

# 80x86 MICROPROCESSOR

## Instructions

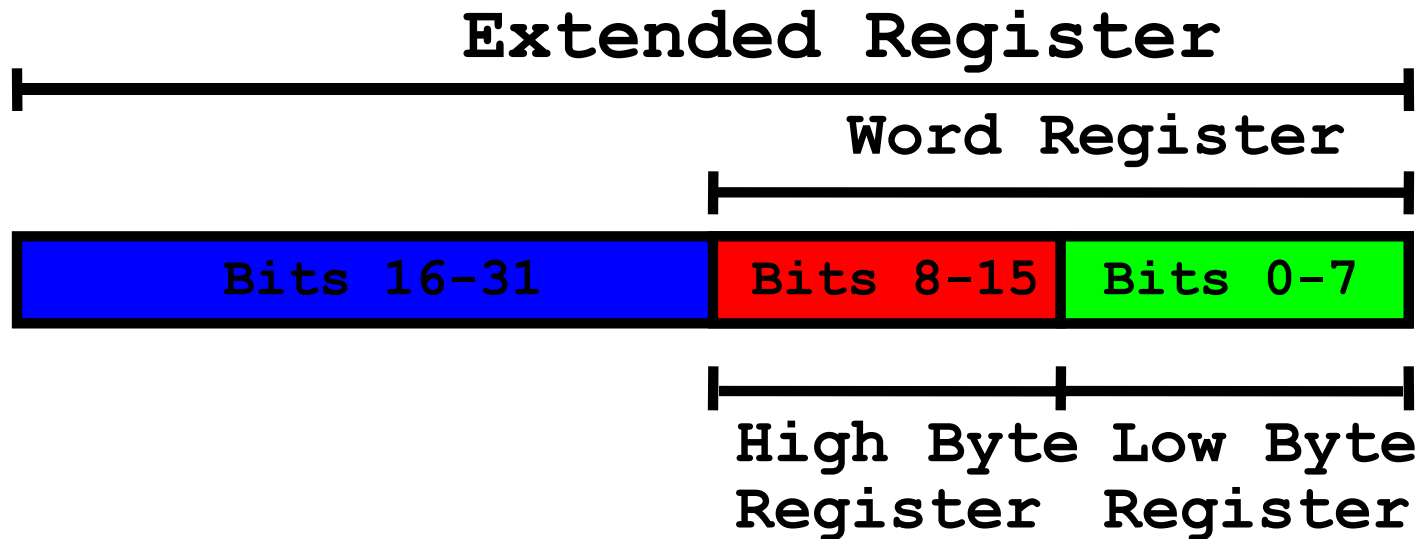
# Inside The 8088/8086...*registers*

- **Registers**
  - Verileri geçici olarak tutar

| AX              |            |
|-----------------|------------|
| 16-bit register |            |
| AH              | AL         |
| 8-bit reg.      | 8-bit reg. |

| Category    | Bits | Register Names                            |
|-------------|------|-------------------------------------------|
| General     | 16   | AX, BX, CX, DX                            |
|             | 8    | AH, AL, BH, BL, CH, CL, DH, DL            |
| Pointer     | 16   | SP (stack pointer), BP (base pointer)     |
| Index       | 16   | SI (source index), DI (destination index) |
| Segment     | 16   | CS (code segment), DS (data segment)      |
|             |      | SS (stack segment), ES (extra segment)    |
| Instruction | 16   | IP (instruction pointer)                  |
| Flag        | 16   | FR (flag register)                        |

# Anatomy of a Register



# General Registers

| 32 bit Registers |     | 16 bit Registers |    | 8 bit Registers |    |
|------------------|-----|------------------|----|-----------------|----|
| EAX              | EBP | AX               | BP | AH              | AL |
| EBX              | ESI | BX               | SI | BH              | BL |
| ECX              | EDI | CX               | DI | CH              | CL |
| EDX              | ESP | DX               | SP | DH              | DL |
| Bits 16-31       |     | Bits 8-15        |    | Bits 0-7        |    |

# Registers

## 6 Category

- General
- Pointer
- Index
- Segment
- Instruction
- Flag

|        |  |       |               |    |
|--------|--|-------|---------------|----|
| EAX    |  | AH    | <del>AX</del> | AL |
| EBX    |  | BH    | <del>BX</del> | BL |
| ECX    |  | CH    | <del>CX</del> | CL |
| EDX    |  | DH    | <del>DX</del> | DL |
| EBP    |  | BP    |               |    |
| ESI    |  | SI    |               |    |
| EDI    |  | DI    |               |    |
| ESP    |  | SP    |               |    |
| EIP    |  | IP    |               |    |
| EFLAGS |  | FLAGS |               |    |

|    |
|----|
| CS |
| DS |
| ES |
| SS |
| FS |
| GS |

# Register names

- |                       |                   |
|-----------------------|-------------------|
| ■ Accumulator         | Segment registers |
| ■ Base index          | ■ Code            |
| ■ Count               | ■ Data            |
| ■ Data                | ■ Extra           |
| ■ Stack Pointer       | ■ Stack           |
| ■ Base Pointer        |                   |
| ■ Destination index   |                   |
| ■ Source index        |                   |
| ■ Instruction Pointer |                   |
| ■ Flags               |                   |

## *Assembly Instruction format*

General format

mnemonic

operand(s)

;comments

MOV            destination,source    ;copy source operand to destination

Example:

MOV    DX,CX

Example 2:

MOV    CL,55H

MOV    DL,CL

MOV    AH,DL

MOV    AL,AH

MOV    BH,CL

MOV    CH,BH

|    |    |
|----|----|
| AH | AL |
| BH | BL |
| CH | CL |
| DH | DL |

# Instruction Set Classification

## ❑ Transfer

- Move

## ❑ Arithmetic

- Add / Subtract
- Mul / Div, etc.

## ❑ Logic & Shift

- AND/OR/NOT/XOR
- Rotate/Shift

## ❑ Control

- Jump
- Call / Return, etc.



# Data transfer : Move

## □ MOV Dest, Src

|                |            |
|----------------|------------|
| – MOV reg, reg | reg <- reg |
| – MOV reg, mem | reg <- mem |
| – MOV mem, reg | mem <- reg |
| – MOV reg, imm | reg <- imm |
| – MOV mem, imm | mem <- imm |

- Doğrudan bir bellek hücrelerinden diğerine veri transferi yoktur
  - *There is no move 'mem<-mem' instruction*

# Move limitation

- ❑ Operandların ikisi de aynı boyutta olmalı
- ❑ Doğrudan bir değeri segment registre yükleme emri yoktur
  - ✓ Akümülatör (AX) üzerinden yapılır
- ❑ Doğrudan bir değeri belleğe yerleştirmek için verinin boyutu tanımlanmak zorundadır
  - ✓ Byte /Word PTR

# Move (*MOV*) Example

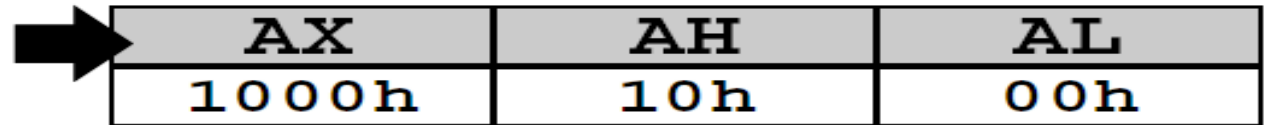
- `MOV AX, 100h`
- `MOV BX, AX`
- `MOV DX, BX`
  
