not below or equal~~
ZF = 1	jz/je	D = S	1	0	je/jne
	jnz/jne				

Mid II mai sign flag ki basis mai comparision hoge. DONOT use jB/jA etc in Mid II ---- allowed in final.

jCXZ → if CX = 0.

```
.data
    x DW 10
.code
    mov ecx, 10
L1: cmp x, eax

    jmp L1
```

```
    mov cx, 5
L1:
    cmp cx, 0
    jcxz Loutofloop
    dec cx
    jmp L1
Loutofloop:
```

-1 Sub ax, 1

Date: M T W T F S S

Unsigned 8 bit 0 to 255

Signed 8 bit range -128 to +127

	Unsigned	Signed
Min	0000 0000 = 0	1000 0000 = -128
Max	1111 1111 = 255	0111 1111 = +127

2 → 128 64 32 16 8 4 2 1
0 0 0 0 0 0 1 0 → (02)h

-2
1 1 1 1 1 1 0 1
+ 1
(-2) 1111 1110

((FE)h

F is telling k ye negative hai.

0 0 0 0 0 0 1 0
+ 1 1 1 1 1 1 1 0
1 0 0 0 0 0 0 0

carry.

02 - 03 → FF (-1)

7F + 01 → 80 [exceeds range]

mov ax, 7FF0h

add al, 10h → CF = 1, SF = 0, OF = 0, ZF = 1

add ah, 10h → 0 1 0 0

add ax, 02h → 0 1 0 0

7FF0h
ah al

7FF0
+ 10
7F00

CF →

1111 0000
0001 0000
0000 0000 ✓

-128 + 64 + 32 + 16 + 16

7F
+ 10
8F

0111 1111 → +ve
+ 0001 0000 → +ve
1000 1111 → -ve

7FF0
+ 02
7FF2

1000 1111 0000 0000
+ 0000 0000 0000 0010
1000 1111 0000 0010
8F02

Cmp, and Subtract Difference :-

→ Both operations affects flag registers.

→ Cmp → store nhi krte subtract → destination is updated.

* Mov instruction is the only instruction which doesn't change flag registers.

tx:-

Date: M T W T F S S

How to change a particular bit?

```
mov al,  0 1 1 0   1011
and al,  1 1 0 1   1111
         ---------------
         0 1 0 0   1011
             ↑
           "off"
```

```
mov al,  0 1 1 0   1 0 1 1
or  al,  1 0 0 1   0 0 0 0
         ---------------
         1 1 1 1   1 0 1 1
```

XOR		
1	1	0
0	0	0
1	0	1
0	1	1

```
mov al, 0110 1011
not al
result :- 1001 0100
```

test :- test instruction → indirectly and hi krte lekin destination ko update nhi krte but flags update hote.

```
mov al, 0110 1011
test al, 1010 0011
```

indirectly "and". ↑

(!update), flags update

sub (al), 5h	cmp (al), 5
↑	↑
Destination update (flag registers update).	Destination isnot update but flag register update
AND al, 5	test al, 5
Destination & flag registers	only flag register will update.

