

Bölüm 2: 8086 Yazmaçlar

Mikroişlemciler

Yazmaç kümesi

AX · BX · CX · DX

SP · BP · SI · DI

Kesim yazmaçları ve IP

Bayrak yazmacı

Durum ve kontrol bayrakları

Adresleme modları

Etkin ve fiziksel adres

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Yazmaç kümesi

8086'nın 14 yazmacını gruplarına göre sayabilmek ve 8/16 bitlik kullanımını açıklamak.

Özel görevler

AX, BX, CX, DX, SP, BP, SI, DI yazmaçlarının komutlardaki örtük (implicit) görevlerini bilmek.

Bayraklar

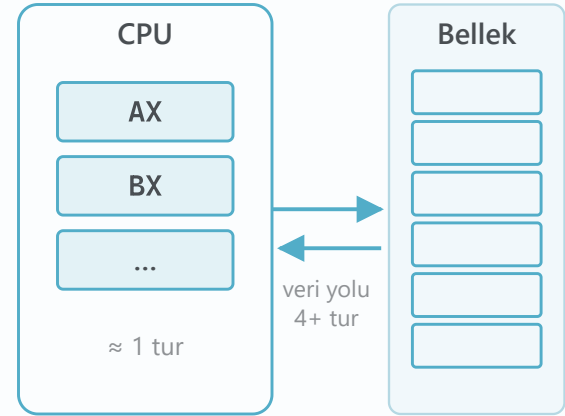
Bir işlemden sonra CF, ZF, SF, OF, PF, AF bayraklarını elle hesaplayabilmek.

Adresleme

Adresleme modlarını tanımak; etkin adres (EA) ve fiziksel adresi hesaplamak.

Yazmaç Nedir?

- **Yazmaç (register)**: işlemcinin *içinde* bulunan, çok hızlı, küçük bellek hücresidir.
- 8086'da yazmaçlar **16 bit** genişliğindedir; dört tanesi ayrıca 8 bitlik iki yarıya bölünebilir.
- Bellek erişimi dış veri yolu üzerinden birkaç saat turu sürer; yazmaç erişimi neredeyse **bedavadır**.
- Geçici veriler, adresler, sayaçlar ve işlem sonuçları yazmaçlarda tutulur.
- Komutların çoğu en az bir işleneni yazmaçta ister: 8086'da **bellekten belleğe** MOV/ADD yoktur.

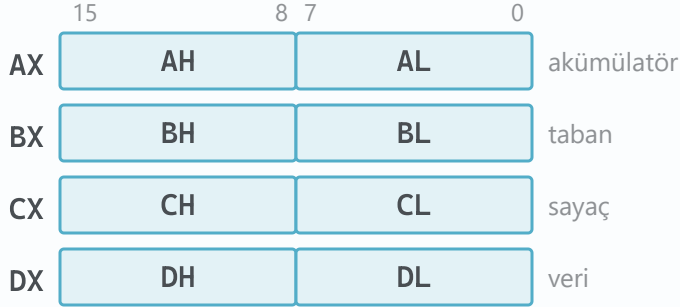


8086'da **14** yazmaç vardır: 4 genel amaçlı veri yazmacı, 4 işaretçi/indis yazmacı, 4 kesim yazmacı, komut işaretçisi (IP) ve bayrak yazmacı.

YAZMAÇ KÜMESİ

8086 Yazmaç Kümesi

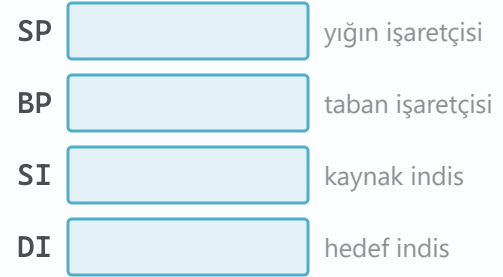
Genel amaçlı (veri) yazmaçları



Kesim (segment) yazmaçları



İşaretçi ve indis yazmaçları



Özel yazmaçlar



Hepsi 16 bittir. Yalnızca AX, BX, CX ve DX ikiye bölünebilir (AH/AL ...); SP, BP, SI, DI ve kesim yazmaçlarının 8 bitlik parçası **yoktur**. Bu derste AX...DI sekizlisi **genel amaçlı yazmaçlar** olarak sayılır.

Genel Amaçlı Yazmaçlar

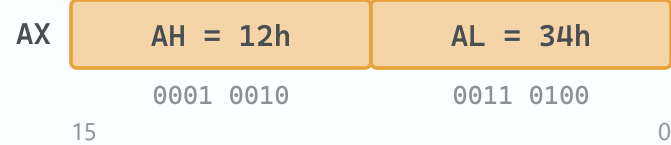
Geçici verileri saklamak için kullanılırlar (general purpose registers). Çoğu komutta herhangi biri kullanılabilir; ancak bazı komutlar belirli bir yazmacı **örtük** olarak kullanır.

Yazmaç	Ad	8 bitlik yarılar	Örtük kullanım (örnek)
AX	Accumulator	AH, AL	MUL, DIV, IN, OUT, LODS/STOS
BX	Base	BH, BL	bellek adresleme [BX], XLAT
CX	Counter	CH, CL	LOOP, REP, kaydırma sayısı CL
DX	Data	DH, DL	MUL/DIV yüksek yarısı, port adresi
SP	Stack Pointer	—	PUSH, POP, CALL, RET
BP	Base Pointer	—	yığın çerçevesi [BP+n]
SI	Source Index	—	dize kaynağı DS:SI
DI	Destination Index	—	dize hedefi ES:DI

AX = AH : AL — 8 ve 16 Bitlik Kullanım

```
► MOV AX, 1234h ; AH=12h, AL=34h
2 MOV AL, 0FFh ; yalnız alt yarı
3 ADD AL, 1 ; 8 bitlik toplama
4 MOV AX, 12FFh
5 ADD AX, 1 ; 16 bitlik toplama
```

Bir 8 bitlik yarıya yazmak diğer yarıyı **değiştirmez**. 8 bitlik işlemin eldesi komşu yarıya geçmez, CF bayrağına gider.



AX'e 1234h yazılır: yüksek bayt **AH = 12h**, düşük bayt **AL = 34h**.

Adım 1 / 5

DEĞİŞKENLER

AX = 1234h

AH = 12h

AL = 34h

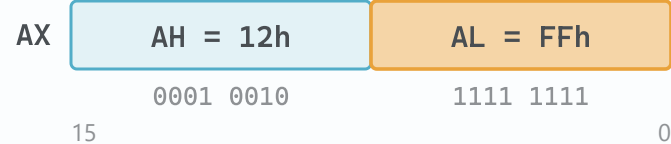
CF = 0

ZF = 0

AX = AH : AL — 8 ve 16 Bitlik Kullanım

```
1 MOV AX, 1234h ; AH=12h, AL=34h
▶ MOV AL, 0FFh ; yalnız alt yarı
3 ADD AL, 1 ; 8 bitlik toplama
4 MOV AX, 12FFh
5 ADD AX, 1 ; 16 bitlik toplama
```

Bir 8 bitlik yarıya yazmak diğer yarıyı **değiştirmez**. 8 bitlik işlemin eldesi komşu yarıya geçmez, CF bayrağına gider.



Yalnızca AL değişir; AH korunur → AX = **12FFh**.

Adım 2 / 5

DEĞİŞKENLER

AX = 12FFh

AH = 12h

AL = FFh

CF = 0

ZF = 0

AX = AH : AL — 8 ve 16 Bitlik Kullanım

```
1 MOV AX, 1234h ; AH=12h, AL=34h
2 MOV AL, 0FFh ; yalnız alt yarı
3 ► ADD AL, 1 ; 8 bitlik toplama
4 MOV AX, 12FFh
5 ADD AX, 1 ; 16 bitlik toplama
```

Bir 8 bitlik yarıya yazmak diğer yarıyı **değiştirmez**. 8 bitlik işlemin eldesi komşu yarıya geçmez, CF bayrağına gider.

AX

AH = 12h

AL = 00h

0001 0010

0000 0000

15

0

FFh + 1 = 100h, 8 bite sığmaz: AL = **00h**, CF = 1, ZF = 1.
Elde **AH'ye geçmez**; AX = 1200h.

Adım 3 / 5

DEĞİŞKENLER

AX = 1200h

AH = 12h

AL = 00h

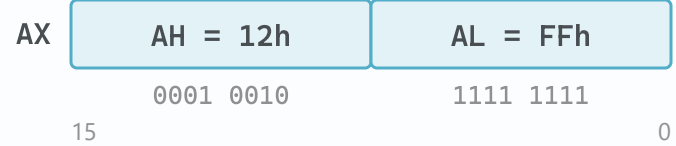
CF = 1

ZF = 1

AX = AH : AL — 8 ve 16 Bitlik Kullanım

```
1 MOV AX, 1234h ; AH=12h, AL=34h
2 MOV AL, 0FFh ; yalnız alt yarı
3 ADD AL, 1 ; 8 bitlik toplama
▶ MOV AX, 12FFh
5 ADD AX, 1 ; 16 bitlik toplama
```

Bir 8 bitlik yarıya yazmak diğer yarıyı **değiştirmez**. 8 bitlik işlemin eldesi komşu yarıya geçmez, CF bayrağına gider.



AX yeniden 12FFh yapılır.

Adım 4 / 5

DEĞİŞKENLER

AX = 12FFh

AH = 12h

AL = FFh

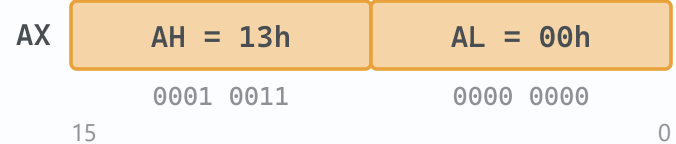
CF = 1

ZF = 1

AX = AH : AL — 8 ve 16 Bitlik Kullanım

```
1 MOV AX, 1234h ; AH=12h, AL=34h
2 MOV AL, 0FFh ; yalnız alt yarı
3 ADD AL, 1 ; 8 bitlik toplama
4 MOV AX, 12FFh
▶ ADD AX, 1 ; 16 bitlik toplama
```

Bir 8 bitlik yarıya yazmak diğer yarıyı **değiştirmez**. 8 bitlik işlemin eldesi komşu yarıya geçmez, CF bayrağına gider.



Bu kez işlem 16 bitlik: 12FFh + 1 = **1300h**; elde AH'ye geçti, CF = 0.

Adım 5 / 5

DEĞİŞKENLER

AX = 1300h

AH = 13h

AL = 00h

CF = 0

ZF = 0

AX — Akümülatör

- 16 bit genişliğindedir; **AH** (yüksek sıra) ve **AL** (düşük sıra) olarak 8 bitlik iki bölüme ayrılır.
- 8 bitlik komutlarda AH ve AL ayrı ayrı kullanılabilir.
- Genellikle aritmetik ve mantıksal komutlarda kullanılır.
- Bazı komutlar AX/AL'yi **örtük** kullanır: `MUL`, `DIV`, `IN / OUT`, `CBW`, `DAA`, `LODSB / STOSB`.
- AX ile çalışan birçok komutun makine kodu bir bayt **daha kısadır** (ör. `ADD AX, imm16`).

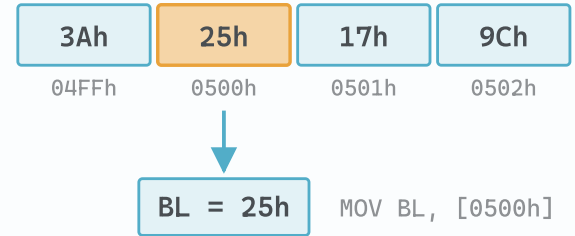
```
ADD AX, AX      ; AX = AX + AX
MOV AX, 0005h
ADD AX, AX      ; AX = 000Ah
MOV AH, 4Ch     ; DOS: programı bitir
INT 21h
```

AX'i kendisiyle toplamak, değeri 2 ile çarpmakla aynıdır.

BX — Taban (Base)

- 16 bit; **BH** ve **BL** olarak 8 bitlik iki yarıya ayrılır.
- Dolaylı bellek adreslemede **bağlı konum (offset)** değerini tutabilen tek veri yazmacıdır:
[BX] , [BX+SI] , [BX+4] .
- Dizi ve tablo (matris) işlemlerinde başlangıç adresini tutmak için kullanılır.
- [AX] , [CX] , [DX] biçiminde adresleme 8086'da **yoktur**.

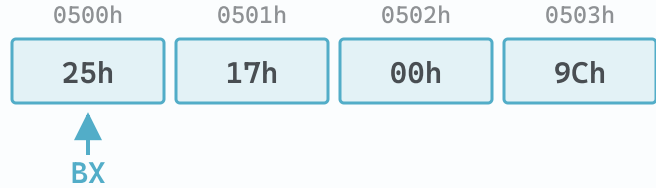
DS kesimi



Dikkat: MOV BL, [500h] BL'ye 500h *sayısını* yazmaz; DS:0500h adresindeki **baytı** okur. Sayıyı yazmak için köşeli parantez kullanılmaz: MOV BX, 0500h .

BX ile Dolaylı Erişim

```
► MOV BX, 0500h ; BX = adres
2 MOV AL, [BX] ; AL = [0500h]
3 MOV DL, [0501h] ; doğrudan adres
4 ADD AL, DL ; AL = 25h + 17h
5 MOV [BX+2], AL ; [0502h] = AL
```



BX'e **0500h** sayısı yazılır; artık BX bir adres (işaretçi) olarak kullanılabilir.

Adım 1 / 5

DEĞİŞKENLER

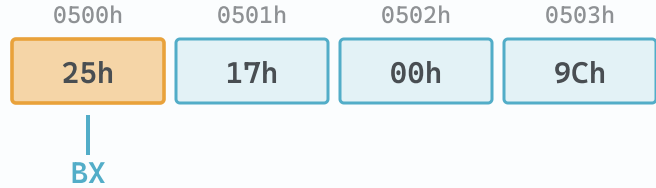
BX = 0500h

AL = 00h

DL = 00h

BX ile Dolaylı Erişim

```
1 MOV BX, 0500h    ; BX = adres
▶ MOV AL, [BX]      ; AL = [0500h]
3 MOV DL, [0501h]   ; doğrudan adres
4 ADD AL, DL        ; AL = 25h + 17h
5 MOV [BX+2], AL    ; [0502h] = AL
```



[BX] : BX'in gösterdiği bayt okunur → AL = **25h**.