- `MOV AX, 1234h`
- `MOV DX, 5678h`
- `MOV AL, DL`
- `MOV BH, DH`

# MOV Example

- `MOV AX, 1000h`
- `MOV [100h], AX`
- `MOV BX, [100h]`
- `MOV BYTE PTR [200h], 10h`
- `MOV WORD PTR [300h], 10h`
- `MOV AX, 2300h`
- `MOV DS, AX`

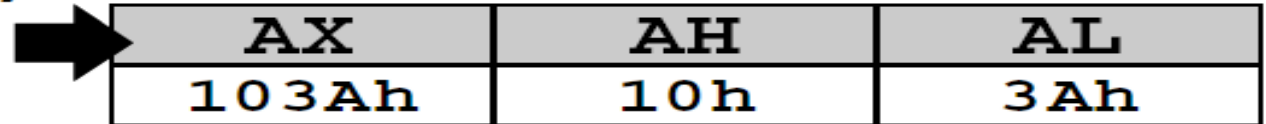
# MOV : 16 / 8 Bit register

MOV AX, 1000h



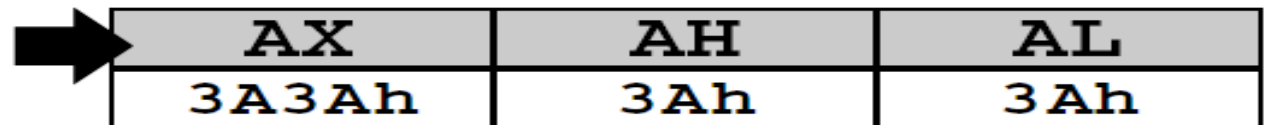
| AX    | AH  | AL  |
|-------|-----|-----|
| 1000h | 10h | 00h |

MOV AL, 3Ah



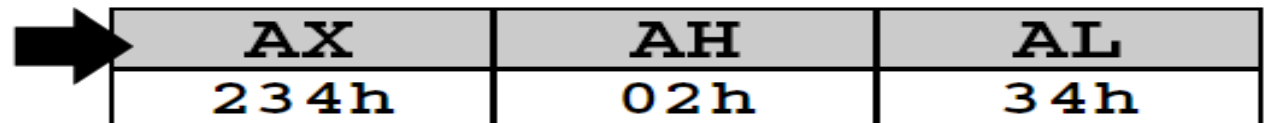
| AX    | AH  | AL  |
|-------|-----|-----|
| 103Ah | 10h | 3Ah |

MOV AH, AL



| AX    | AH  | AL  |
|-------|-----|-----|
| 3A3Ah | 3Ah | 3Ah |

MOV AX, 234h



| AX   | AH  | AL  |
|------|-----|-----|
| 234h | 02h | 34h |

# MOV : Memory

|     |            |         |  |
|-----|------------|---------|--|
| MOV | AX, 6789h  | DS:100h |  |
| MOV | DX, 1234h  | DS:101h |  |
| MOV | [100h], AX | DS:102h |  |
| MOV | [102h], DX |         |  |
| MOV | [104h], AH |         |  |
| MOV | [105h], DL |         |  |
| MOV | BX, [104h] |         |  |
| MOV | CX, [103h] |         |  |
| MOV | [106h], CL |         |  |

- Verinin koyulacağı yere ilişkin olarak sadece offset verildiğinde, otomatik olarak DS registeri segment register olur

# Endian conversion

- **Little endian conversion:**

In the case of 16-bit data, the low byte goes to the low memory location and the high byte goes to the high memory address. (Intel, Digital VAX)

- **Big endian conversion:**

The high byte goes to low address. (Motorola)

*Example:*

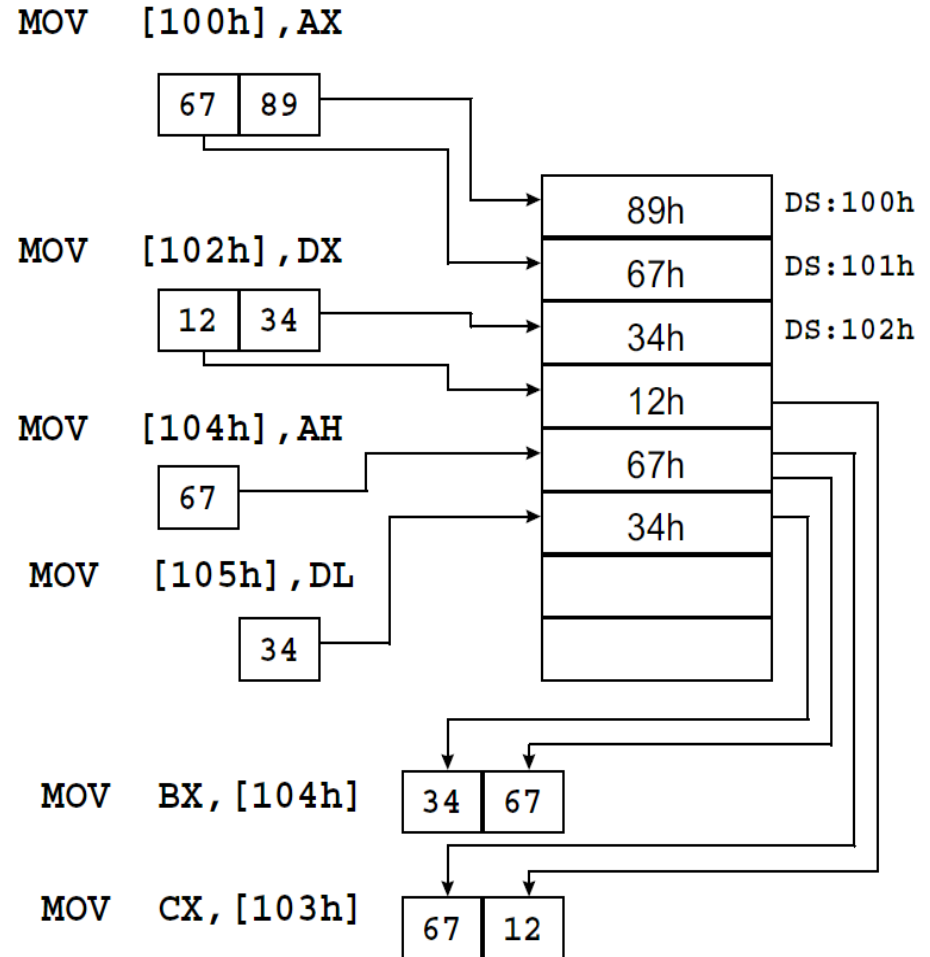
Suppose DS:6826 = 48, DS:6827 = 22,

Show the contents of register BX in the instruction **MOV BX,[6826]**

**Little endian conversion: BL = 48H, and BH = 22H**

# Byte ordering : Little endian

- ❑ x86'nın byte sırası: little endian
  - LSB küçük adrese yerleşir, MSB büyük adrese yerleşir