Q.)

```
mov al, (1000 0000)h
shl al, 2
```

128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0

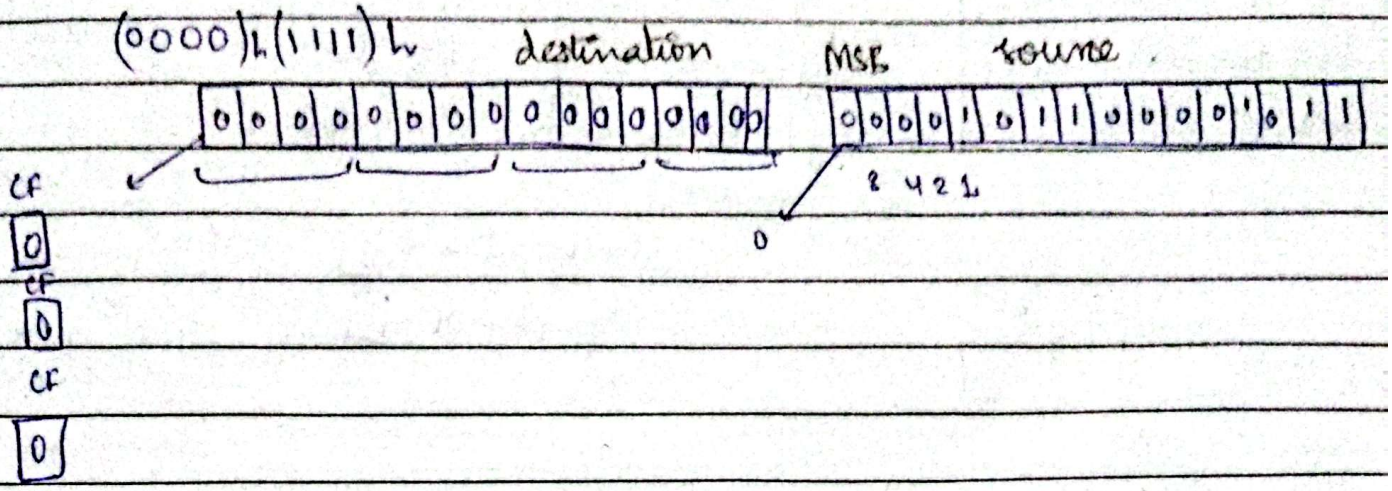
operation valid hai $128 * 2^2 = 128 \times 4 = \boxed{512}$

but value 512 nhi aaegi.

Shift & Rotate Instructions :-

Date: M T W T F S S

SHLD bx, cx (3) → shifts 3 times shift left → MSB se copy horaha



Difference b/w MOV & SHLD.

restriction → same bits, SHLD dest, source

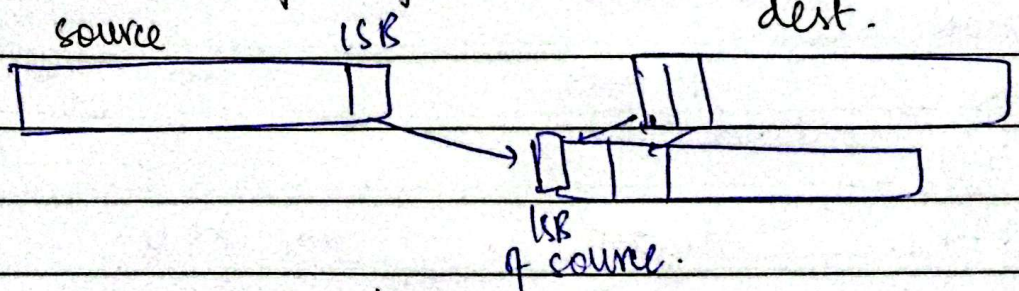
mov dest, source
16 bit 16 bit
8bit 8bit
must be same
↳ to move all bits

↳ to move particular bits only.
↳ to invert

SHLD reg16, reg16, CL/imm8
these two are same
↳ this can be different

SHRD dest, source, count

LSB se copy hoga



Q. to copy invert bits.
use SHRD

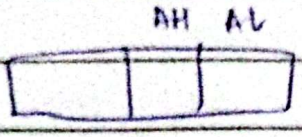
	Multiplicand	Multiplier	Product	
8bit ←	AL	reg/mem8	AX	
16bit ←	AX	reg/mem16	DX : AX	DX : AX MSB LSB
32bit ←	EAX	reg/mem32	EDX : EAX	AX will have LSB DX will have MSB

EBX
ECX | mem32
EDX

unsigned → CF = 1
signed → OF = 1

(0 → 255)

use span answer paraphrase



learn table

EAX

AL	R8 bit/mem8	AX
AX	R 16 bit/mem16	DX : AX
EAX	R32 bit/mem32	EDX : EAX

Overflow
signed OF = 1
unsigned CF = 1

cmp

left MULTIPLICATION (*) Right

left DIVISION (/) Right

IMUL → for signed

MUL → for unsigned.