Adım 2 / 5

DEĞİŞKENLER

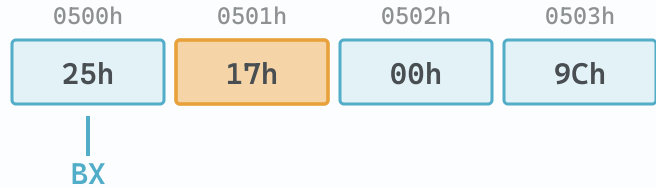
BX = 0500h

AL = 25h

DL = 00h

BX ile Dolaylı Erişim

```
1 MOV BX, 0500h    ; BX = adres
2 MOV AL, [BX]      ; AL = [0500h]
▶ 3 MOV DL, [0501h] ; doğrudan adres
4 ADD AL, DL        ; AL = 25h + 17h
5 MOV [BX+2], AL    ; [0502h] = AL
```



Aynı türden erişim adres sabit verilerek de yapılabilir: DL = [0501h] = **17h**.

Adım 3 / 5

DEĞİŞKENLER

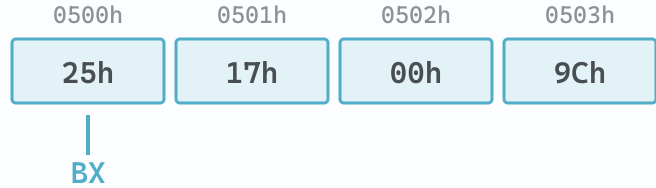
BX = 0500h

AL = 25h

DL = 17h

BX ile Dolaylı Erişim

```
1 MOV BX, 0500h    ; BX = adres
2 MOV AL, [BX]      ; AL = [0500h]
3 MOV DL, [0501h]   ; doğrudan adres
▶ 4 ADD AL, DL       ; AL = 25h + 17h
5 MOV [BX+2], AL    ; [0502h] = AL
```



AL = 25h + 17h = **3Ch**.

Adım 4 / 5

DEĞİŞKENLER

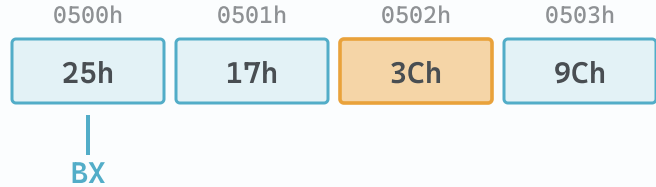
BX = 0500h

AL = 3Ch

DL = 17h

BX ile Dolaylı Erişim

```
1 MOV BX, 0500h    ; BX = adres
2 MOV AL, [BX]      ; AL = [0500h]
3 MOV DL, [0501h]   ; doğrudan adres
4 ADD AL, DL        ; AL = 25h + 17h
▶ MOV [BX+2], AL    ; [0502h] = AL
```



[BX+2] = 0502h adresine AL yazılır. Bellek değişti, BX değişmedi.

Adım 5 / 5

DEĞİŞKENLER

BX = 0500h

AL = 3Ch

DL = 17h

CX — Sayaç (Counter)

- 16 bit; **CH** ve **CL** olarak 8 bitlik iki yarıya ayrılır.
- Döngü (**LOOP**), tekrar (**REP**) ve sayma görevlerinde kullanılır.
- **LOOP etiket** : önce **CX** ← **CX** - 1, sonra **CX** ≠ 0 ise etikete atlar. Bayrakları etkilemez.
- Kaydırma ve döndürme (shift/rotate) komutlarında sayı 1'den büyükse **CL**'de verilir: **SHL AX, CL** .
- **JCXZ** : **CX** = 0 ise atla (boş döngüyü korur).

```
MOV CX, 0005h    ; 5 tur
DONGU:
    ; ... gövde ...
    LOOP DONGU    ; CX--, CX≠0 ise atla
```

Tuzak: **CX** = 0 iken döngüye girilirse ilk **LOOP CX**'i FFFFh yapar ve döngü **65536** kez döner.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
► MOV CX, 5      ; sayaç
2 MOV AX, 0      ; toplam
3 TEKRAR: ADD AX, CX
4 LOOP TEKRAR    ; CX--, ≠0 ise atla
5 MOV DX, AX     ; sonuç
```

CX = 5: döngü 5 kez dönecek.

Adım 1 / 13

DEĞİŞKENLER

CX = 0005h

AX = 0000h

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1
değerleriyle çalışır; CX = 0 değeriyle
çalışmaz.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5      ; sayaç
▶ MOV AX, 0      ; toplam
3 TEKRAR: ADD AX, CX
4 LOOP TEKRAR    ; CX--, ≠0 ise atla
5 MOV DX, AX     ; sonuç
```

Toplam AX = 0 ile başlar.

Adım 2 / 13

DEĞİŞKENLER

CX = 0005h

AX = 0000h

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle çalışmaz.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
▶ TEKRAR: ADD AX, CX
4 LOOP TEKRAR        ; CX--, ≠0 ise atla
5 MOV DX, AX         ; sonuç
```

$AX = AX + CX \rightarrow 0 + 5 = 5.$

Adım 3 / 13

DEĞİŞKENLER

CX = 0005h

AX = 0005h

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle çalışmaz.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5      ; sayaç
2 MOV AX, 0      ; toplam
3 TEKRAR: ADD AX, CX
▶ LOOP TEKRAR    ; CX--, ≠0 ise atla
5 MOV DX, AX     ; sonuç
```

LOOP: CX = 5 → 4; sıfır değil, **TEKRAR**'a atla.

Adım 4 / 13

DEĞİŞKENLER

CX = 0004h

AX = 0005h

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle **çalışmaz**.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
▶ TEKRAR: ADD AX, CX
4 LOOP TEKRAR        ; CX--, ≠0 ise atla
5 MOV DX, AX         ; sonuç
```

$AX = AX + CX \rightarrow 5 + 4 = 9.$

Adım 5 / 13

DEĞİŞKENLER

CX = 0004h

AX = 0009h

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle çalışmaz.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
3 TEKRAR: ADD AX, CX
▶ LOOP TEKRAR        ; CX--, ≠0 ise atla
5 MOV DX, AX         ; sonuç
```

LOOP: CX = 4 → 3; sıfır değil, **TEKRAR**'a atla.

Adım 6 / 13

DEĞİŞKENLER

CX = 0003h

AX = 0009h

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle **çalışmaz**.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
▶ TEKRAR: ADD AX, CX
4 LOOP TEKRAR        ; CX--, ≠0 ise atla
5 MOV DX, AX         ; sonuç
```

$AX = AX + CX \rightarrow 9 + 3 = 12.$

Adım 7 / 13

DEĞİŞKENLER

CX = 0003h

AX = 000Ch

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle çalışmaz.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
3 TEKRAR: ADD AX, CX
▶ LOOP TEKRAR        ; CX--, ≠0 ise atla
5 MOV DX, AX         ; sonuç
```

LOOP: CX = 3 → 2; sıfır değil, **TEKRAR**'a atla.

Adım 8 / 13

DEĞİŞKENLER

CX = 0002h

AX = 000Ch

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle **çalışmaz**.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5      ; sayaç
2 MOV AX, 0      ; toplam
▶ TEKRAR: ADD AX, CX
4 LOOP TEKRAR    ; CX--, ≠0 ise atla
5 MOV DX, AX     ; sonuç
```

$AX = AX + CX \rightarrow 12 + 2 = 14.$

Adım 9 / 13

DEĞİŞKENLER

CX = 0002h

AX = 000Eh

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle çalışmaz.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
3 TEKRAR: ADD AX, CX
▶ LOOP TEKRAR        ; CX--, ≠0 ise atla
5 MOV DX, AX         ; sonuç
```

LOOP: CX = 2 → 1; sıfır değil, **TEKRAR**'a atla.

Adım 10 / 13

DEĞİŞKENLER

CX = 0001h

AX = 000Eh

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle **çalışmaz**.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
▶ TEKRAR: ADD AX, CX
4 LOOP TEKRAR        ; CX--, ≠0 ise atla
5 MOV DX, AX         ; sonuç
```

$AX = AX + CX \rightarrow 14 + 1 = 15$.

Adım 11 / 13

DEĞİŞKENLER

CX = 0001h

AX = 000Fh

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle çalışmaz.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
3 TEKRAR: ADD AX, CX
▶ LOOP TEKRAR        ; CX--, ≠0 ise atla
5 MOV DX, AX         ; sonuç
```

LOOP: CX = 1 → 0; atlamaz, döngüden çıkılır.

Adım 12 / 13

DEĞİŞKENLER

CX = 0000h

AX = 000Fh

DX = 0000h

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle çalışmaz.

CX ve LOOP: 5 + 4 + 3 + 2 + 1

```
1 MOV CX, 5          ; sayaç
2 MOV AX, 0          ; toplam
3 TEKRAR: ADD AX, CX
4 LOOP TEKRAR        ; CX--, ≠0 ise atla
▶ MOV DX, AX         ; sonuç
```

Sonuç DX'e kopyalanır: $5+4+3+2+1 = 15 = \mathbf{000Fh}$.

Adım 13 / 13

DEĞİŞKENLER

CX = 0000h

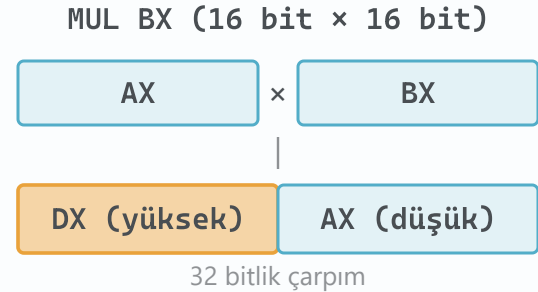
AX = 000Fh

DX = 000Fh

Döngü gövdesi CX = 5, 4, 3, 2, 1 değerleriyle çalışır; CX = 0 değeriyle çalışmaz.

DX — Veri (Data)

- 16 bit; **DH** ve **DL** olarak 8 bitlik iki yarıya ayrılır.
- 16 bitlik çarpmada sonuç 32 bittir: **DX** yüksek sırayı, **AX** düşük sırayı tutar → **DX:AX**.
- 16 bitlik bölmede bölünen **DX:AX** 'tir; bölüm AX'e, kalan DX'e yazılır.
- 255'ten büyük G/Ç port numaraları DX ile verilir: **OUT DX, AL**.
- DOS kesmelerinde (INT 21h) dize adresini taşır: **LEA DX, mesaj**.



Örnek: `MUL BX` → `DX:AX = AX × BX`. 8 bitlik `MUL BL` ise `AX = AL × BL` üretir; DX'e dokunmaz.

MUL ile DX:AX

```
► MOV AX, 1234h
2 MOV BX, 0100h
3 MUL BX          ; DX:AX = AX*BX
4 MOV AX, 00FFh
5 MOV BL, 2
6 MUL BL          ; AX = AL*BL
7 MOV DX, 03F8h   ; port adresi
8 OUT DX, AL      ; porta yaz
```

AX = 1234h.

Adım 1 / 8

DEĞİŞKENLER

AX = 1234h

BX = 0000h

DX = 0000h

CF = 0

OF = 0

MUL işaretsiz çarpmadır; işaretli için

IMUL kullanılır. MUL sonrası SF, ZF, PF, AF **tanımsızdır**.

MUL ile DX:AX

```
1 MOV AX, 1234h
▶ MOV BX, 0100h
3 MUL BX          ; DX:AX = AX*BX
4 MOV AX, 00FFh
5 MOV BL, 2
6 MUL BL          ; AX = AL*BL
7 MOV DX, 03F8h   ; port adresi
8 OUT DX, AL      ; porta yaz
```

BX = 0100h (256).

Adım 2 / 8

DEĞİŞKENLER

AX = 1234h

BX = 0100h

DX = 0000h

CF = 0

OF = 0

MUL işaretsiz çarpmadır; işaretli için

IMUL kullanılır. MUL sonrası SF, ZF, PF, AF **tanımsızdır**.

MUL ile DX:AX

```
1 MOV AX, 1234h
2 MOV BX, 0100h
▶ MUL BX ; DX:AX = AX*BX
4 MOV AX, 00FFh
5 MOV BL, 2
6 MUL BL ; AX = AL*BL
7 MOV DX, 03F8h ; port adresi
8 OUT DX, AL ; porta yaz
```

$1234h \times 0100h = \mathbf{00123400h} \rightarrow DX = 0012h, AX = 3400h$. Yüksek yarı sıfır olmadığı için $CF = OF = 1$.

Adım 3 / 8

DEĞİŞKENLER

AX = **3400h**

BX = **0100h**

DX = **0012h**

CF = **1**

OF = **1**

MUL işaretsiz çarpmadır; işaretli için

IMUL kullanılır. MUL sonrası SF, ZF, PF, AF **tanımsızdır**.

MUL ile DX:AX

```
1 MOV AX, 1234h
2 MOV BX, 0100h
3 MUL BX      ; DX:AX = AX*BX
▶ 4 MOV AX, 00FFh
5 MOV BL, 2
6 MUL BL      ; AX = AL*BL
7 MOV DX, 03F8h ; port adresi
8 OUT DX, AL   ; porta yaz
```

AX = 00FFh; DX'teki eski sonuç duruyor.

Adım 4 / 8

DEĞİŞKENLER

AX = 00FFh

BX = 0100h

DX = 0012h

CF = 1

OF = 1

MUL işaretsiz çarpmadır; işaretli için

IMUL kullanılır. MUL sonrası SF, ZF, PF, AF **tanımsızdır**.

MUL ile DX:AX

```
1 MOV AX, 1234h
2 MOV BX, 0100h
3 MUL BX          ; DX:AX = AX*BX
4 MOV AX, 00FFh
▶ MOV BL, 2
6 MUL BL          ; AX = AL*BL
7 MOV DX, 03F8h   ; port adresi
8 OUT DX, AL      ; porta yaz
```

BL = 02h (BX = 0102h, BH değişmedi).

Adım 5 / 8

DEĞİŞKENLER

AX = 00FFh

BX = 0102h

DX = 0012h

CF = 1

OF = 1

MUL işaretsiz çarpmadır; işaretli için

IMUL kullanılır. MUL sonrası SF, ZF, PF, AF **tanımsızdır**.