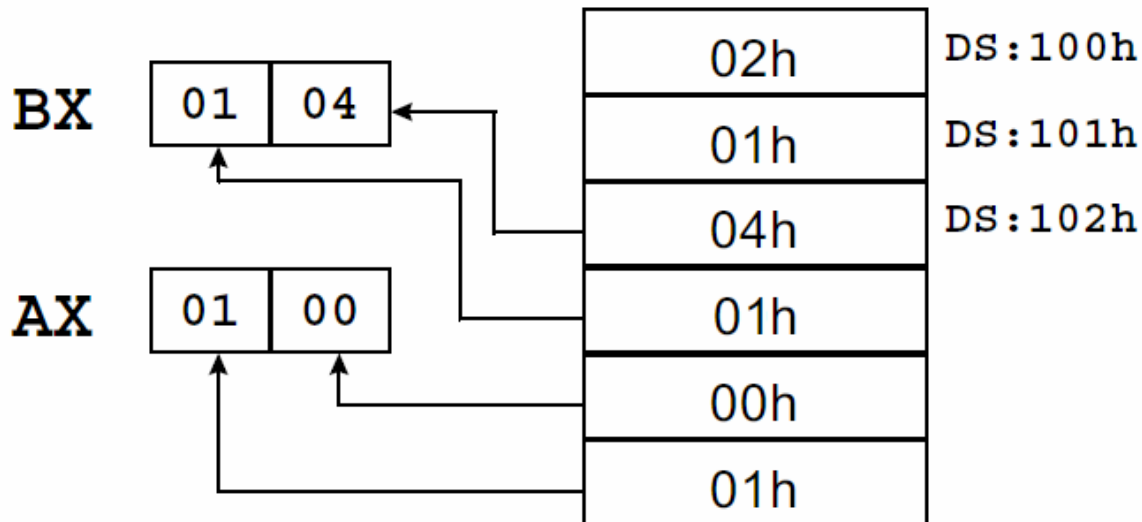
*Little endian* : Intel, Digital VAX

*Big endian* : Motorola



# Displacement

- BX(base) registerini kullanarak bellekte bir lokasyona işaret edebiliriz
- AX = ?



```
MOV AX, 102h
MOV BX, 100h
MOV CX, 4004h
MOV DX, 1201h
MOV [BX], AX
MOV [BX+2], CX
MOV [BX+3], DX
MOV [BX+4], BX
MOV BX, [102h]
MOV AX, [BX]
```

# What is the result of ...

- `MOV [100h] , 10h`
  - Address 100 = 10h
  - What about address 101?
  - Word or Byte?
    - `MOV WORD PTR [100h] , 10h`
    - `MOV BYTE PTR [100h] , 10h`
- What about `MOV [100h], AX` ?

# Hatırlatma...Complements

- Negatif sayıları temsil etmek için datalar, Complement formda saklanır
- **One's complements** of 01001100 (Bire tümleyen)

$$\begin{array}{r} 1111\ 1111 \\ -0100\ 1100 \\ \hline 1011\ 0011 \end{array}$$

- **Two's complements** (İkiye tümleyen)

$$\begin{array}{r} 1011\ 0011 \\ +0000\ 0001 \\ \hline 1011\ 0100 \end{array}$$

## Two's Complement

Decimal

-128

-127

-126

...

-2

-1

0

+1

+2

...

+127

Binary

1000 0000

1000 0001

1000 0010

...

1111 1110

1111 1111

0000 0000

0000 0001

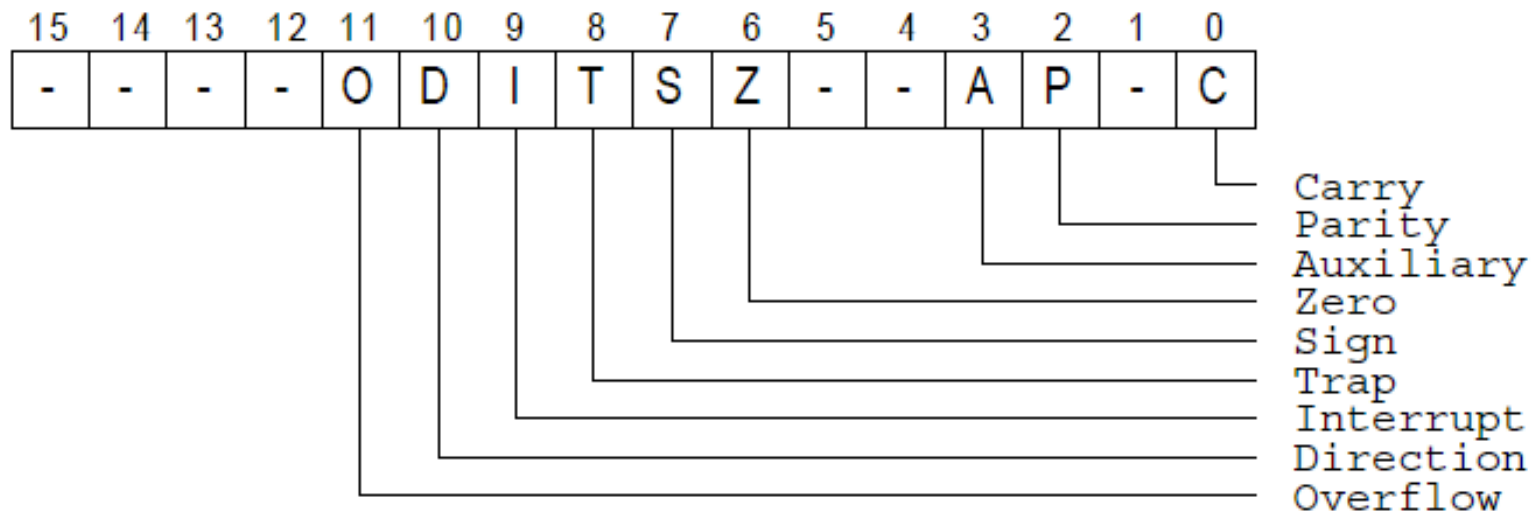
0000 0010

...

0111 1111

# Status Register (Flag)

- 8086, 16 bitlik bayrak registerine sahiptir
  - En son yapılan aritmetik işlemin durumunu gösterir



# Zero Flag

- İşlem sonucunun sıfır çıktığını gösterir
  - Eğer sonuç sıfır ise ZF=1
  - Sıfır değilse ZF=0
- İşlem sonucu pek çok şekilde sıfır çıkabilir

|                  |                       |  |                  |                         |  |                  |                    |
|------------------|-----------------------|--|------------------|-------------------------|--|------------------|--------------------|
| <code>mov</code> | <code>AL, 0FH</code>  |  | <code>mov</code> | <code>AX, 0FFFFH</code> |  | <code>mov</code> | <code>AX, 1</code> |
| <code>add</code> | <code>AL, 0F1H</code> |  | <code>inc</code> | <code>AX</code>         |  | <code>dec</code> | <code>AX</code>    |

Bu üç örneğin sonucu da sıfır çıkar ve ZF setlenir.

- İlgili emirler
  - **jz**     jump if zero (jump if ZF = 1)
  - **jnz**    jump if not zero (jump if ZF = 0)

# Zero Flag

- Zero bayrağının kullanılışı
  - İki ana kullanımı vardır
    - Eşitlik testi
      - Genellikle `cmp` emiriyle birlikte kullanılır
      - **`cmp AX, BX`**
    - Belirli bir değere kadar sayma
      - Bir register sayma değeri yüklenir
      - Register **`dec`** emiri ile azaltılır
      - **`jz/jnz`** emirleri kullanılarak program akışı yönlendirilir

# Zero Flag

- İşlem sonucu sıfır olduğunda Zero (**Z**) bayrağı setlenir.