Remainder	Quotient	Divisor	Dividend
AH	AL	Reg/mem8	AX
DX	AX	Reg/mem16	DX : AX
EDX	EAX	Reg/mem32	EDX : EAX

Modulus ← DIVISION → Quotient

DIV IDIV

```
mov ax, 0083h    ; dividend
mov bl, 2        ; divisor
div bl           ; AL = 41h , AH = 01h
```

```
mov dx,0
mov ax,8003h
mov cx, 100h
div cx
```

agar idhar ah hote hain toe
cdw / cwd hote hain.
cword to dword.

Date: M T W T F S S

(al) FF

cbw / FF FF   cbw → extend byte to word.

mov bl, +5
idiv bl

mov eax, -6101563

cdq. dword to quad
sign extend

al = unsigned 0 — 255

al = 1111 1111
bl = 1111 1111

mov al, 255 d   1111 1111 ← 255
mov bl, 255d   + 1111 1111 ← 255 + 510
add al, bl   CF [1] 1111 1110
Dest. Source   this unit 510

So use adc   ADC dl, 0
                     0000 0000
                     +   1
                     0000 0001 DL

mov edx, 0
mov dl, 0 — mov eax, 0
mov al, 0FFh
add al, 0FFh
~~adc dl, 0~~
rcr edx, 1
shld edx, eax, 8   shld dl, al, 8

```
mov edx, 7
mov eax, 1                    00 00 00 01
sub eax, 2
sbb edx, 0
```

```
  1 1 1 1
  0 0 0 0 0 1
- 0 0 0 0 0 2
-------------
  0 1 1 1 1 1        FFFFFF
```

Signed → OF is set if value exceeds register
Unsigned → CF is set if value exceeds register

```
mov ax, 10h
mov bx, 20h
mul bx
jc valuelarge,
jmp Print:
valuelarge:
    mov ecx, 0
    mov cx, ax
    mov ax, dx
    call WriteHex
    mov ax, cx
    call WriteHex
Print:
    Call WriteHex
```

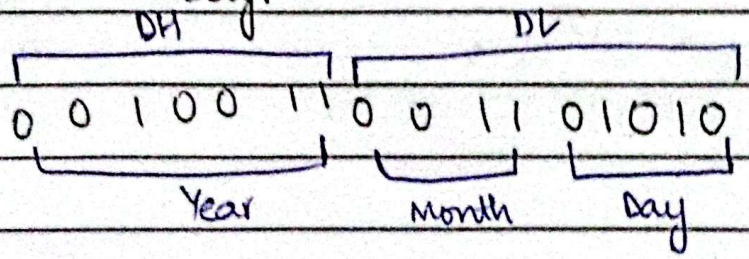
move it to CX

DX : AX

Q) EAX * 36 = EAX * (2^5 + 2^2)
If not directly Power of 2, = EAX * (32 + 4)
= (EAX * 32) + (EAX * 4)

01111

Q) Year - Month - Days



* Original register ko change nhi krna.

Date: M T W T F S S

```
DL      0 1 1 0 1 0 1 0
and     0 0 0 1 1 1 1 1
AL =    0 0 0 0 1 0 1 0
                8 4 2 1
```

Day = 10

```
0 0 1 1 0 1 0 1 0
AL = 0 0 1 1 0 1 0 1
```

Month

```
and  1 1 1 1 0 0 0 0
     0 0 1 1 0 0 0 0
       0 0 1 1 0 0 0
         0 0 1 1 0 0
           0 0 1 1 0
```