MUL ile DX:AX

```
1 MOV AX, 1234h
2 MOV BX, 0100h
3 MUL BX          ; DX:AX = AX*BX
4 MOV AX, 00FFh
5 MOV BL, 2
▶ MUL BL          ; AX = AL*BL
7 MOV DX, 03F8h   ; port adresi
8 OUT DX, AL      ; porta yaz
```

8 bitlik çarpım: $FFh \times 02h = \mathbf{01FEh} \rightarrow AX$. DX değişmez;
AH $\neq 0$ olduğundan CF = OF = 1.

Adım 6 / 8

DEĞİŞKENLER

AX = **01FEh**

BX = **0102h**

DX = **0012h**

CF = **1**

OF = **1**

MUL işaretsiz çarpmadır; işaretli için

IMUL kullanılır. MUL sonrası SF, ZF, PF, AF **tanımsızdır**.

MUL ile DX:AX

```
1 MOV AX, 1234h
2 MOV BX, 0100h
3 MUL BX          ; DX:AX = AX*BX
4 MOV AX, 00FFh
5 MOV BL, 2
6 MUL BL          ; AX = AL*BL
▶ MOV DX, 03F8h   ; port adresi
8 OUT DX, AL      ; porta yaz
```

DX bu kez port numarası: 03F8h (COM1 seri port).

Adım 7 / 8

DEĞİŞKENLER

AX = 01FEh

BX = 0102h

DX = 03F8h

CF = 1

OF = 1

MUL işaretsiz çarpmadır; işaretli için

IMUL kullanılır. MUL sonrası SF, ZF, PF, AF **tanımsızdır**.

MUL ile DX:AX

```
1 MOV AX, 1234h
2 MOV BX, 0100h
3 MUL BX          ; DX:AX = AX*BX
4 MOV AX, 00FFh
5 MOV BL, 2
6 MUL BL          ; AX = AL*BL
7 MOV DX, 03F8h   ; port adresi
▶ OUT DX, AL      ; porta yaz
```

AL'deki FEh, 03F8h numaralı porta gönderilir. 255'ten büyük port için DX şarttır.

Adım 8 / 8

DEĞİŞKENLER

AX = 01FEh

BX = 0102h

DX = 03F8h

CF = 1

OF = 1

MUL işaretsiz çarpmadır; işaretli için **IMUL** kullanılır. MUL sonrası SF, ZF, PF, AF **tanımsızdır**.

Kod İzleme: DIV ile Bölüm ve Kalan

```
1 org 100h
▶ mov ax, 805      ; sayı
▶ mov bx, 10       ; bölen
▶ mov cx, 0        ; basamak sayısı
5 ayir: mov dx, 0   ; DX:AX
6 div bx           ; bölüm, kalan
7 push dx          ; basamağı sakla
8 inc cx
9 cmp ax, 0
10 jne ayir
11 hlt
```

AX

805

BX

10

yığın (SS:SP)

CX (basamak)

0

AX = **805** basamaklarına ayrılacak. BX = 10 bölen; CX basamakları sayacak.

Adım 1 / 7

DEĞİŞKENLER

AX = **805**DX = **0**CX = **0**yığın = **[]**

Kod İzleme: DIV ile Bölüm ve Kalan

```
1 org 100h
2 mov ax, 805      ; sayı
3 mov bx, 10       ; bölen
4 mov cx, 0        ; basamak sayısı
5 ▶ ayir: mov dx, 0 ; DX:AX
6 ▶ div bx         ; bölüm, kalan
7 push dx          ; basamağı sakla
8 inc cx
9 cmp ax, 0
10 jne ayir
11 hlt
```

DX:AX ÷ BX

805 ÷ 10

AX (bölüm)

80

DX (kalan)

5

yığın (SS:SP)

div bx : 32 bitlik **DX:AX** 16 bitlik BX'e bölünür. DX önce 0 yapılmalı; yoksa eski kalan bölünenin yüksek yarısı olur.
Bölüm AX = **80**, kalan DX = **5** (birler basamak).

Adım 2 / 7

DEĞİŞKENLERAX = **80**DX = **5**CX = **0**yığın = **[]**

Kod İzleme: DIV ile Bölüm ve Kalan

```
1 org 100h
2 mov ax, 805      ; sayı
3 mov bx, 10       ; bölen
4 mov cx, 0        ; basamak sayısı
5 ayir: mov dx, 0   ; DX:AX
6 div bx           ; bölüm, kalan
▶ push dx          ; basamağı sakla
▶ inc cx
▶ cmp ax, 0
▶ jne ayir
11 hlt
```



Kalan yığına itildi, CX = 1. AX $\neq$ 0 → bir basamak daha var.

Adım 3 / 7

DEĞİŞKENLER

AX = 80

DX = 5

CX = 1

yığın = [5]

Kod İzleme: DIV ile Bölüm ve Kalan

```
1 org 100h
2 mov ax, 805      ; sayı
3 mov bx, 10       ; bölen
4 mov cx, 0        ; basamak sayısı
5 ▶ ayır: mov dx, 0 ; DX:AX
6 ▶ div bx         ; bölüm, kalan
7 push dx          ; basamağı sakla
8 inc cx
9 cmp ax, 0
10 jne ayır
11 hlt
```

DX:AX ÷ BX

80 ÷ 10

AX (bölüm)

8

DX (kalan)

0

yığın (SS:SP)

5

← SP

Bölüm AX = **8**, kalan DX = **0** (onlar basamak).

Adım 4 / 7

DEĞİŞKENLERAX = **8**DX = **0**CX = **1**yığın = [**5**]

Kod İzleme: DIV ile Bölüm ve Kalan

```
1 org 100h
2 mov ax, 805      ; sayı
3 mov bx, 10       ; bölen
4 mov cx, 0        ; basamak sayısı
5 ayir: mov dx, 0   ; DX:AX
6 div bx           ; bölüm, kalan
▶ push dx          ; basamağı sakla
▶ inc cx
▶ cmp ax, 0
▶ jne ayir
11 hlt
```



Kalan yığına itildi, CX = 2. AX $\neq$ 0 → bir basamak daha var.

Adım 5 / 7

DEĞİŞKENLER

AX = 8

DX = 0

CX = 2

yığın = [5, 0]

Kod İzleme: DIV ile Bölüm ve Kalan

```
1 org 100h
2 mov ax, 805      ; sayı
3 mov bx, 10       ; bölen
4 mov cx, 0        ; basamak sayısı
5 ▶ ayir: mov dx, 0 ; DX:AX
6 ▶ div bx         ; bölüm, kalan
7 push dx          ; basamağı sakla
8 inc cx
9 cmp ax, 0
10 jne ayir
11 hlt
```

DX:AX ÷ BX

8 ÷ 10

AX (bölüm)

0

DX (kalan)

8

yığın (SS:SP)

0

← SP

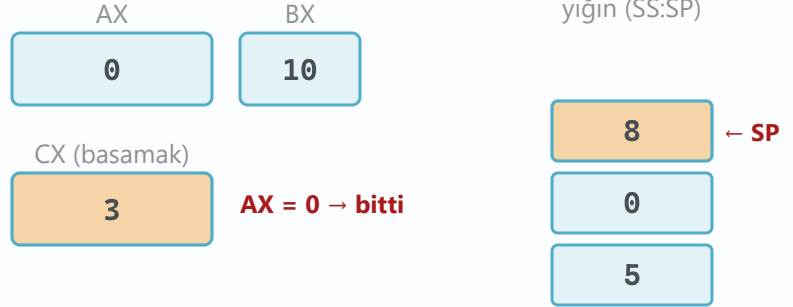
5Bölüm AX = **0**, kalan DX = **8** (yüzler basamak).

Adım 6 / 7

DEĞİŞKENLERAX = **0**DX = **8**CX = **2**yığın = [**5**, **0**]

Kod İzleme: DIV ile Bölüm ve Kalan

```
1 org 100h
2 mov ax, 805      ; sayı
3 mov bx, 10       ; bölen
4 mov cx, 0        ; basamak sayısı
5 ayir: mov dx, 0   ; DX:AX
6 div bx           ; bölüm, kalan
▶ push dx          ; basamağı sakla
▶ inc cx
▶ cmp ax, 0
▶ jne ayir
11 hlt
```



Kalan yığına itildi, CX = 3. AX = 0: bitti. Yığında üstten **8, 0, 5**: POP sırası basamakları soldan sağa verir.

Adım 7 / 7

DEĞİŞKENLER

AX = 0

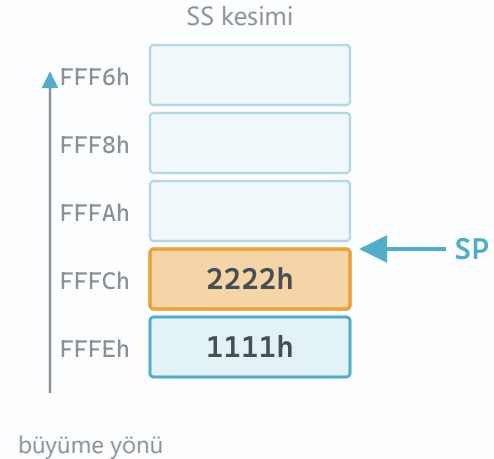
DX = 8

CX = 3

yığın = [5, 0, 8]

SP — Yığın İşaretçisi

- 16 bitlik **Stack Pointer**; yığının **en üst** ögesinin bağıl konumunu (offset) tutar.
- Adres her zaman **yığın kesimine** göre dir: fiziksel adres `SS:SP`.
- Yığın **LIFO** yapısıdır: `PUSH` ile eklenir, `POP` ile çıkarılır; 8086'da ögeler hep **16 bittir**.
- Yığın belleğin **aşağısına doğru** büyür: `PUSH` önce SP'yi 2 azaltır, sonra yazar; `POP` önce okur, sonra SP'yi 2 artırır.
- "Yığın boşken SP = FFFEh" bilgisi **.COM** programları içindir (DOS/emu8086 SP'yi FFFEh yapar); genel kural değildir.



İŞARETÇİ VE İNDİS YAZMAÇLARI · KODU ADIM ADIM ÇALIŞTIR

PUSH / POP ile SP

```
► MOV AX, 1111h
2 MOV BX, 2222h
3 PUSH AX      ; SP-=2, [SP]=AX
4 PUSH BX
5 POP AX       ; AX=[SP], SP+=2
6 POP BX       ; değerler takas oldu
```

ÇAĞRI YIĞINI

boş



AX = 1111h. Yığın boş: SP = FFFEh (.COM programı).

Adım 1 / 6

DEĞİŞKENLER

AX = 1111h

BX = 0000h

SP = FFFEh

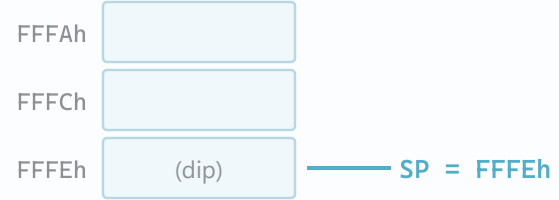
İŞARETÇİ VE İNDİS YAZMAÇLARI · KODU ADIM ADIM ÇALIŞTIR

PUSH / POP ile SP

```
1 MOV AX, 1111h
▶ MOV BX, 2222h
3 PUSH AX      ; SP-=2, [SP]=AX
4 PUSH BX
5 POP AX       ; AX=[SP], SP+=2
6 POP BX       ; değerler takas oldu
```

ÇAĞRI YIĞINI

boş



BX = 2222h.

Adım 2 / 6

DEĞİŞKENLER

AX = 1111h

BX = 2222h

SP = FFFEh

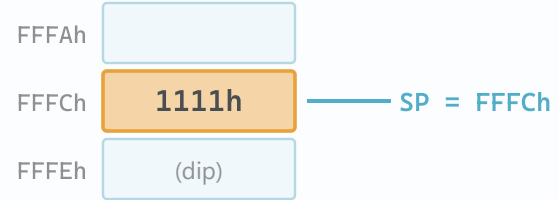
İŞARETÇİ VE İNDİS YAZMAÇLARI · KODU ADIM ADIM ÇALIŞTIR

PUSH / POP ile SP

```
1 MOV AX, 1111h
2 MOV BX, 2222h
▶ 3 PUSH AX      ; SP-=2, [SP]=AX
4 PUSH BX
5 POP AX         ; AX=[SP], SP+=2
6 POP BX         ; değerler takas oldu
```

ÇAĞRI YIĞINI

[FFFCh] = 1111h



PUSH AX: $SP = FFFEh - 2 = \mathbf{FFFCh}$, sonra [FFFCh] = 1111h.

Adım 3 / 6

DEĞİŞKENLER

AX = 1111h

BX = 2222h

SP = FFFCh

İŞARETÇİ VE İNDİS YAZMAÇLARI · KODU ADIM ADIM ÇALIŞTIR

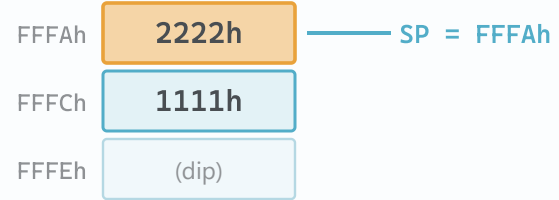
PUSH / POP ile SP

```
1 MOV AX, 1111h
2 MOV BX, 2222h
3 PUSH AX      ; SP-=2, [SP]=AX
4 ► PUSH BX
5 POP AX       ; AX=[SP], SP+=2
6 POP BX       ; değerler takas oldu
```

ÇAĞRI YIĞINI

[FFFAh] = 2222h

[FFFCh] = 1111h



PUSH BX: SP = **FFFAh**, [FFFAh] = 2222h. En üstte 2222h var.

Adım 4 / 6

DEĞİŞKENLER

AX = 1111h

BX = 2222h

SP = FFFAh

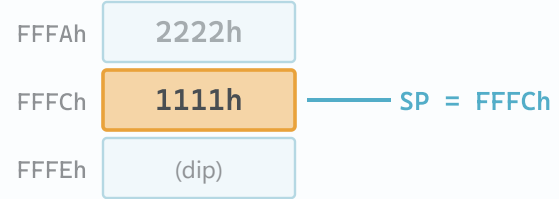
İŞARETÇİ VE İNDİS YAZMAÇLARI · KODU ADIM ADIM ÇALIŞTIR

PUSH / POP ile SP

```
1 MOV AX, 1111h
2 MOV BX, 2222h
3 PUSH AX      ; SP-=2, [SP]=AX
4 PUSH BX
5 ► POP AX     ; AX=[SP], SP+=2
6 POP BX      ; değerler takas oldu
```

ÇAĞRI YIĞINI

[FFFCh] = 1111h



POP AX: en üstteki 2222h AX'e okunur, SP = FFFCh. Eski hücredeki veri silinmez, sadece **geçersiz** sayılır.