EX

|     |         |       |          |
|-----|---------|-------|----------|
| MOV | AL, 10h | Z = ? |          |
| ADD | AL, F0h | Z = 1 | AL = 0   |
| ADD | AL, 20h | Z = 0 | AL = 20h |
| SUB | AL, 10h | Z = 0 | AL = 10h |
| SUB | AL, 10h | Z = 1 | AL = 0   |



# Parity Flag

- Sonucun alt 8 bitinin çift parity olma durumunu gösterir
- Alt 8 bit, çift sayıda değeri lojik-1 olan bit içeriyorsa Parity bayrağı setlenir
- 16- veya 32-bitlik değerler için sadece en düşük değerlikli 8 bit için party hesaplanır

## \* Example

|                         |                                |
|-------------------------|--------------------------------|
| <code>mov AL, 53</code> | <code>53D = 0011 0101B</code>  |
| <code>add AL, 89</code> | <code>89D = 0101 1001B</code>  |
|                         | <hr/>                          |
|                         | <code>142D = 1000 1110B</code> |

» As the result has even number of 1 bits, parity flag is set

# Parity Flag

- ilgili emirler

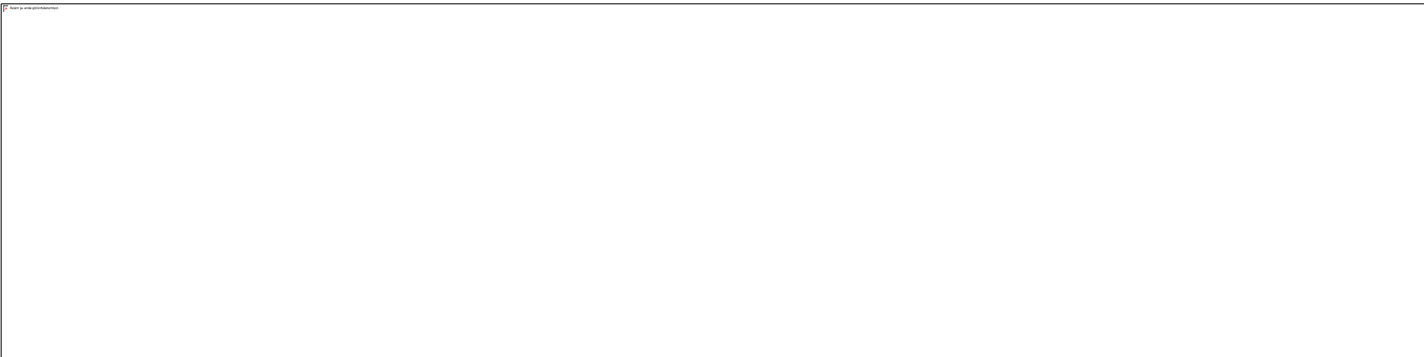
`jp`            jump on even parity (jump if PF = 1)  
`jnp`          jump on odd parity (jump if PF = 0)

EX

|     |         |     |        |
|-----|---------|-----|--------|
| MOV | AL, 14h | P=? |        |
| ADD | AL, 20h | P=0 | AL=34h |
| ADD | AL, 10h | P=1 | AL=44h |
| SUB | AL, 8h  | P=1 | AL=3Ch |
| SUB | AL, 10h | P=0 | AL=2Ch |

# Carry flag

- Bir elde veya borç alma durumu olduğunda Carry bayrağı setlenir (*sadece işaretsiz işlemlerle*)



EX

|     |         |       |          |
|-----|---------|-------|----------|
| MOV | AL, 77h | C = ? |          |
| ADD | AL, 50h | C = 0 | AL = C7h |
| ADD | AL, 50h | C = 1 | AL = 17h |
| SUB | AL, A0h | C = 1 | AL = 77h |
| ADD | AL, 27h | C = 0 | AL = 9Eh |

# Carry flag

- Carry bayrağı `inc` ve `dec` emirlerinden etkilenmez
  - Carry bayrağının içeriği aşağıdaki örneklerde değişmez

```
mov    AL, 0FFH
inc     AL
```

```
mov     AX, 0
dec     AX
```

- İlgili emirler
  - **jc**    jump if carry (jump if CF = 1)
  - **jnc**    jump if no carry (jump if CF = 0)
- Carry bayrağı doğrudan aşağıdaki emirler ile manipüle edilebilir

**stc** set carry flag (set CF to 1)    **cld** clear carry flag (clears CF to 0)

**cmc** complement carry flag (inverts CF value)

# Overflow flag

- Elde edilen sonuçta taşma olduğunda Overflow bayrağı setlenir. (*işaretli işlemlerle*)
- İşaretli sayılarla yapılan işlem sonucunda '*out-of-range*' durumu olduğunu gösterir
- Aşağıdaki kod overflow bayrağını set eder, fakat carry bayrağını set etmez

`mov AL, 72H` ; *72H = 114D*

`add AL, 0EH` ; *0EH = 14D*

# Overflow flag

Range of 8-, 16-, and 32-bit signed numbers

| size    | range                             |                              |
|---------|-----------------------------------|------------------------------|
| 8 bits  | – 128 to +127                     | – $2^7$ to $(2^7 - 1)$       |
| 16 bits | – 32,768 to +32,767               | – $2^{15}$ to $(2^{15} - 1)$ |
| 32 bits | – 2,147,483,648 to +2,147,483,647 | – $2^{31}$ to $(2^{31} - 1)$ |

EX

|     |         |     |        |
|-----|---------|-----|--------|
| MOV | AL, 77h | O=? |        |
| ADD | AL, 50h | O=1 | AL=C7h |
| ADD | AL, 50h | O=0 | AL=17h |
| SUB | AL, A0h | O=0 | AL=77h |
| ADD | AL, 27h | O=1 | AL=9Eh |

# Overflow flag

- Signed or unsigned: Sistem bunu nasıl bilecek?
  - ❖ İşlemci bunun yorumu yapamaz
  - ❖ Bu nedenle, hem carry hem overflow bayraklarını günceller

## Unsigned interpretation

```
mov    AL, 72H
add    AL, 0EH
jc     overflow
no_overflow:
    (no overflow code here)
    ....
overflow:
    (overflow code here)
    ....
```

## Signed interpretation

```
mov    AL, 72H
add    AL, 0EH
jo     overflow
no_overflow:
    (no overflow code here)
    ....
overflow:
    (overflow code here)
    ....
```

# Sign flag

- Sonucun işaretini belirtir.
  - (İşaretli işlemlerde) sonuç negatif çıktığında Sign flag setlenir.
- İşaretli sayılarla çalışıldığında kullanışlıdır.
  - Basitçe sonucun en değerli bitinin (MSb) bir kopyasıdır

EX

|     |         |       |          |
|-----|---------|-------|----------|
| MOV | AL, 77h | S = ? |          |
| ADD | AL, 50h | S = 1 | AL = C7h |
| ADD | AL, 50h | S = 0 | AL = 17h |
| SUB | AL, A0h | S = 0 | AL = 77h |
| ADD | AL, 27h | S = 1 | AL = 9Eh |