Month → 0 0 1 1

DH/AH

```
0 0 1 0 0 1 1 0
    Year
0 0 1 0 0 1 1 1
```

16 8 Year 4 2 1 → 19

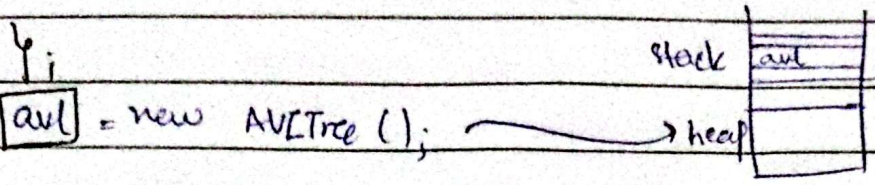
```
mov al, 0
mov al, dl
and al, 00011111 ; al = Day
mov day, al
mov ax, 0
mov ax, dx
shr ax, 1
and al, 1111
shr al, #4
mov month, al
mov ah, 0
mov ah, dh
shr ah, 1
mov year, ah
add year, baseyear.
```

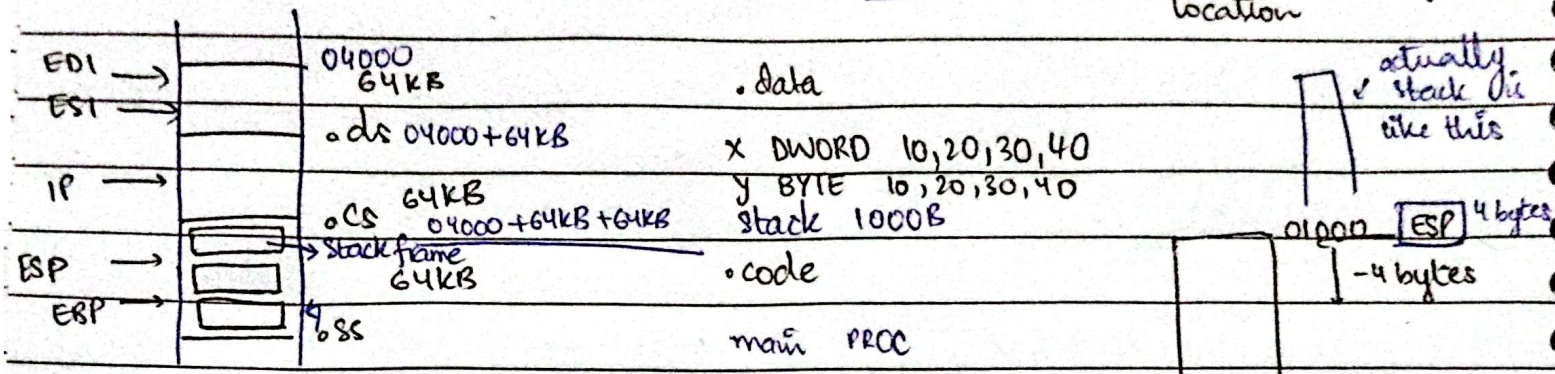
2000
+ 19
2019

Add 2000 in 19 to get year.

Class AVLTree {

Date: M T W T F S S





Agar BYTE hai { movzx eax, [y+0] ; push eax } ESP

toe use MOVZX

push → -4 bytes

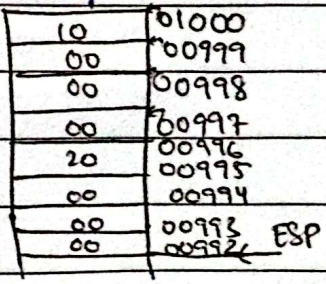
In case of DWORD { mov eax, [x+0] ; push eax } pop → +4 bytes

SP

Push Reg/Mem 32/16 → push → -2 bytes

Push Imm 32/16 → pop → +2 bytes

Pop Reg/Mem 32/16



main PROC

```
0013A9   mov eax, 10h        001309 Add PROC
0013B0   mov ebx, 20h        0013C4 add ecx, eax
0013B6   call Add (001309)   0013CB add ecx, ebx
0013B7   mov eax, ecx        (IP)
(IP)     call WriteHex       0013D0 ret
                             (IP) 001400 Add Endp
                             001401 End main
```