Adım 5 / 6

DEĞİŞKENLER

AX = 2222h

BX = 2222h

SP = FFFCh

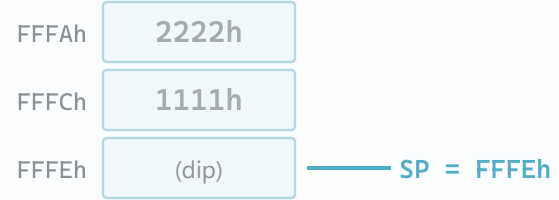
İŞARETÇİ VE İNDİS YAZMAÇLARI · KODU ADIM ADIM ÇALIŞTIR

PUSH / POP ile SP

```
1 MOV AX, 1111h
2 MOV BX, 2222h
3 PUSH AX      ; SP-=2, [SP]=AX
4 PUSH BX
5 POP AX       ; AX=[SP], SP+=2
▶ POP BX      ; değerler takas oldu
```

ÇAĞRI YIĞINI

boş



POP BX: 1111h BX'e, SP = FFEh. LIFO sayesinde AX ile BX **takas** edildi.

Adım 6 / 6

DEĞİŞKENLER

AX = 2222h

BX = 1111h

SP = FFEh

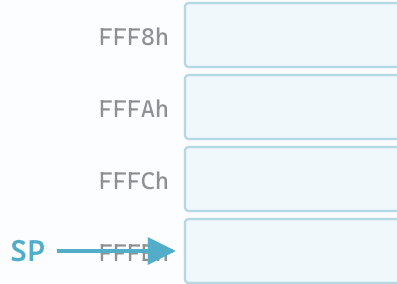
BP — Taban İşaretçisi

- 16 bitlik **Base Pointer**; yığın üzerindeki **sabit bir konumu** gösterir.
- BP ile yapılan bellek erişimleri varsayılan olarak **SS** kesimini kullanır: `[BP+4]` → `SS:BP+4`.
- Yığın üzerinden fonksiyonlara (prosedürlere) geçirilen **parametrelere** ve yerel değişkenlere erişmek için kullanılır.
- SP, PUSH/POP ile sürekli değişir; BP ise prosedür boyunca **sabit** kalır, bu yüzden parametrelerin konumu bellidir.

```
TOPLA PROC
    PUSH BP
    MOV BP, SP ; çerçeve tabanı
    MOV AX, [BP+4] ; 1. parametre
    POP BP
    RET 2
TOPLA ENDP
```

Klasik prosedür girişi: `PUSH BP` +
`MOV BP, SP`.

BP ile Parametreye Erişim

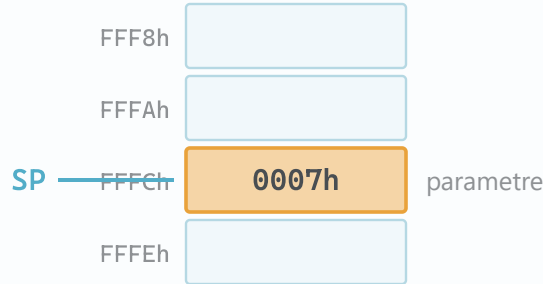


Başlangıç: yığın boş, SP = FFFEh. Çağırın, parametre olarak AX = 0007h'yi geçirecek.

Adım 1 / 6

Prosedürler ve CALL/RET ayrıntısı Bölüm 7 ve 8'de; burada BP'nin neden gerektiği gösteriliyor.

BP ile Parametreye Erişim

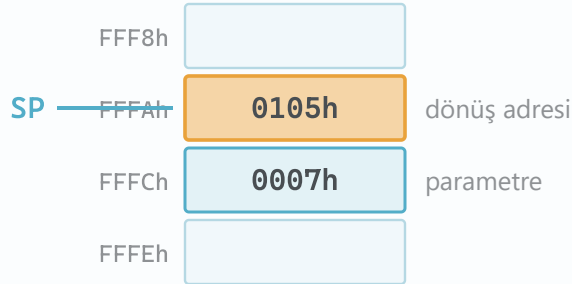


PUSH AX : parametre yığına konur; SP = FFFCh.

Adım 2 / 6

Prosedürler ve CALL/RET ayrıntısı Bölüm 7 ve 8'de; burada BP'nin neden gerektiği gösteriliyor.

BP ile Parametreye Erişim

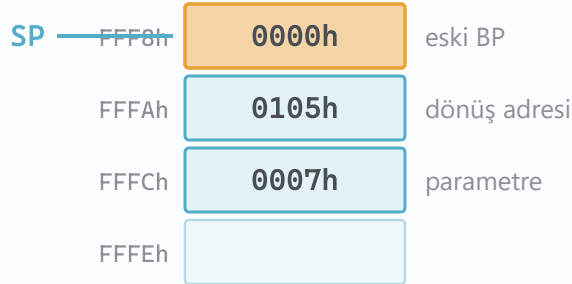


CALL TOPLA : dönüş adresi (sonraki komutun IP'si, ör. 0105h) yığına konur; SP = FFFAh.

Adım 3 / 6

Prosedürler ve CALL/RET ayrıntısı Bölüm 7 ve 8'de; burada BP'nin neden gerektiği gösteriliyor.

BP ile Parametreye Erişim

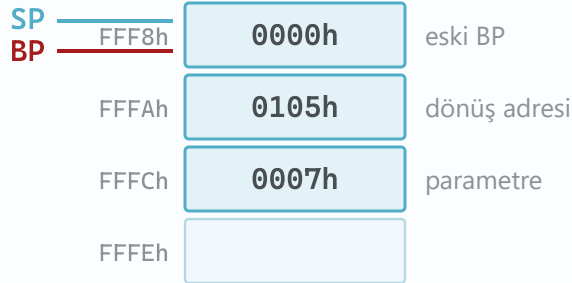


PUSH BP : çağıranın BP değeri saklanır; SP = FFF8h.

Adım 4 / 6

Prosedürler ve CALL/RET ayrıntısı Bölüm 7 ve 8'de; burada BP'nin neden gerektiği gösteriliyor.

BP ile Parametreye Erişim

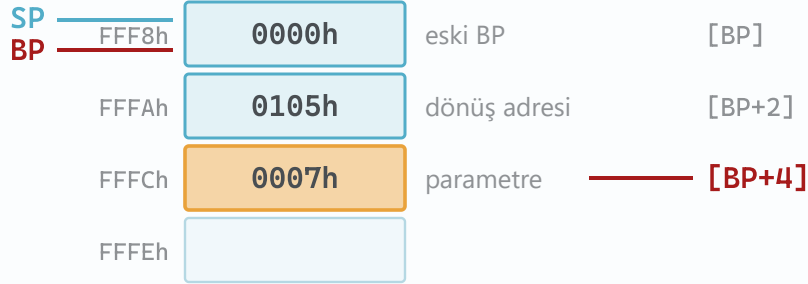


`MOV BP, SP` : BP = FFF8h. Bundan sonra SP değişse de BP sabit kalır.

Adım 5 / 6

Prosedürler ve CALL/RET ayrıntısı Bölüm 7 ve 8'de; burada BP'nin neden gerektiği gösteriliyor.

BP ile Parametreye Erişim



`MOV AX, [BP+4]` : $\text{FFF8h} + 4 = \text{FFFC}$ → parametre **0007h** okunur. [BP+2] dönüş adresidir.

Adım 6 / 6

Prosedürler ve CALL/RET ayrıntısı Bölüm 7 ve 8'de; burada BP'nin neden gerektiği gösteriliyor.

SI ve DI — İndis Yazmaçları

SI — Source Index (kaynak indis)

- 16 bit; veri kesiminde (**DS**) bağlı konumu temsil eder.
- Okunacak verinin konumunu işaret eder.
- Dize komutlarında **kaynak**: `MOVSB`, `CMPBSB`, `LODSB` → `DS:SI`.
- Örnek: `MOV SI, 2000h` — SI'yi bir bellek konumuna ayarlar.

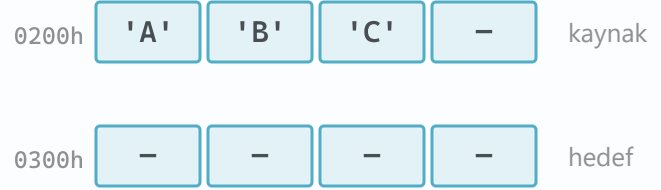
DI — Destination Index (hedef indis)

- 16 bit; dize komutlarında ekstra kesimde (**ES**) bağlı konumu temsil eder.
- Yazılacak verinin konumunu işaret eder.
- Dize komutlarında **hedef**: `MOVSW`, `STOSB`, `SCASB` → `ES:DI`.
- Örnek: `MOV DI, 3000h` — DI'yi bir bellek konumuna ayarlar.

Dize komutu dışında `[SI]` ve `[DI]` sıradan indis olarak **DS** kesimini kullanır. Her dize adımından sonra SI ve DI, **DF** bayrağına göre 1 (bayt) ya da 2 (word) artar veya azalır.

SI ve DI ile Dize Kopyalama

```
► CLD ; DF=0: artan adres
2 MOV SI, 0200h ; kaynak
3 MOV DI, 0300h ; hedef
4 MOV CX, 3 ; bayt sayısı
5 REP MOVSB ; ES:[DI] ← DS:[SI]
```



CLD: DF = 0 → SI ve DI her adımda **artacak**.

Adım 1 / 7

DEĞİŞKENLER

SI = 0000h

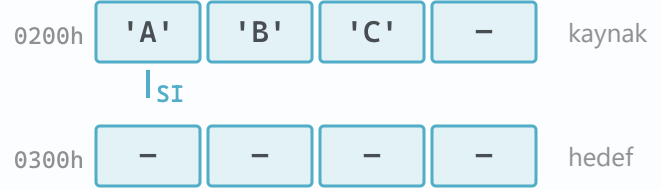
DI = 0000h

CX = 0000h

DF = 0

SI ve DI ile Dize Kopyalama

```
1 CLD ; DF=0: artan adres
▶ 2 MOV SI, 0200h ; kaynak
3 MOV DI, 0300h ; hedef
4 MOV CX, 3 ; bayt sayısı
5 REP MOVSB ; ES:[DI] ← DS:[SI]
```



SI = 0200h: kopyalanacak "ABC" dizesinin başı.

Adım 2 / 7

DEĞİŞKENLER

SI = 0200h

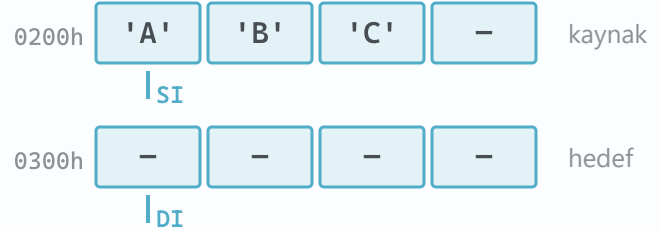
DI = 0000h

CX = 0000h

DF = 0

SI ve DI ile Dize Kopyalama

```
1 CLD ; DF=0: artan adres
2 MOV SI, 0200h ; kaynak
3 MOV DI, 0300h ; hedef
4 MOV CX, 3 ; bayt sayısı
5 REP MOVSB ; ES:[DI] ← DS:[SI]
```



DI = 0300h: hedef alanın başı.

Adım 3 / 7

DEĞİŞKENLER

SI = 0200h

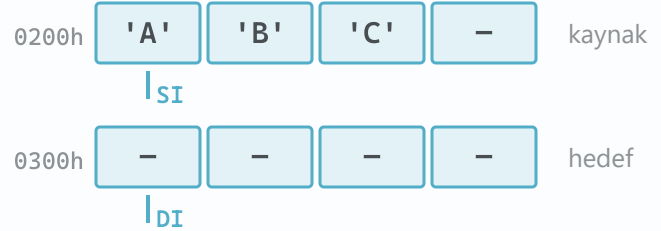
DI = 0300h

CX = 0000h

DF = 0

SI ve DI ile Dize Kopyalama

```
1 CLD ; DF=0: artan adres
2 MOV SI, 0200h ; kaynak
3 MOV DI, 0300h ; hedef
▶ MOV CX, 3 ; bayt sayısı
5 REP MOVSB ; ES:[DI] ← DS:[SI]
```



CX = 3: REP en fazla 3 kez tekrarlar.

Adım 4 / 7

DEĞİŞKENLER

SI = 0200h

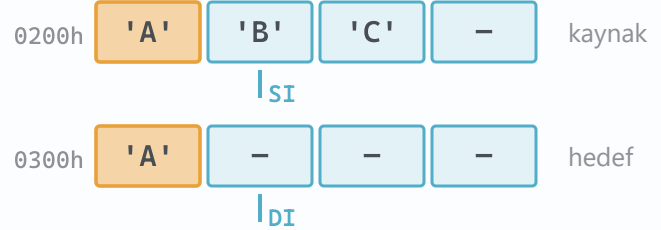
DI = 0300h

CX = 0003h

DF = 0

SI ve DI ile Dize Kopyalama

```
1 CLD ; DF=0: artan adres
2 MOV SI, 0200h ; kaynak
3 MOV DI, 0300h ; hedef
4 MOV CX, 3 ; bayt sayısı
▶ REP MOVSB ; ES:[DI] ← DS:[SI]
```



1. tur: 'A' kopyalanır; SI = 0201h, DI = 0301h, CX = 2.

Adım 5 / 7

DEĞİŞKENLER

SI = 0201h

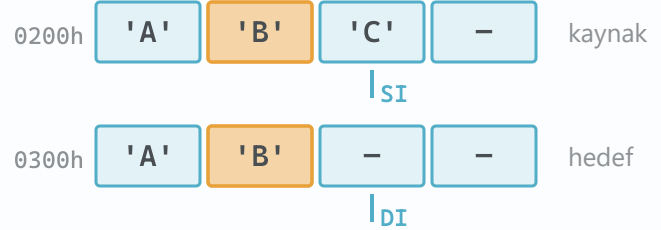
DI = 0301h

CX = 0002h

DF = 0

SI ve DI ile Dize Kopyalama

```
1 CLD ; DF=0: artan adres
2 MOV SI, 0200h ; kaynak
3 MOV DI, 0300h ; hedef
4 MOV CX, 3 ; bayt sayısı
▶ REP MOVSB ; ES:[DI] ← DS:[SI]
```



2. tur: 'B' kopyalanır; SI = 0202h, DI = 0302h, CX = 1.