# Sign flag

## \* Examples

|                                                                                                                                     |                                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <code>mov AL, 15</code><br><code>add AL, 97</code><br><i>clears</i> the sign flag as<br>the result is 112<br>(or 0111000 in binary) | <code>mov AL, 15</code><br><code>sub AL, 97</code><br><i>sets</i> the sign flag as<br>the result is -82<br>(or 10101110 in binary) |
|-------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|

## \* Related instructions

`js`          jump if sign (jump if SF = 1)  
`jns`        jump if no sign (jump if SF = 0)

# Auxiliary flag

- Bir işlem sonucunda 8-, 16- veya 32-bitlik operandların en alt 4 bitinde (nibble) bir elde veya borç alma durumu olup olmadığını gösterir

- Operandın boyu fark etmez

\* Example

1 ← carry from lower 4 bits

|     |        |               |             |              |
|-----|--------|---------------|-------------|--------------|
| mov | AL, 43 | 43D =         | 0010        | 1011B        |
| add | AL, 94 | 94D =         | 0101        | 1110B        |
|     |        | <u>137D =</u> | <u>1000</u> | <u>1001B</u> |

- Alt nibble dan gelen bir elde olduğundan  
✓ Auxiliary flag setlenir

# Auxiliary flag

- İlgili emirler
  - Bu bayrak ile ilgili şartlı dallanma emiri yoktur
  - BCD sayılar üzerinde yapılan aritmetik işlemler bu bayrağı kullanır
    - **aaa** ASCII adjust for addition
    - **aas** ASCII adjust for subtraction
    - **aam** ASCII adjust for multiplication
    - **aad** ASCII adjust for division
    - **daa** Decimal adjust for addition
    - **das** Decimal adjust for subtraction

# More flag

- Direction flag
  - String işlemlerinde (artış/azalış) yönü tayin etmek için kullanılır
- Trap flag
  - Her işlemin ardından CPU yu kesmeye uğratmak için kullanılır
- Interrupt flag
  - Donanımsal kesmeyi aktif/pasif etmek için kullanılır

# Flag set/reset instructions

- Carry flag                      STC / CLC
- Direction flag                STD / CLD
- Interrupt flag                STI / CLI

# **Arithmetic instructions**

# Increment - Decrement

## ❑ INC / DEC

- |                |              |
|----------------|--------------|
| ▪ INC register | DEC register |
| ▪ INC memory   | DEC memory   |

## ❑ Example

- INC AX
- DEC BL

# Increment - Decrement

`inc destination`

» Performs increment operation

`destination := destination + 1`

`dec destination`

» Performs decrement operation

`destination := destination - 1`

- ❑ INC ve DEC emirleri carry bayrağını etkilemez
  - Diğer 5 durum bayrağını etkiler

\* In general

`inc BX`

is better than

`add BX, 1`

- ✓ Her ikisi de aynı sürede gerçekleşse de INC emiri bellekte daha az yer kaplar



# Add

- 5 operand kombinasyonu mümkündür

- ☐ ADD reg, imm
- ☐ ADD reg, mem
- ☐ ADD reg, reg
- ☐ ADD mem, imm
- ☐ ADD mem, reg

- ADC reg, imm
- ADC reg, mem
- ADC reg, reg
- ADC mem, imm
- ADC mem, reg

# Add

## \* Basic format

`add destination, source`

» Performs simple integer addition

`destination := destination + source`

---

## \* Basic format

`adc destination, source`

» Performs integer addition with carry

`destination := destination + source + CF`

- ADC emiri, uzun sayıların (Long int) toplamasında kullanılır
- Carry bayrağını manipüle etmek için 3 tane emir kullanılır
  - **stc** set carry flag (set CF to 1)
  - **c1c** clear carry flag (clears CF to 0)
  - **cmc** complement carry flag (inverts CF value)

# Example ADD

- `MOV AL, 10h`
- `ADD AL, 20h` ; *AL = 30h*
- `MOV BX, 200h` ; *BX = 0200h*
- `MOV WORD PTR [BX], 10h`
- `ADD WORD PTR [BX], 70h`
- `MOV AH, 89h` ; *AX = 8930h*
- `ADD AX, 9876h` ; *AX = 21A6h*
- `ADC BX, 01h` ; *BX = 0202h ?*

# Subtract

- 5 operand kombinasyonu mümkündür

☐ SUB reg, imm

☐ SUB reg, mem

☐ SUB reg, reg

☐ SUB mem, imm

☐ SUB mem, reg

SBB reg, imm

SBB reg, mem

SBB reg, reg

SBB mem, imm

SBB mem, reg

# Sub

## Subtraction instructions

**sub**      **destination, source**

» Performs simple integer subtraction

**destination := destination - source**

---

**sbb**      **destination, source**

» Performs integer subtraction with borrow

**destination := destination - source - CF**

# Example SUB

- `MOV AL, 10h`
- `ADD AL, 20h` ; *AL = 30h*
- `MOV BX, 200h` ; *BX = 0200h*
- `MOV WORD PTR [BX], 10h`
- `SUB WORD PTR [BX], 70h`
- `MOV AH, 89h` ; *AX = 8930h*
- `SBB AX, 0001h` ; *AX = 892Eh ?*
- `SBB AX, 0001h` ; *AX = 892Dh*

# Compare

- ❑ CMP reg, imm
- ❑ CMP reg, mem
- ❑ CMP reg, reg
- ❑ CMP mem, reg

`cmp destination, source`

» Performs subtraction without updating destination

`destination - source`

❖ There is no “CMP mem, imm”

- 6 durum bayrağının hepsini etkiler
- CMP emirini genellikle bir koşullu dallanma emri takip eder

# Example CMP

- MOV CX, 10h
- CMP CX, 20h ; Z=0, S=1, C=1, OF=0
- MOV BX, 40h
- CMP BX, 40h ; Z=1, S=0, C=0, OF=0
- MOV AX, 30h
- CMP AX, 20h ; Z=0, S=0, C=0, OF=0



# Negation

- Compute 2's complement.
- Carry bayrağı her zaman setlenir.
- Kullanımı
  - NEG reg
  - NEG mem

# Example NEG

- `MOV CX, 10h`
- `NEG CX` ; `CX = 0FFF0h`
- `MOV AX, 0FFFFH`
- `NEG AX` ; `AX = 1`
- `MOV BX, 1H`
- `NEG BX` ; `BX = 0FFFFh`