Adım 6 / 7

DEĞİŞKENLER

SI = 0202h

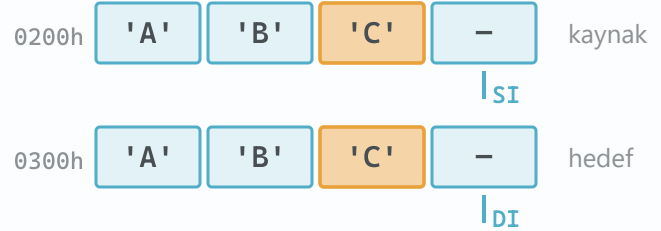
DI = 0302h

CX = 0001h

DF = 0

SI ve DI ile Dize Kopyalama

```
1 CLD ; DF=0: artan adres
2 MOV SI, 0200h ; kaynak
3 MOV DI, 0300h ; hedef
4 MOV CX, 3 ; bayt sayısı
▶ REP MOVSB ; ES:[DI] ← DS:[SI]
```



3. tur: 'C' kopyalanır; CX = 0 → REP biter.

Adım 7 / 7

DEĞİŞKENLER

SI = 0203h

DI = 0303h

CX = 0000h

DF = 0

Kesim Yazmaçları ve IP

- **CS** (kod), **DS** (veri), **SS** (yığın), **ES** (ekstra): her biri 64 KB'lık bir kesimin başlangıcını tutar.
- **IP** (Instruction Pointer): CS içinde çalıştırılacak **sonraki komutun** bağlı konumu.
- IP'ye **MOV** ile yazılamaz; yalnızca JMP, CALL, RET, kesmeler değiştirir.
- Kesim yazmacına anlık değer yüklenemez: **MOV DS, AX** gerekir.

Çift	Kullanım
CS:IP	sonraki komut
SS:SP	yığının üstü
SS:BP	yığın çerçevesi
DS:BX , DS:SI , DS:DI	veri
DS:SI → ES:DI	dize komutları

Fiziksel adres (20 bit) = kesim × 16 + bağlı konum. 16 ile çarpmak, onaltılık sayının sonuna bir 0 eklemektir.

Fiziksel Adres Hesabı

DS = 2000h

BX = 0500h

MOV AL, [BX]

Komut **[BX]** ile DS kesimindeki bir bayta erişmek istiyor.

Adım 1 / 4

Farklı kesim:konum çiftleri aynı fiziksel adresi verebilir: $2000h:0500h = 2050h:0000h = 1FF0h:0600h = 20500h$.

Fiziksel Adres Hesabı

DS = 2000h

20000h kesim × 16

BX = 0500h

MOV AL, [BX]

Kesim değeri 16 ile çarpılır (onaltılıkta sonuna 0 eklenir): 2000h → **20000h**.

Adım 2 / 4

Farklı kesim:konum çiftleri aynı fiziksel adresi verebilir: 2000h:0500h = 2050h:0000h = 1FF0h:0600h = 20500h.

Fiziksel Adres Hesabı

DS = 2000h

BX = 0500h

MOV AL, [BX]

20000h kesim × 16

+ 0500h bağlı konum (EA)

Bağlı konum (BX = 0500h) eklenir.

Adım 3 / 4

Farklı kesim:konum çiftleri aynı fiziksel adresi verebilir: $2000h:0500h = 2050h:0000h = 1FF0h:0600h = 20500h$.

Fiziksel Adres Hesabı

DS = 2000h

20000h kesim × 16

BX = 0500h

+ 0500h bağlı konum (EA)

MOV AL, [BX]

20500h 20 bitlik fiziksel adres

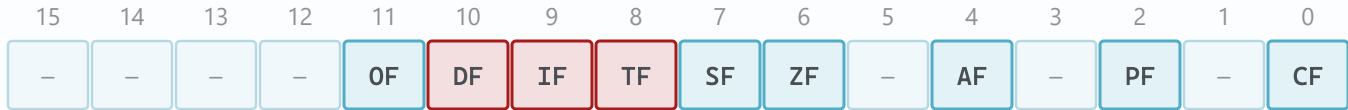
Fiziksel adres = **20500h**. Bu 20 bitlik değer adres yoluna (A0–A19) konur.

Adım 4 / 4

Farklı kesim:konum çiftleri aynı fiziksel adresi verebilir: $2000h:0500h = 2050h:0000h = 1FF0h:0600h = 20500h$.

Bayrak Yazmacı

- **Durum yazmacı** (status / FLAGS) olarak da adlandırılır; 16 bitlik yazmacın **9** bitini kullanılır.
- Mikroişlemcinin yürüttüğü **son aritmetik/mantık komutunun** sonucu hakkında bilgi içerir.
- **Koşullu atlama** ve dallanma komutlarının (JZ, JC, JG ...) davranışını belirler.



☐ durum bayrağı (sonuçtan etkilenir)

☐ kontrol bayrağı (programcı ayarlar)

6 durum bayrağı

CF, PF, AF, ZF, SF, OF: işlem sonucuna göre donanım tarafından kurulur (1) ya da temizlenir (0).

3 kontrol bayrağı

DF, IF, TF: işlemcinin *çalışma biçimini* değiştirir; programcı CLD/STD, CLI/STI gibi komutlarla ayarlar.

SF — İşaret Bayrağı

- Sign Flag, işlem sonucunun **negatif mi pozitif mi** olduğunu belirtir.
- Sonucun **en önemli biti** (MSB) kopyalanır:
- $MSB = 1 \rightarrow$ sayı (işaretli yorumda) negatiftir $\rightarrow$ **SF = 1**.
- $MSB = 0 \rightarrow$ sayı pozitif ya da sıfırdır $\rightarrow$ **SF = 0**.
- 8 bitte MSB bit 7, 16 bitte bit 15'tir.

Sonuç (8 bit)	MSB	SF
00h ... 7Fh	0	0
80h ... FFh	1	1
35h = 0011 0101b	0	0
C2h = 1100 0010b	1	1

İşlemci sayının işaretli mi işaretsiz mi olduğunu bilmez; SF yalnızca MSB'nin kopyasıdır. Sonucu işaretli yorumlamak programcının işidir (Bölüm 3: ikiye tümleyen).

ZF ve PF

ZF — Sıfır Bayrağı (Zero)

- İşlem sonucunun **sıfır** olup olmadığını belirtir.
- Sonuç 0 ise **ZF = 1**; değilse ZF = 0.
- `CMP AX, BX` eşitlikte ZF = 1 yapar → `JE / JZ` atlar.
- Örnek: `SUB AL, AL` → AL = 00h, ZF = 1.

PF — Eşlik Bayrağı (Parity)

- Sonucun **düşük 8 bitindeki** 1'lerin sayısı **çift** ise PF = 1; tek ise PF = 0.
- 16 bitlik işlemde de yalnızca düşük bayta bakılır.
- Örnek: 03h = 0000 0011b → iki 1 → PF = 1.
- Örnek: 07h = 0000 0111b → üç 1 → PF = 0.

03h

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

 2 tane 1 → çift → PF = 1

07h

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

 3 tane 1 → tek → PF = 0

Dikkat: ZF = 1 "sonuç sıfır" demektir; bayrağın değeri ile sonucun değeri ters görünür. PF eskiden seri iletişimde hata denetimi için kullanılırdı.

AF — Yardımcı Elde Bayrağı

- Auxiliary Carry Flag: işlem sırasında **bit 3'ten bit 4'e** elde (toplamada) ya da ödünç (çıkarmada) oluşursa **AF = 1**; aksi hâlde 0.
- Yani düşük **nibble**'dan (4 bit) taşma olup olmadığını gösterir.
- **BCD** (Binary Coded Decimal, her basamak 0–9) aritmetiğine özgüdür: **DAA** , **DAS** , **AAA** , **AAS** bu bayrağı kullanır.

08h	0000 1000
09h	+ 0000 1001
11h	0001 0001

↑ bit 3 → bit 4 eldesi

AF = 1 → DAA ile 17h (BCD 17)

AF'yi test eden **koşullu atlama komutu yoktur** ve AF'yi doğrudan kuran/silen komut yoktur; yine de LAHF , PUSHF ile okunabilir, SAHF , POPF ile yazılabilir.

Kod İzleme: AF ile BCD Düzeltmesi

```
1 org 100h
```

```
▶ mov al, 19h ; BCD 19
```

```
3 add al, 28h ; ikili toplama
```

```
4 daa ; BCD düzeltmesi
```

```
5 mov bl, al ; 47 sakla
```

```
6 mov al, 56h ; BCD 56
```

```
7 add al, 37h
```

```
8 daa
```

```
9 ret
```

AL

1

9

0001

1001

yüksek · düşük nibble

AF

0

CF

0

BL = 00h

ondalık: 19

Paketli BCD: her yarım bayt (nibble) bir ondalık basamak. AL = **19h** "19" demektir.

Adım 1 / 7

DEĞİŞKENLER

AL = 19h

AF = 0

CF = 0

BL = 00h

Kod İzleme: AF ile BCD Düzeltmesi

```
1 org 100h
2 mov al, 19h ; BCD 19
3 ► add al, 28h ; ikili toplama
4 daa ; BCD düzeltmesi
5 mov bl, al ; 47 sakla
6 mov al, 56h ; BCD 56
7 add al, 37h
8 daa
9 ret
```



İşlemci ikili toplar: $19h + 28h = 41h$. Düşük nibble'lar $9 + 8 = 17 > 15 \rightarrow$ bit 3'ten bit 4'e elde: **AF = 1**. 41h BCD olarak yanlış (doğrusu 47).

Adım 2 / 7

DEĞİŞKENLER

AL = 41h

AF = 1

CF = 0

BL = 00h

Kod İzleme: AF ile BCD Düzeltmesi

```
1 org 100h
2 mov al, 19h      ; BCD 19
3 add al, 28h      ; ikili toplama
4 ▶ daa            ; BCD düzeltmesi
5 mov bl, al       ; 47 sakla
6 mov al, 56h      ; BCD 56
7 add al, 37h
8 daa
9 ret
```



BL = 00h

 $41h + 06h = 47h$

ondalık: 47

daa : AF = 1 olduğu için düşük nibble'a +6 eklenir → 47h = 19 + 28 ✓.

Adım 3 / 7

DEĞİŞKENLER

AL = 47h

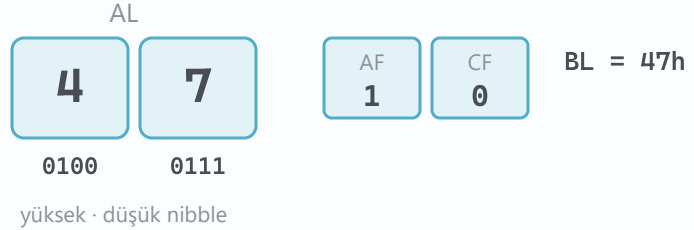
AF = 1

CF = 0

BL = 00h

Kod İzleme: AF ile BCD Düzeltmesi

```
1 org 100h
2 mov al, 19h      ; BCD 19
3 add al, 28h      ; ikili toplama
4 daa              ; BCD düzeltmesi
5 ► mov bl, al      ; 47 sakla
6 mov al, 56h      ; BCD 56
7 add al, 37h
8 daa
9 ret
```



ondalık: 47

Sonuç BL'ye saklandı: BL = 47h.

Adım 4 / 7

DEĞİŞKENLER

AL = 47h

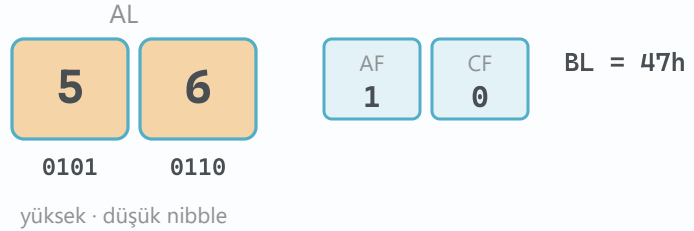
AF = 1

CF = 0

BL = 47h

Kod İzleme: AF ile BCD Düzeltmesi

```
1 org 100h
2 mov al, 19h      ; BCD 19
3 add al, 28h      ; ikili toplama
4 daa              ; BCD düzeltmesi
5 mov bl, al       ; 47 sakla
▶ mov al, 56h      ; BCD 56
7 add al, 37h
8 daa
9 ret
```



ondalık: 56

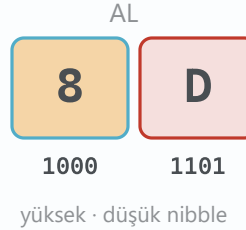
İkinci örnek: AL = **56h**.

Adım 5 / 7

DEĞİŞKENLERAL = **56h**AF = **1**CF = **0**BL = **47h**

Kod İzleme: AF ile BCD Düzeltmesi

```
1 org 100h
2 mov al, 19h      ; BCD 19
3 add al, 28h      ; ikili toplama
4 daa              ; BCD düzeltmesi
5 mov bl, al        ; 47 sakla
6 mov al, 56h      ; BCD 56
7 ► add al, 37h
8 daa
9 ret
```



BL = 47h

ikili sonuç, henüz BCD değil

ondalık: —

56h + 37h = **8Dh**. Bu kez elde yok (AF = 0) ama düşük nibble D > 9: geçerli bir BCD basamağı değil.

Adım 6 / 7

DEĞİŞKENLER

AL = 8Dh

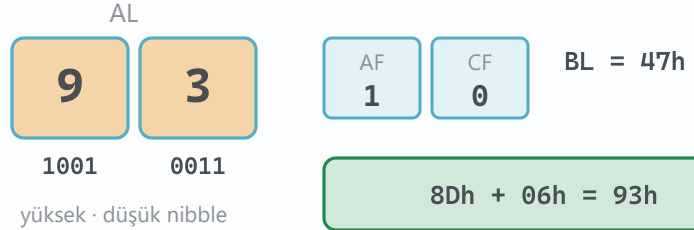
AF = 0

CF = 0

BL = 47h

Kod İzleme: AF ile BCD Düzeltmesi

```
1 org 100h
2 mov al, 19h      ; BCD 19
3 add al, 28h      ; ikili toplama
4 daa              ; BCD düzeltmesi
5 mov bl, al        ; 47 sakla
6 mov al, 56h      ; BCD 56
7 add al, 37h
▶ daa
9 ret
```



ondalık: 93

daa : düşük nibble > 9 olduğu için +6 → **93h** = 56 + 37 ✓.
Yüksek nibble ≤ 9 ve CF = 0 → üst basamak düzeltilmez.

Adım 7 / 7

DEĞİŞKENLER

AL = 93h

AF = 1

CF = 0

BL = 47h

CF — Elde Bayrağı

- Carry Flag: n bitlik işlemde sonuç **n bite sığmazsa** (MSB'den elde çıkarsa) **CF = 1**; aksi hâlde 0.
- Çıkarmada **ödünç** (küçükten büyük çıkarma) olursa da CF = 1 olur.
- **İşaretsiz** sayılar için taşıma göstergesidir.
- Kaydırma/döndürme komutlarında dışarı atılan bit CF'ye gider.
- Programcı da değiştirebilir: **STC** (1), **CLC** (0), **CMC** (tersle).
- Çok baytlı toplamada **ADC** bu eldeyi bir sonraki bayta taşır.

İşlem (8 bit)	Sonuç	CF
FFh + 01h	00h	1
80h + 80h	00h	1
70h + 10h	80h	0
05h - 07h	FEh	1 (ödünç)
07h - 05h	02h	0

OF — Taşma Bayrağı

- Overflow Flag: sonuç, **işaretli** olarak yorumlandığında kullanılan bit sayısına **sığmıyorsa** $OF = 1$; aksi hâlde 0.
- 8 bitte işaretli aralık $-128 \dots +127$, 16 bitte $-32768 \dots +32767$ 'dir.
- Kural (toplama): iki **pozitif** toplanıp negatif, ya da iki **negatif** toplanıp pozitif sonuç çıkarsa $OF = 1$. Farklı işaretli sayıların toplamı asla taşmaz.
- Donanımca: MSB'ye **giren** elde ile MSB'den **çıkan** elde farklıysa $OF = 1$.

İşlem (8 bit)	İşaretsiz yorum	İşaretli yorum	Sonuç	CF	OF
05h + 03h	$5 + 3 = 8$	$5 + 3 = 8$	08h	0	0
7Fh + 01h	$127 + 1 = 128$	$127 + 1 \neq -128$	80h	0	1
FFh + 01h	$255 + 1 \neq 0$	$-1 + 1 = 0$	00h	1	0
80h + 80h	$128 + 128 \neq 0$	$-128 + (-128) \neq 0$	00h	1	1

CF işaretsiz taşmayı, OF işaretli taşmayı gösterir; ikisi birbirinden bağımsızdır.

Bayrakları Elle Hesaplamak: 6Ah + 5Ch

	bit 7 ... 0							
elde								
6Ah	0	1	1	0	1	0	1	0
+5Ch	0	1	0	1	1	1	0	0

`MOV AL, 6Ah` / `ADD AL, 5Ch` : iki sayı ikili yazılır ve sağdan sola toplanır.

Adım 1 / 4

İşaretsiz yorum: $106 + 92 = 198$ doğru (CF = 0). İşaretli yorum: $+106 + (+92) = -58$ görünür → yanlış, çünkü OF = 1.

Bayrakları Elle Hesaplamak: 6Ah + 5Ch

	bit 7 ... 0							
elde					0	0	0	AF = 1 bit 3 → 4 elde var
6Ah	0	1	1	0	1	0	1	0
+5Ch	0	1	0	1	1	1	0	0
					0	1	1	0

Düşük nibble: $1010 + 1100 = 1\ 0110 \rightarrow$ bit 3'ten bit 4'e **elde** geçer $\rightarrow$ **AF = 1**.

Adım 2 / 4

İşaretsiz yorum: $106 + 92 = 198$ doğru (CF = 0). İşaretli yorum: $+106 + (+92) = -58$ görünür $\rightarrow$ yanlış, çünkü OF = 1.

Bayrakları Elle Hesaplamak: 6Ah + 5Ch

	bit 7 ... 0							
elde	1	1	1	1	0	0	0	
6Ah	0	1	1	0	1	0	1	0
+5Ch	0	1	0	1	1	1	0	0
C6h	1	1	0	0	0	1	1	0

AF = 1 bit 3 → 4 elde var
CF = 0 bit 7'den elde çıkmadı
OF = 1 bit 7'ye giren 1 ≠ çıkan 0

Yüksek nibble eldeyle toplanır: sonuç **C6h**. Bit 7'den elde çıkmadı → **CF = 0**; bit 7'ye 1 girdi ama 0 çıktı → **OF = 1** ($106 + 92 = 198 > 127$).

Adım 3 / 4

İşaretsiz yorum: $106 + 92 = 198$ doğru (**CF = 0**). İşaretili yorum: $+106 + (+92) = -58$ görünür → yanlış, çünkü **OF = 1**.

Bayrakları Elle Hesaplamak: 6Ah + 5Ch

	bit 7 ... 0							
elde	1	1	1	1	0	0	0	
6Ah	0	1	1	0	1	0	1	
+5Ch	0	1	0	1	1	1	0	
C6h	1	1	0	0	0	1	1	

AF = 1 bit 3 → 4 elde var
CF = 0 bit 7'den elde çıkmadı
OF = 1 bit 7'ye giren 1 ≠ çıkan 0
SF = 1 MSB = 1
ZF = 0 sonuç ≠ 0
PF = 1 1100 0110: dört 1

Sonuçtan: MSB = 1 → SF = 1; sonuç sıfır değil → ZF = 0; 1'lerin sayısı 4 (çift) → PF = 1.

Adım 4 / 4

İşaretsiz yorum: $106 + 92 = 198$ doğru (CF = 0). İşaretili yorum: $+106 + (+92) = -58$ görünür → yanlış, çünkü OF = 1.

Bayrak Hesaplayıcı: ADD / SUB / AND / INC

A = B = genişlik

ADD A, B

SUB A, B

AND A, B

INC A

0110 1010 6Ah
+ 0101 1100 5Ch
= 1100 0110 C6h

CF=0
elde

ZF=0
sıfır

SF=1
işaret

OF=1
taşma

PF=1
çiftlik

AF=1
yrd. elde

ADD · İşaretsiz: $106 + 92 = 198$ ✓ · İşaretli: $106 + 92 \rightarrow -58$ **yanlış** (OF = 1)

Onaltılık değer gir (ör. 7F ve 01, FF ve 01, 80 ve 80). INC A, CF bayrağını değiştirmez; AND ise CF = OF = 0 yapar (AF tanımsızdır).

Bayrak İzleme

► **MOV** AL, 7Fh

2 **ADD** AL, 1 ; 7Fh+1
3 **ADD** AL, 80h ; 80h+80h
4 **SUB** AL, 1 ; 00h-1
5 **INC** AL ; FFh+1
6 **CMP** AL, 0 ; çıkar, sakla

AL = 7Fh (+127). MOV bayrakları **etkilemez**.

Adım 1 / 6

DEĞİŞKENLER

AL = **7Fh**

CF = **0**

ZF = **0**

SF = **0**

OF = **0**

PF = **0**

AF = **0**

Bayrak İzleme

```
1 MOV AL, 7Fh
▶ ADD AL, 1 ; 7Fh+1
3 ADD AL, 80h ; 80h+80h
4 SUB AL, 1 ; 00h-1
5 INC AL ; FFh+1
6 CMP AL, 0 ; çıkar, sakla
```

$7Fh + 1 = 80h$: SF = 1, OF = 1 (+127 + 1 işaretli aralığı aştı),
AF = 1 (F + 1), CF = 0.

Adım 2 / 6

DEĞİŞKENLER

AL = 80h

CF = 0

ZF = 0

SF = 1

OF = 1

PF = 0

AF = 1

Bayrak İzleme

```
1 MOV AL, 7Fh
2 ADD AL, 1      ; 7Fh+1
3 ► ADD AL, 80h  ; 80h+80h
4 SUB AL, 1      ; 00h-1
5 INC AL         ; FFh+1
6 CMP AL, 0      ; çıkar, sakla
```

$80h + 80h = 100h \rightarrow AL = \mathbf{00h}$: CF = 1, ZF = 1, OF = 1 (iki negatifin toplamı pozitif çıktı).

Adım 3 / 6

DEĞİŞKENLER

AL = **00h**

CF = **1**

ZF = **1**

SF = **0**

OF = **1**

PF = **1**

AF = **0**

Bayrak İzleme

```
1 MOV AL, 7Fh
2 ADD AL, 1      ; 7Fh+1
3 ADD AL, 80h    ; 80h+80h
4 ► SUB AL, 1     ; 00h-1
5 INC AL         ; FFh+1
6 CMP AL, 0      ; çıkar, sakla
```

00h - 1 = **FFh**: ödünç → CF = 1, SF = 1, AF = 1; işaretli 0 - 1 = -1 doğru → OF = 0.

Adım 4 / 6

DEĞİŞKENLER

AL = **FFh**

CF = **1**

ZF = **0**

SF = **1**

OF = **0**

PF = **1**

AF = **1**

Bayrak İzleme

```
1 MOV AL, 7Fh
2 ADD AL, 1      ; 7Fh+1
3 ADD AL, 80h    ; 80h+80h
4 SUB AL, 1      ; 00h-1
5 ► INC AL       ; FFh+1
6 CMP AL, 0      ; çıkar, sakla
```

$FFh + 1 = 00h$: ZF = 1. Dikkat: **INC/DEC CF'yi değiştirmez**, CF önceki komuttan 1 kaldı.

Adım 5 / 6

DEĞİŞKENLER

AL = 00h

CF = 1

ZF = 1

SF = 0

OF = 0

PF = 1

AF = 1

Bayrak İzleme

```
1 MOV AL, 7Fh
2 ADD AL, 1      ; 7Fh+1
3 ADD AL, 80h    ; 80h+80h
4 SUB AL, 1      ; 00h-1
5 INC AL         ; FFh+1
▶ CMP AL, 0      ; çıkar, sakla
```

CMP, AL – 0 hesaplar ama sonucu yazmaz: ZF = 1 (eşit), CF = 0.

Adım 6 / 6

DEĞİŞKENLER

AL = 00h

CF = 0

ZF = 1

SF = 0

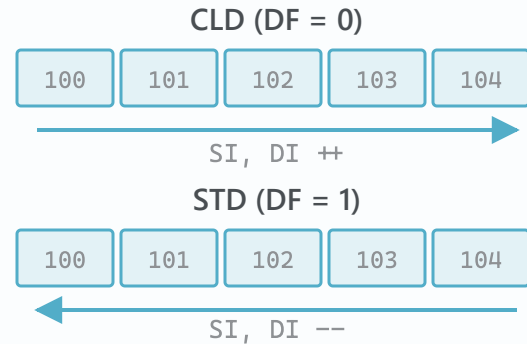
OF = 0

PF = 1

AF = 0

DF — Yön Bayrağı

- Direction Flag, **dize (string)** komutlarında veri erişim yönünü belirler.
- **DF = 0**: düşük adresten yükseğe; her adımda SI/DI **artar**.
- **DF = 1**: yüksek adresten düşüğe; SI/DI **azalır**.
- Adım büyüklüğü bayt komutlarında 1, sözcük komutlarında 2'dir.
- **CLD** → DF = 0, **STD** → DF = 1.
- Üst üste binen alanları kopyalarken yön önemlidir: hedef kaynağın ilerisindeyse **sondan başa** (STD) kopyalanır.



Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
```

```
▶ mov si, offset dizi+3 ; kaynak
```

```
3 mov di, offset dizi+4 ; hedef
```

```
4 mov cx, 4
```

```
5 std ; DF = 1
```

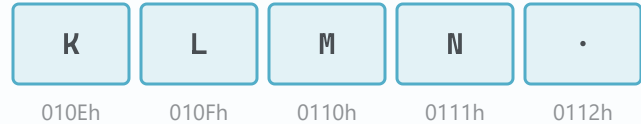
```
6 rep movsb
```