# Multiplication

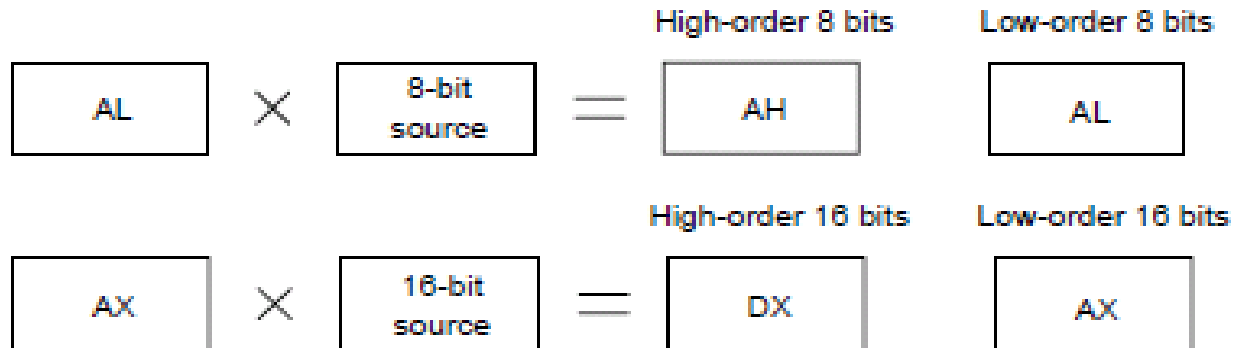
- Add/sub işlemlerinden daha karmaşıktır
  - İki katı uzunlukta sonuçlar üretir
    - Mesela iki tane 8-bitlik sayıları çarpmak 16-bit gerektiren bir sonuç üretir
  - İşaretli ve işaretsiz sayıların her ikisi için tek bir çarpma emiri kullanılmaz
    - fakat `add` ve `sub` emirleri hem işaretli hem de işaretsiz sayılarda çalışıyordu
    - Çarpma işlemi için ise iki ayrı emire ihtiyaç duyulur
      - ✓ **`mul`** for unsigned numbers (işaretsiz sayılar)
      - ✓ **`imul`** for signed numbers (işaretli sayılar)
- Her zaman akümülatör üzerinden gerçekleşir
- Sadece **Carry** ve **Overflow** bayrağını etkiler

# Multiplication

## ■ Unsigned multiplication

**`mul`** *source*

- *source* operandının uzunluğuna bağlı olarak diğer kaynak operandı ve hedefler belirlenir



## ■ Örnek

```
mov AL,10
```

```
mov DL,25
```

```
mul DL ; produces 250D in AX register (result fits in AL)
```

# Multiplication

- signed multiplication

**imul** emiri aynı *syntaxı* kullanır

- Örnek

```
mov DL,0FFH ; DL := -1
```

```
mov AL,0BEH ; AL := -66
```

```
imul DL
```

produces 66D in AX register (again, result fits in AL)

# 8 bit multiplication

- AL is multiplicand (*çarpılan*)
- AX keep the result

```
MOV  AL, 10h      ; AL = 10h
MOV  CL, 13h      ; CL = 13h
IMUL CL           ; AX = 0130h
```

# 16 bit multiplication

- AX is multiplicand (*çarpılan*)
- DX:AX keep the result

```
MOV AX, 0100h
```

*; AX = 0100h*

```
MOV BX, 1234h
```

*; BX = 1234h*

```
IMUL BX
```

*; DX = 0012h*

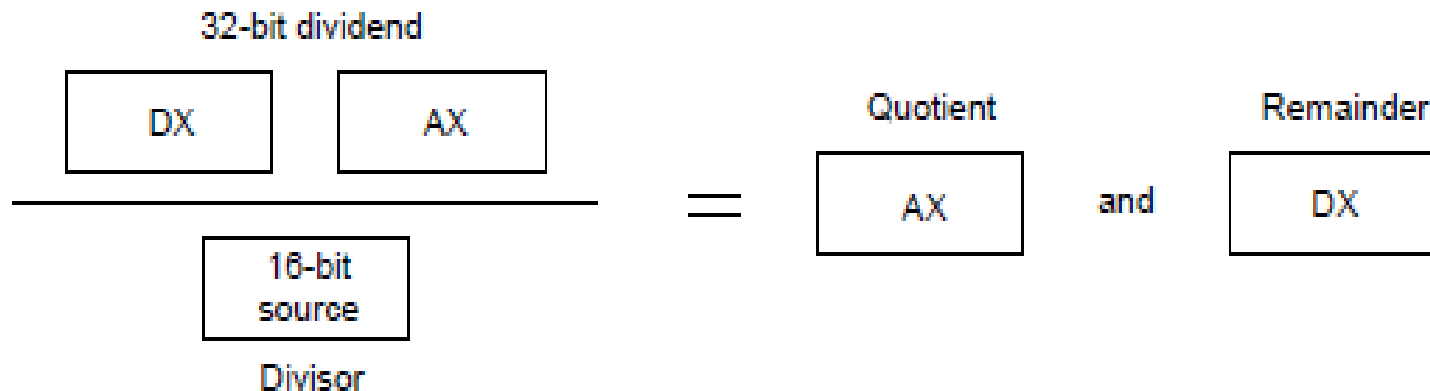
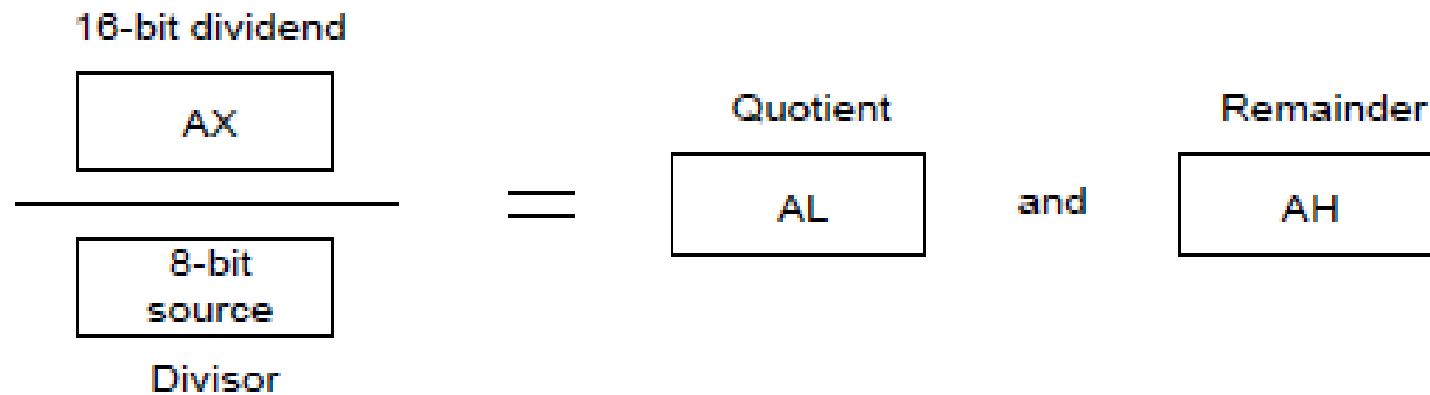
*; AX = 3400h*

# Division

- Division instruction
  - Çarpma işleminden daha karmaşıktır
    - İki sonuç üretir
      - 1) Quotient (*Bölüm*)
      - 2) Remainder (*Kalan*)
  - Çarpmadaki gibi iki emir vardır
    - ✓ `div` source for unsigned numbers
    - ✓ `idiv` source for signed numbers



# Division



# Division

## ■ Example (8-bit division)

`mov AX, 2F1BH` ; *AX = 12059D*

`mov CL, 64H` ; *CL = 100D*

`div CL`

produces 120D(78h) in AL and 59D(3Bh) as remainder in AH

*; AL=78h, AH=3Bh*

## Example (16-bit division)

`mov DX, 0001H`

`mov AX, 2345H` ; *AX = 74565D*

`mov CX, 012CH` ; *CX = 300D*

`div CX`

produces 248D (F8h) in AX and 165D(A5h) as remainder in DX

*; AX=F8D, DX=A5H*

# Conversion

İşaretli bölme biraz hassas bir konudur, yardıma ihtiyaç duyar

- 16 bitlik işaretsiz sayılar, üst 16 bitine lojik-0 konulmak suretiyle 32 bite genişletilebilir
- Fakat bu mantık işaretli sayılarda işe yaramaz
  - işaretli bir sayıyı genişletmek için sayının işaret bitinin bu üst bitlere kopyalanması gerekmektedir.