```
7 cld
```

```
8 ret
```

```
9 dizi db 'KLMN', 0
```

dizi (010Eh–0112h)



DF = 0 · CX = 0

“KLMN” bir sağa kaydırılacak; kaynak ve hedef **örtüşüyor**. SI = 0111h: kaynağın **son** baytı.

Adım 1 / 9

DEĞİŞKENLER

SI = 0111h

DI = 0000h

CX = 0

DF = 0

Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
2 mov si, offset dizi+3 ; kaynak
3 ► mov di, offset dizi+4 ; hedef
4 mov cx, 4
5 std ; DF = 1
6 rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh-0112h)



DF = 0 · CX = 0

DI = 0112h: hedefin son baytı (bir sağda).

Adım 2 / 9

DEĞİŞKENLER

SI = 0111h

DI = 0112h

CX = 0

DF = 0

Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
2 mov si, offset dizi+3 ; kaynak
3 mov di, offset dizi+4 ; hedef
▶ mov cx, 4
5 std ; DF = 1
6 rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh-0112h)



DF = 0 · CX = 4

CX = 4 bayt kopyalanacak.

Adım 3 / 9

DEĞİŞKENLER

SI = 0111h

DI = 0112h

CX = 4

DF = 0

Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
2 mov si, offset dizi+3 ; kaynak
3 mov di, offset dizi+4 ; hedef
4 mov cx, 4
5 ► std ; DF = 1
6 rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh-0112h)



$$DF = 1 \cdot CX = 4$$

std : **DF = 1**; dizgi komutları SI ve DI'yi her adımda azaltacak.

Adım 4 / 9

DEĞİŞKENLER

SI = 0111h

DI = 0112h

CX = 4

DF = 1

Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
2 mov si, offset dizi+3 ; kaynak
3 mov di, offset dizi+4 ; hedef
4 mov cx, 4
5 std ; DF = 1
6 rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh-0112h)



$DF = 1 \cdot CX = 3$

MOVSB 1: [0111h] → [0112h] ('N'); SI, DI 1 azaldı, CX = 3.

Adım 5 / 9

DEĞİŞKENLER

SI = 0110h

DI = 0111h

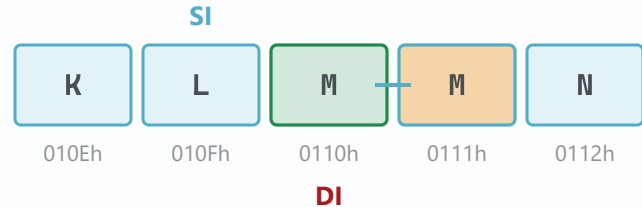
CX = 3

DF = 1

Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
2 mov si, offset dizi+3 ; kaynak
3 mov di, offset dizi+4 ; hedef
4 mov cx, 4
5 std                      ; DF = 1
6 ► rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh-0112h)



$$DF = 1 \cdot CX = 2$$

MOVSB 2: [0110h] → [0111h] ('M'); SI, DI 1 azaldı, CX = 2.

Adım 6 / 9

DEĞİŞKENLER

SI = 010Fh

DI = 0110h

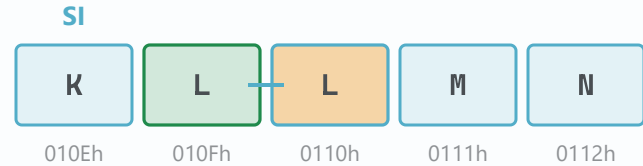
CX = 2

DF = 1

Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
2 mov si, offset dizi+3 ; kaynak
3 mov di, offset dizi+4 ; hedef
4 mov cx, 4
5 std                      ; DF = 1
6 rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh-0112h)



$DF = 1 \cdot CX = 1$

MOVSB 3: [010Fh] → [0110h] ('L'); SI, DI 1 azaldı, CX = 1.

Adım 7 / 9

DEĞİŞKENLER

SI = 010Eh

DI = 010Fh

CX = 1

DF = 1

Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
2 mov si, offset dizi+3 ; kaynak
3 mov di, offset dizi+4 ; hedef
4 mov cx, 4
5 std                      ; DF = 1
6 rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh-0112h)

SI



DI

DF = 1 · CX = 0

MOVSB 4: [010Eh] → [010Fh] ('K'); SI, DI 1 azaldı, CX = 0.

Sonuç **KKLMN**: dizi doğru kaydı.

Adım 8 / 9

DEĞİŞKENLER

SI = 010Dh

DI = 010Eh

CX = 0

DF = 1

Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
2 mov si, offset dizi+3 ; kaynak
3 mov di, offset dizi+4 ; hedef
4 mov cx, 4
5 std                      ; DF = 1
6 rep movsb
7 ► cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh-0112h)

SI



DI

DF = 0 · CX = 0

`cld` ile DF = 0'a dönlür. CLD ile baştan kopyalansaydı 'K' her bayta taşınır, sonuç **KKKKK** olurdu: örtüşen alanda hedef kaynağın ilerisindeyse geriye kopyalanır.

Adım 9 / 9

DEĞİŞKENLER

SI = 010Dh

DI = 010Eh

CX = 0

DF = 0

IF ve TF

IF — Kesme Bayrağı (Interrupt)

- Dış donanım **kesmeleri** (INTR girişi) ile ilişkilidir.
- **IF = 1**: çevre birimlerinden gelen maskelenebilir kesme talepleri ele alınır.
- **IF = 0**: bu talepler göz ardı edilir (maskelenir).
- `STI` → IF = 1, `CLI` → IF = 0.
- NMI (maskelenemeyen kesme) ve yazılım kesmeleri (`INT n`) IF'den **etkilenmez**.

TF — Tuzak Bayrağı (Trap)

- Yonga içi **hata ayıklama** için kullanılır.
- **TF = 1**: tek adım (single step) modu; her komuttan sonra dahili **INT 1** kesmesi üretilir ve tek adım ISR'si (kesme hizmet yordamı) çalışır.
- Böylece program komut komut incelenebilir (DEBUG, emu8086 "single step").
- **TF = 0**: normal çalışma.
- Doğrudan komutu yoktur; `PUSHF` → bit 8'i değiştirir → `POPF` .



Bayrakları Değiştiren Komutlar

Komut	Etki	Komut	Etki
CLC	CF = 0	CLD	DF = 0
STC	CF = 1	STD	DF = 1
CMC	CF = $\neg$ CF	CLI / STI	IF = 0 / 1
LAHF	AH $\leftarrow$ SF ZF – AF – PF – CF	SAHF	AH $\rightarrow$ düşük bayrak baytı
PUSHF	FLAGS'i yığına koy	POPF	yığından FLAGS'e yükle

Komut grubu	Bayraklara etkisi
ADD, ADC, SUB, SBB, CMP, NEG	CF, PF, AF, ZF, SF, OF hepsi güncellenir
INC, DEC	CF hariç hepsi güncellenir
AND, OR, XOR, TEST	CF = OF = 0; PF, ZF, SF güncellenir; AF tanımsız
MOV, PUSH, POP, XCHG, LEA, LOOP, JMP	bayrakları etkilemez

Kod İzleme: Bayrakları Okumak ve Yazmak

```
1 org 100h
```

```
► mov al, 0FFh
```

```
3 add al, 1 ; CF ZF AF PF = 1
```

```
4 lahf ; AH ← bayraklar
```

```
5 cmc ; CF tersle
```

```
6 clc ; CF = 0
```

```
7 stc ; CF = 1
```

```
8 mov ah, 0C1h ; 1100 0001b
```

```
9 sahf ; bayraklar ← AH
```

```
10 ret
```

bayraklar (düşük bayt)

SF	ZF	–	AF	–	PF	–	CF
0	0	0	0	0	0	1	0

AH

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

AH = 00h

AL = FFh. Bayrakların düşük baytı: SF ZF – AF – PF – CF.

Adım 1 / 8

DEĞİŞKENLER

AH = 00h

SF = 0

ZF = 0

AF = 0

PF = 0

CF = 0

Kod İzleme: Bayrakları Okumak ve Yazmak

```
1 org 100h
2 mov al, 0FFh
3 ► add al, 1 ; CF ZF AF PF = 1
4 lahf ; AH ← bayraklar
5 cmc ; CF tersle
6 cll ; CF = 0
7 stc ; CF = 1
8 mov ah, 0C1h ; 1100 0001b
9 sahf ; bayraklar ← AH
10 ret
```

bayraklar (düşük bayt)

SF	ZF	–	AF	–	PF	–	CF
0	1	0	1	0	1	1	1

AH

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

AH = 00h

FFh + 1 = 00h: **CF = 1**, **ZF = 1**, AF = 1 (F + 1), PF = 1 (00h'de sıfır tane 1, çift).

Adım 2 / 8

DEĞİŞKENLER

AH = 00h

SF = 0

ZF = 1

AF = 1

PF = 1

CF = 1

Kod İzleme: Bayrakları Okumak ve Yazmak

```
1 org 100h
2 mov al, 0FFh
3 add al, 1      ; CF ZF AF PF = 1
4 ► lahf         ; AH ← bayraklar
5 cmc           ; CF tersle
6 clc           ; CF = 0
7 stc           ; CF = 1
8 mov ah, 0C1h  ; 1100 0001b
9 sahf          ; bayraklar ← AH
10 ret
```

bayraklar (düşük bayt)

SF	ZF	–	AF	–	PF	–	CF
0	1	0	1	0	1	1	1

AH

0	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---

AH = 57h

lahf : bayrakların düşük baytı AH'ye kopyalanır → AH = **57h** = 0101 0111b (bit 1 her zaman 1).

Adım 3 / 8

DEĞİŞKENLER

AH = 57h

SF = 0

ZF = 1

AF = 1

PF = 1

CF = 1

Kod İzleme: Bayrakları Okumak ve Yazmak

```
1 org 100h
2 mov al, 0FFh
3 add al, 1      ; CF ZF AF PF = 1
4 lahf          ; AH ← bayraklar
5 ▶ cmc         ; CF tersle
6 clc          ; CF = 0
7 stc          ; CF = 1
8 mov ah, 0C1h  ; 1100 0001b
9 sahf         ; bayraklar ← AH
10 ret
```

bayraklar (düşük bayt)

SF	ZF	–	AF	–	PF	–	CF
0	1	0	1	0	1	1	0

AH

0	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---

AH = 57h

cmc : CF tersine döner, 1 → 0. Diğer bayraklar aynı.

Adım 4 / 8

DEĞİŞKENLER

AH = 57h

SF = 0

ZF = 1

AF = 1

PF = 1

CF = 0

Kod İzleme: Bayrakları Okumak ve Yazmak

```
1 org 100h
2 mov al, 0FFh
3 add al, 1      ; CF ZF AF PF = 1
4 lahf          ; AH ← bayraklar
5 cmc           ; CF tersle
6 ► clc         ; CF = 0
7 stc           ; CF = 1
8 mov ah, 0C1h  ; 1100 0001b
9 sahf          ; bayraklar ← AH
10 ret
```

bayraklar (düşük bayt)

SF	ZF	–	AF	–	PF	–	CF
0	1	0	1	0	1	1	0

AH

0	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---

AH = 57h

clc : CF = 0 (zaten 0'dı).

Adım 5 / 8

DEĞİŞKENLER

AH = 57h

SF = 0

ZF = 1

AF = 1

PF = 1

CF = 0

Kod İzleme: Bayrakları Okumak ve Yazmak

```
1 org 100h
2 mov al, 0FFh
3 add al, 1      ; CF ZF AF PF = 1
4 lahf          ; AH ← bayraklar
5 cmc          ; CF tersle
6 clc          ; CF = 0
7 ► stc        ; CF = 1
8 mov ah, 0C1h  ; 1100 0001b
9 sahf         ; bayraklar ← AH
10 ret
```

bayraklar (düşük bayt)

SF	ZF	–	AF	–	PF	–	CF
0	1	0	1	0	1	1	1

AH

0	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---

AH = 57h

stc : **CF = 1**. Çok baytlı ADC/SBB öncesi eldeyi hazırlamak için kullanılır.

Adım 6 / 8

DEĞİŞKENLER

AH = 57h

SF = 0

ZF = 1

AF = 1

PF = 1

CF = 1

Kod İzleme: Bayrakları Okumak ve Yazmak

```
1 org 100h
2 mov al, 0FFh
3 add al, 1      ; CF ZF AF PF = 1
4 lahf          ; AH ← bayraklar
5 cmc          ; CF tersle
6 clc          ; CF = 0
7 stc          ; CF = 1
▶ 8 mov ah, 0C1h ; 1100 0001b
9 sahf         ; bayraklar ← AH
10 ret
```

bayraklar (düşük bayt)

SF	ZF	–	AF	–	PF	–	CF
0	1	0	1	0	1	1	1

AH

1	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

AH = C1h

AH = C1h = 1100 0001b: bit 7 (SF), bit 6 (ZF), bit 0 (CF) = 1.

Adım 7 / 8

DEĞİŞKENLER

AH = C1h

SF = 0

ZF = 1

AF = 1

PF = 1

CF = 1

Kod İzleme: Bayrakları Okumak ve Yazmak