işaret genişletmeye yardımcı olan emirler:

❑ Byte to Word : **CBW**

- Signed convert AL -> AX

❑ Word to Double word : **CWD**

- Signed convert AX -> DX:AX

# Example Conversion

MOV AL, 22h

CBW ; AX=0022h

MOV AL, F0h

CBW ; AX=FFF0h

MOV AX, 3422h

CWD ; DX=0000h  
; AX=3422h

| Instruction | Flag affected |                  |            |            |            |
|-------------|---------------|------------------|------------|------------|------------|
|             | Z-flag        | C-flag           | S-flag     | O-flag     | A-flag     |
| ADD         | <i>yes</i>    | <i>yes</i>       | <i>yes</i> | <i>yes</i> | <i>yes</i> |
| ADC         | <i>yes</i>    | <i>yes</i>       | <i>yes</i> | <i>yes</i> | <i>yes</i> |
| SUB         | <i>yes</i>    | <i>yes</i>       | <i>yes</i> | <i>yes</i> | <i>yes</i> |
| SBB         | <i>yes</i>    | <i>yes</i>       | <i>yes</i> | <i>yes</i> | <i>yes</i> |
| INC         | <i>yes</i>    | <b><i>no</i></b> | <i>yes</i> | <i>yes</i> | <i>yes</i> |
| DEC         | <i>yes</i>    | <b><i>no</i></b> | <i>yes</i> | <i>yes</i> | <i>yes</i> |
| NEG         | <i>yes</i>    | <i>yes</i>       | <i>yes</i> | <i>yes</i> | <i>yes</i> |
| CMP         | <i>yes</i>    | <i>yes</i>       | <i>yes</i> | <i>yes</i> | <i>yes</i> |
| MUL         | <i>no</i>     | <i>yes</i>       | <i>no</i>  | <i>yes</i> | <i>no</i>  |
| IMUL        | <i>no</i>     | <i>yes</i>       | <i>no</i>  | <i>yes</i> | <i>no</i>  |
| DIV         | <i>no</i>     | <i>no</i>        | <i>no</i>  | <i>no</i>  | <i>no</i>  |
| IDIV        | <i>no</i>     | <i>no</i>        | <i>no</i>  | <i>no</i>  | <i>no</i>  |
| CBW         | <i>no</i>     | <i>no</i>        | <i>no</i>  | <i>no</i>  | <i>no</i>  |
| CWD         | <i>no</i>     | <i>no</i>        | <i>no</i>  | <i>no</i>  | <i>no</i>  |

# Example about flag with arithmetic

| Instruction |           | Z-flag | C-flag | O-flag | S-flag | P-flag |           |
|-------------|-----------|--------|--------|--------|--------|--------|-----------|
| MOV         | AX, 7100h | ?      | ?      | ?      | ?      | ?      |           |
| MOV         | BX, 4000h | ?      | ?      | ?      | ?      | ?      |           |
| ADD         | AX, BX    | 0      | 0      | 1      | 1      | 1      | AX=0B100h |
| ADD         | AX, 7700h | 0      | 1      | 0      | 0      | 1      | AX=2800h  |
| SUB         | AX, 2000h | 0      | 0      | 0      | 0      | 1      | AX=0800h  |
| SUB         | AX, 1000h | 0      | 1      | 0      | 1      | 1      | AX=F800h  |
| ADD         | AX, 0800h | 1      | 1      | 0      | 0      | 1      | AX=0000h  |

# Example about flag with arithmetic

| Instruction |         | Z-flag | C-flag | O-flag | S-flag | P-flag |         |
|-------------|---------|--------|--------|--------|--------|--------|---------|
| MOV         | AL, 10  | ?      | ?      | ?      | ?      | ?      |         |
| ADD         | AL, F0h | 0      | 0      | 0      | 1      | 1      | AL=0FAh |
| ADD         | AL, 6   | 1      | 1      | 0      | 0      | 1      | AL=0    |
| SUB         | AL, 5   | 0      | 1      | 0      | 1      | 0      | AL=0FBh |
| INC         | AL      | 0      | 1      | 0      | 1      | 1      | AL=0FCh |
| ADD         | AL, 10  | 0      | 1      | 0      | 0      | 1      | AL=6h   |
| ADD         | AL, FBh | 0      | 1      | 0      | 0      | 0      | AL=1h   |
| DEC         | AL      | 1      | 1      | 0      | 0      | 1      | AL=0h   |
| DEC         | AL      | 0      | 1      | 0      | 1      | 1      | AL=0FFh |
| INC         | AL      | 1      | 1      | 0      | 0      | 1      | AL=0    |

| Instruction |         | Z-flag | C-flag | O-flag | S-flag | P-flag |              |
|-------------|---------|--------|--------|--------|--------|--------|--------------|
| MOV         | AL, 120 | ?      | ?      | ?      | ?      | ?      |              |
| ADD         | AL, 15  | 0      | 0      | 1      | 1      | 1      | AL=87h= -121 |
| NEG         | AL      | 0      | 1      | 0      | 0      | 0      | AL=79h       |
| SUB         | AL, 130 | 0      | 1      | 1      | 1      | 0      | AL=0F7h      |

# **Logical & Shift, Rotation** **instructions**



# AND, OR, XOR

□ AND / OR / XOR

REG, memory

memory, REG

REG, REG

memory, immediate

REG, immediate

## ■ Bayraklar

| C | Z | S | O | P |
|---|---|---|---|---|
| 0 | r | r | 0 | r |

# NOT

□ NOT

REG  
memory

## ■ Bayraklar

|                         |          |          |          |          |          |
|-------------------------|----------|----------|----------|----------|----------|
| <b>C</b>                | <b>Z</b> | <b>S</b> | <b>O</b> | <b>P</b> | <b>A</b> |
| <hr/>                   |          |          |          |          |          |
| unchanged<br>(değişmez) |          |          |          |          |          |

# SHift Left

□ SHL

memory, immediate

REG, immediate

memory, CL

REG, CL

- Operand1'in içeriği 1 bit sola kaydırılır. Eğer operand2 verilirse bu, kaydırma miktarını belirtir.
  - MSb'den dışarı çıkan bit **CF**'ye kopyalanır.
  - LSb'den içeri *lojik-0* alınır.
- Bayraklar

| C | O |
|---|---|
| r | r |

Eğer ilk operand işaretini  
korursa OF=0 olur

# SHift Right

□ SHR

memory, immediate

REG, immediate

memory, CL

REG, CL

- Operand1'in içeriği 1 bit sağa kaydırılır. Eğer operand2 verilirse bu, kaydırma miktarını belirtir.
  - LSb'den dışarı çıkan bit **CF**'ye kopyalanır.
  - MSb'den içeri *lojik-0* alınır.
- Bayraklar

| C | O |
|---|---|
| r | r |

Eğer ilk operand işaretini  
korursa OF=0 olur

# ROtate Left

□ ROL

memory, immediate

REG, immediate

memory, CL

REG, CL

- Operand1'in içeriği 1 bit sola döndürülür. Eğer operand2 verilirse bu, döndürme miktarını belirtir.
- MSb'den dışarı çıkan bit **CF**'ye kopyalanır ve ayrıca aynı bit LSb'den içeri alınır.

▪ Bayraklar

|   |   |
|---|---|
| c | o |
| r | r |

Eğer ilk operand işaretini  
korursa OF=0 olur

# ROtate Right

□ ROR

memory, immediate

REG, immediate

memory, CL

REG, CL

- Operand1'in içeriği 1 bit sağa döndürülür. Eğer operand2 verilirse bu, döndürme miktarını belirtir.
- LSb'den dışarı çıkan bit **CF**'ye kopyalanır ve ayrıca aynı bit MSb'den içeri alınır.

▪ Bayraklar

|   |   |
|---|---|
| c | o |
| r | r |

Eğer ilk operand işaretini  
korsa OF=0 olur

# Rotate Left through Carry

□ RCL

memory, immediate

REG, immediate

memory, CL

REG, CL

- Operand1'in içeriği Carry bayrağı üzerinden 1 bit sola döndürülür. Eğer operand2 verilirse bu, döndürme miktarını belirtir.
- MSb'den dışarı çıkan bit **CF**'ye kopyalanır ve CF'nin eski değeri LSb'den içeri alınır.

■ Bayraklar

| c | o |
|---|---|
| r | r |

Eğer ilk operand işaretini  
korsa OF=0 olur

# Rotate Right through Carry

□ RCR

memory, immediate

REG, immediate

memory, CL

REG, CL

- Operand1'in içeriği Carry bayrağı üzerinden 1 bit sağa döndürülür. Eğer operand2 verilirse bu, döndürme miktarını belirtir.
- LSb'den dışarı çıkan bit **CF**'ye kopyalanır ve CF'nin eski değeri MSb'den içeri alınır.

■ Bayraklar

|   |   |
|---|---|
| c | o |
| r | r |

Eğer ilk operand işaretini  
korsa OF=0 olur



# SHL, SHR Example

- SHL example

```
MOV AL, 11100000b
```

```
SHL AL, 1    ; AL = 11000000b, CF=1
```

- SHR example

```
MOV AL, 10000111b
```

```
SHR AL, 1    ; AL = 01000011b, CF=1
```

# ROL, ROR Example

- ROL example

MOV AL, 90h ; *AL = 10010000b*

ROL AL, 1 ; *AL = 00100001b, CF=1*

- ROR example

MOV AL, 0Fh ; *AL = 00001111b*

ROR AL, 1 ; *AL = 10000111b, CF=1*

# RCL, RCR Example

- RCL example

```
STC                ; set carry (CF=1)
MOV AL, 1Ch        ; AL = 00011100b
RCL AL, 1           ; AL = 00111001b, CF=0
```

- RCR example

```
STC                ; set carry (CF=1)
MOV AL, 1Ch        ; AL = 00011100b
RCR AL, 1           ; AL = 10001110b, CF=0
```

# Jump and Loops

# Jump Instructions

| Emir  | Anlamı             | İlgili Bayrak |
|-------|--------------------|---------------|
| JMP   | Jump               |               |
| ----- |                    |               |
| JZ    | Jump if Zero       | ZF=1          |
| JNZ   | Jump if Not Zero   | ZF=0          |
| JO    | Jump if Overflow   | OF=1          |
| JNO   | J. if Not Overflow | OF=0          |
| JC    | Jump if Carry      | CF=1          |
| JNC   | Jump if No Carry   | CF=0          |
| JS    | Jump if Sign       | SF=1          |
| JNS   | Jump if No Sign    | SF=0          |

# Jump Instructions

|          | Emir | Anlamı                 | İlgili Bayrak      |
|----------|------|------------------------|--------------------|
|          | JPO  | Jump if Parity Odd     | PF=0               |
|          | JPE  | Jump if Parity Even    | PF=1               |
| Unsigned | JA   | Jump if Above          | (CF and ZF)=0      |
|          | JB   | Jump if Below          | CF=1               |
|          | JAE  | J. if Above or Equal   | CF=0               |
|          | JBE  | J. if Below or Equal   | (CF or ZF)=1       |
| Signed   | JG   | Jump if Greater        | ZF=0 and SF=OF     |
|          | JL   | Jump if Less           | SF<>OF             |
|          | JGE  | J. if Greater or Equal | SF=OF              |
|          | JLE  | J. if Less or Equal    | (ZF=1) or (SF<>OF) |
|          | JCXZ | Jump if CX=0           | CX=0<br>(SF<>OF)   |
|          | JCXZ | Jump if CX=0           | CX=0               |

# Jump Instructions

---

|     |      |                      |
|-----|------|----------------------|
| JZ  | JE   | Equal                |
| JNZ | JNE  | Not Equal            |
| JA  | JNBE | Not Below or Equal   |
| JB  | JNAE | Not Above or Equal   |
| JAЕ | JNB  | Not Below            |
| JBE | JNA  | Not Above            |
| JG  | JNLE | Not Less or Equal    |
| JL  | JNGE | Not Greater or Equal |
| JGE | JNL  | Not Less             |
| JLE | JNG  | Not Greater          |

# Loop Instructions

❑ Döngüyü sayısı için CX (Counter) registerine bağımlıdır

❑ Döngü(Loop) emirleri

- LOOP
- LOOPZ / LOOPE
- LOOPNZ / LOOPNE
- JCXZ



# Loop Instructions

- LOOP label

- ❖ Decrease CX, jump to label if CX not zero.

- ❖ Algorithm:

- $CX = CX - 1$

- if  $CX \neq 0$  then  
                                jump

- else  
                                no jump, continue

# Loop Instructions

- LOOPZ / LOOPE label

- ❖ Decrease CX, jump to label if CX not zero and ZF = 1.

- ❖ Algorithm:

- $CX = CX - 1$
    - if  $(CX \neq 0)$  and  $(ZF=1)$  then  
                                jump
    - else  
                                no jump, continue

# Loop Instructions

- LOOPNZ / LOOPNE label
  - ❖ Decrease CX, jump to label if CX not zero and ZF = 0.
  - ❖ Algorithm:
    - $CX = CX - 1$
    - if  $(CX \neq 0)$  and  $(ZF=0)$  then  
jump
    - else  
no jump, continue

# Loop Instructions

- JCXZ label

- ❖ Jump if CX register is 0.

- ❖ LOOP emiri ile birlikte döngüye girmeden önce CX registerinin içeriğini öğrenmeye yarar

- ❖ Kullanım şekli:

```
        initialization
        jcxz  endloop          ; CX = 0 ?
label1:
        actions
        loop  label1          ; loop
endloop:
```

# Nested For Loop in Assembly Lang.

- Consider the following code

```
sum := 0
for (I = 1 to M)
    for (j = 1 to N)
        sum := sum + 1
    end for
end for
```

- Assembly code

```
sub    AX,AX ; AX := 0
mov    DX,M
outer_loop:
    mov    CX,N
inner_loop:
    inc    AX
    loop   inner_loop
    dec    DX
    jnz    outer_loop
exit_loops:
    mov    sum,AX
```