```
1 org 100h
2 mov al, 0FFh
3 add al, 1      ; CF ZF AF PF = 1
4 lahf          ; AH ← bayraklar
5 cmc           ; CF tersle
6 clc           ; CF = 0
7 stc           ; CF = 1
8 mov ah, 0C1h  ; 1100 0001b
9 ▶ sahf        ; bayraklar ← AH
10 ret
```

bayraklar (düşük bayt)



AH



AH = C1h

sahf : AH bayrakların düşük baytına yazılır → **SF = 1, ZF = 1, CF = 1**, AF = PF = 0. OF, DF, IF, TF etkilenmez.

Adım 8 / 8

DEĞİŞKENLER

AH = C1h

SF = 1

ZF = 1

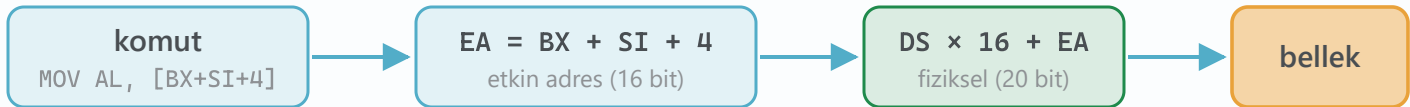
AF = 0

PF = 0

CF = 1

Adresleme Modları

- İşlenen verinin (operand) **belirtilme yöntemine** adresleme modu denir.
- İşlenenin bir **veri** (immediate data) mi, bir **yazmaç** mı, yoksa bir **bellek adresi** mi olduğunu belirtir.
- Bellek işlenenlerinde işlemci önce **etkin adresi** (EA, effective address = bağlı konum) hesaplar, sonra kesim yazmacıyla fiziksel adresi bulur.



ADRESLEME MODLARI

Modlara Genel Bakış

Mod	Örnek	İşlenen / etkin adres
Yazmaç	<code>MOV AX, BX</code>	yazmacın kendisi
Anlık (immediate)	<code>MOV CL, 0Ah</code>	komutun içindeki sabit
Doğrudan (direct)	<code>MOV AX, [0500h]</code>	EA = sabit adres
Yazmaç dolaylı	<code>MOV AX, [SI]</code>	EA = BX, SI ya da DI
Taban	<code>MOV AL, [BP+0100h]</code>	EA = BX/BP + yer değiştirme
İndis	<code>MOV AX, [SI+2000h]</code>	EA = SI/DI + yer değiştirme
Taban indisli	<code>MOV AX, [BX+DI]</code>	EA = BX/BP + SI/DI
Taban indisli yer değiştirme	<code>MOV AL, [BP+SI+2000h]</code>	EA = taban + indis + yer değiştirme
Dize	<code>MOVSB</code>	DS:SI → ES:DI, otomatik ±
G/Ç	<code>IN AL, 45h</code>	port numarası
Göreceli	<code>JNZ etiket</code>	IP = IP + işaretli uzaklık

Yazmaç ve Anlık Mod

Yazmaç Modu (Register)

- Her iki işlenen de yazmaçtır; bellek erişimi yoktur → **en hızlı** mod.
- `MOV AX, BX`
- `XOR AX, DX`
- `ADD AL, BL`
- İki işlenen **aynı boyda** olmalı: `MOV AL, BX` hatalıdır.

Anlık Mod (Immediate)

- Kaynak işlenen komutun içinde verilen 8 ya da 16 bitlik sabittir.
- `MOV AX, 2000h`
- `MOV CL, 0Ah`
- `ADD AL, 45h`
- `AND AX, 0000h`
- Hedef işlenen **asla** anlık olamaz: `MOV 5, AL` anlamsızdır.

`MOV CS, AX` geçerli bir komut değildir: CS, MOV ile değiştirilemez (yalnızca uzak JMP/CALL/RET/IRET ve kesmeler değiştirir). Kesim yazmacına sabit yüklemek için bir yazmaç gerekir: `MOV AX, 2000h + MOV DS, AX`.

Geçersiz İşlenen Bileşimleri

Hatalı komut	Neden?	Doğrusu
<code>MOV [SI], [DI]</code>	bellekten belleğe aktarım yok	<code>MOV AL, [DI]</code> + <code>MOV [SI], AL</code> (ya da <code>MOVSB</code>)
<code>MOV DS, 2000h</code>	kesim yazmacına sabit yüklenemez	<code>MOV AX, 2000h</code> + <code>MOV DS, AX</code>
<code>MOV ES, DS</code>	kesimden kesime aktarım yok	<code>PUSH DS</code> + <code>POP ES</code>
<code>MOV AL, BX</code>	boyutlar uyuşmuyor	<code>MOV AL, BL</code>
<code>MOV [BX], 5</code>	boyut belirsiz	<code>MOV BYTE PTR [BX], 5</code>
<code>MOV AX, [CX]</code>	CX adresleme yazmacı değil	<code>MOV BX, CX</code> + <code>MOV AX, [BX]</code>
<code>MOV AX, [SI+DI]</code>	iki indis birlikte kullanılamaz	<code>MOV BX, SI</code> + <code>MOV AX, [BX+DI]</code>
<code>MOV IP, AX</code>	IP doğrudan yazılamaz	<code>JMP AX</code>

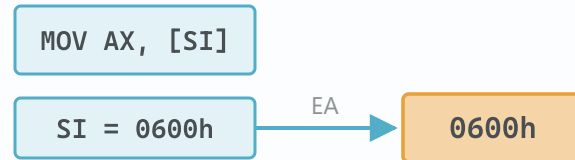
Doğrudan ve Yazmaç Dolaylı Mod

Doğrudan Mod (Displacement / Direct)

- Etkin adres komut içinde **doğrudan** verilir.
- `MOV AX, [0500h]`
- `MOV AX, [DISP]` — DISP bir değişken adı ya da sabit adres.
- Değişken adları da bu moddadır: `MOV AL, sayi`.

Yazmaç Dolaylı Mod (Register Indirect)

- Etkin adres **BX, SI ya da DI** yazmacındadır.
- `MOV AX, [DI]`
- `ADD AL, [BX]`
- `MOV AX, [SI]`
- Fiziksel adres = kesim × 16 + etkin adres.



Taban ve İndis Modları

Taban Modu (Based)

- EA = **taban yazmacı** (BX ya da BP) + yer değiştirme.
- `MOV AL, [BP+0100h]`
- `MOV AX, [BX+4]` — bir kaydın 4. baytındaki alan.
- Kayıt/yapı alanlarına erişim için uygundur.

İndis Modu (Indexed)

- EA = **indis yazmacı** (SI ya da DI) + yer değiştirme.
- `MOV AX, [SI+2000h]`
- `MOV AL, [DI+3000h]`
- Yer değiştirme dizinin başlangıç adresi, indis eleman konumu olarak düşünülebilir:
`MOV AL, dizi[SI]`

Taban İndisli Mod (Based Indexed)

- EA = taban (BX/BP) + indis (SI/DI).
- `MOV AL, [BP+SI]`, `MOV AX, [BX+DI]`
- İki boyutlu dizi: BX satır başı, SI sütun.

Taban İndisli Yer değiştirme Modu

- EA = taban + indis + yer değiştirme.
- `MOV AL, [BP+SI+2000h]`
- Yığındaki yerel bir dizinin elemanı gibi karmaşık yapılara erişim.

Kod İzleme: Matrisin Bir Satırını Toplamak

```
1 org 100h
```

```
▶ mov bx, 4           ; satır 1  
▶ mov si, 0           ; sütun  
▶ mov al, 0           ; toplam  
▶ mov cx, 4
```

```
6 topla: add al, mat[bx+si]
```

```
7 inc si
```

```
8 loop topla
```

```
9 ret
```

```
10 mat db 1, 2, 3, 4
```

```
11      db 5, 6, 7, 8
```

```
12      db 9, 10, 11, 12
```

satır 0

1

2

3

4

satır 1

5

6

7

8

satır 2

9

10

11

12

SI=0

SI=1

SI=2

SI=3

AL (toplam)

0

CX = 4

3 × 4'lük matris satır satır saklanır. **BX** = 4 satır 1'in başı (satır × 4 sütun), **SI** sütun, toplam AL = 0.

Adım 1 / 6

DEĞİŞKENLER

BX = 4

SI = 0

CX = 4

AL = 0

Kod İzleme: Matrisin Bir Satırını Toplamak

```
1 org 100h
2 mov bx, 4          ; satır 1
3 mov si, 0          ; sütun
4 mov al, 0          ; toplam
5 mov cx, 4
```

► **topla:** add al, mat[bx+si]

► **inc si**

► **loop topla**

```
9 ret
```

```
10 mat db 1, 2, 3, 4
```

```
11      db 5, 6, 7, 8
```

```
12      db 9, 10, 11, 12
```

satır 0	1	2	3	4	EA = mat + BX + SI = 0113h + 4 + 0 = 0117h AL (toplam) 5 CX = 3
satır 1	5	6	7	8	
satır 2	9	10	11	12	
	SI=0	SI=1	SI=2	SI=3	

Taban indisli mod: EA = mat + BX + SI = 0117h → mat[1][0] = 5. AL = **5**; SI → 1, CX = 3.

Adım 2 / 6

DEĞİŞKENLER

BX = 4

SI = 1

CX = 3

AL = 5

Kod İzleme: Matrisin Bir Satırını Toplamak

```
1 org 100h
2 mov bx, 4           ; satır 1
3 mov si, 0           ; sütun
4 mov al, 0           ; toplam
5 mov cx, 4
```

```
▶ topla: add al, mat[bx+si]
```

```
▶ inc si
```

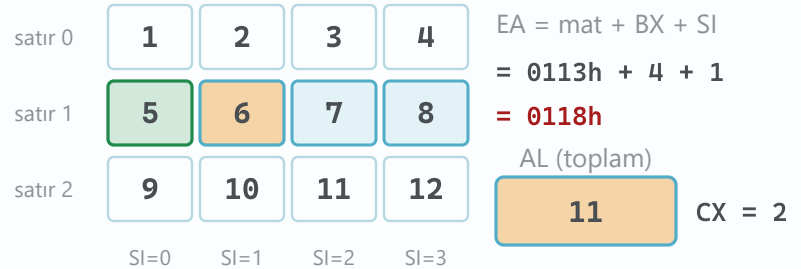
```
▶ loop topla
```

```
9 ret
```

```
10 mat db 1, 2, 3, 4
```

```
11     db 5, 6, 7, 8
```

```
12     db 9, 10, 11, 12
```



Taban indisli mod: EA = mat + BX + SI = 0118h → mat[1][1] = 6. AL = 11; SI → 2, CX = 2.

Adım 3 / 6

DEĞİŞKENLER

BX = 4

SI = 2

CX = 2

AL = 11

Kod İzleme: Matrisin Bir Satırını Toplamak

```
1 org 100h
2 mov bx, 4          ; satır 1
3 mov si, 0          ; sütun
4 mov al, 0          ; toplam
5 mov cx, 4
```

```
▶ topla: add al, mat[bx+si]
```

```
▶ inc si
```

```
▶ loop topla
```

```
9 ret
```

```
10 mat db 1, 2, 3, 4
```

```
11     db 5, 6, 7, 8
```

```
12     db 9, 10, 11, 12
```

satır 0	1	2	3	4	EA = mat + BX + SI = 0113h + 4 + 2 = 0119h AL (toplam) 18 CX = 1
satır 1	5	6	7	8	
satır 2	9	10	11	12	
	SI=0	SI=1	SI=2	SI=3	

Taban indisli mod: EA = mat + BX + SI = 0119h → mat[1][2] = 7. AL = **18**; SI → 3, CX = 1.

Adım 4 / 6

DEĞİŞKENLER

BX = 4

SI = 3

CX = 1

AL = 18

Kod İzleme: Matrisin Bir Satırını Toplamak

```
1 org 100h
2 mov bx, 4           ; satır 1
3 mov si, 0           ; sütun
4 mov al, 0           ; toplam
5 mov cx, 4
```

► **topla:** add al, mat[bx+si]

► **inc si**

► **loop topla**

```
9 ret
```

```
10 mat db 1, 2, 3, 4
```

```
11     db 5, 6, 7, 8
```

```
12     db 9, 10, 11, 12
```

satır 0	1	2	3	4	EA = mat + BX + SI = 0113h + 4 + 3 = 011Ah AL (toplam) 26 CX = 0
satır 1	5	6	7	8	
satır 2	9	10	11	12	
	SI=0	SI=1	SI=2	SI=3	

Taban indisli mod: EA = mat + BX + SI = 011Ah → mat[1][3]
= 8. AL = **26**; SI → 4, CX = 0. Satır toplamı 5 + 6 + 7 + 8 = **26**.

Adım 5 / 6

DEĞİŞKENLER

BX = 4

SI = 4

CX = 0

AL = 26

Kod İzleme: Matrisin Bir Satırını Toplamak

```
1 org 100h
2 mov bx, 4          ; satır 1
3 mov si, 0          ; sütun
4 mov al, 0          ; toplam
5 mov cx, 4
6 topla: add al, mat[bx+si]
7 inc si
8 loop topla
9 ret
10 mat db 1, 2, 3, 4
11      db 5, 6, 7, 8
12      db 9, 10, 11, 12
```

satır 0	1	2	3	4
satır 1	5	6	7	8
satır 2	9	10	11	12
	SI=0	SI=1	SI=2	SI=3

AL (toplam) **26** CX = 0

Satır 1 toplandı: AL = **26**. Başka bir satır için yalnız BX değişir (satır 2 → BX = 8); erişim kodu aynı kalır.

Adım 6 / 6

DEĞİŞKENLER

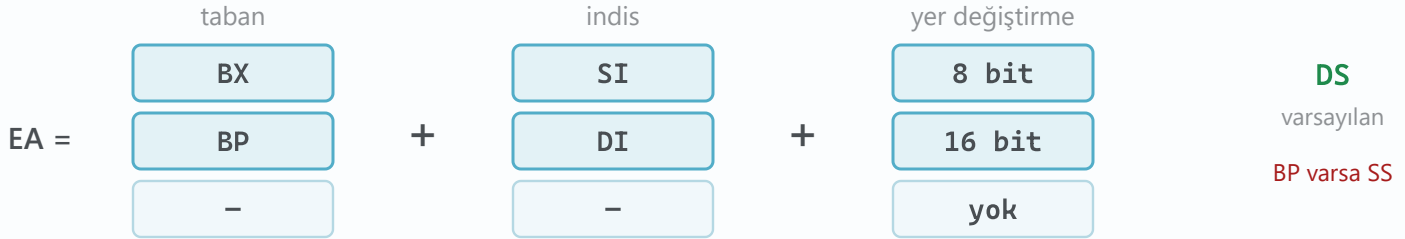
BX = 4

SI = 4

CX = 0

AL = 26

Etkin Adres Bileşimleri



- Her sütundan **en fazla bir** öge seçilir; en az biri bulunmalıdır. Toplam 24 geçerli bileşim vardır (yalnız yer değiştirme = doğrudan mod).
- BP** içeren adresler varsayılan olarak **SS**, diğerleri **DS** kesimini kullanır.
- Varsayılan kesim **kesim önekiyle** değiştirilebilir: `MOV AL, ES:[BX]`, `MOV AX, DS:[BP+2]`.
- Yer değiştirme işaretli 8 bit (-128...+127) ya da 16 bit olabilir; toplam 16 bitte sarar (FFFFh + 1 = 0000h).

Etkin ve Fiziksel Adres: **[BX+SI+10h]**

```
MOV AL, [BX+SI+10h]
```

```
DS = 1000h BX = 0200h SI = 0030h
```

Komut ve yazmaç değerleri. Adreste BP olmadığı için kesim **DS**.

Etkin ve Fiziksel Adres: [BX+SI+10h]

MOV AL, [BX+SI+10h]

0200h BX

DS = 1000h BX = 0200h SI = 0030h

Taban yazmacı BX = 0200h.

Etkin ve Fiziksel Adres: [BX+SI+10h]

MOV AL, [BX+SI+10h]

DS = 1000h BX = 0200h SI = 0030h

0200h BX

+ 0030h SI

İndis yazmacı SI = 0030h eklenir.

Etkin ve Fiziksel Adres: [BX+SI+10h]

```
MOV AL, [BX+SI+10h]
```

```
DS = 1000h BX = 0200h SI = 0030h
```

- 0200h BX
- + 0030h SI
- + 0010h yer değiştirme

Komuttaki 8 bitlik yer değiştirme 10h eklenir.

Etkin ve Fiziksel Adres: [BX+SI+10h]

MOV AL, [BX+SI+10h]

DS = 1000h BX = 0200h SI = 0030h

0200h

BX

+

0030h

SI

+

0010h

yer deęiřtirme

0240h

EA

Etkin adres EA = 0240h (16 bit).

Etkin ve Fiziksel Adres: [BX+SI+10h]

```
MOV AL, [BX+SI+10h]
```

DS = 1000h BX = 0200h SI = 0030h

0200h

BX

+

0030h

SI

+

0010h

yer deęiřtirme

0240h

EA

10000h

DS × 16

+

0240h

DS × 16 = 10000h ile EA toplanır.

Etkin ve Fiziksel Adres: **[BX+SI+10h]**

MOV AL, [BX+SI+10h]

DS = 1000h BX = 0200h SI = 0030h

0200h BX

+ 0030h SI

+ 0010h yer değiştirme

0240h EA

10000h DS × 16

+ 0240h

10240h fiziksel adres

Fiziksel adres = **10240h**; AL bu adresteki baytı alır.

Etkin / Fiziksel Adres Hesaplayıcı

taban BX ▼ indis SI ▼ yer değiştirme 10

BX 0200 BP FFF0 SI 0030 DI 0004

DS 1000 SS 2000 örnek varsayılan ▼

MOV AL, [BX+SI+10h]

EA (bağlı konum)
0240h

kesim
DS = 1000h

fiziksel adres
10240h

Mod: **taban indisli kaydırma** · DS × 16 + EA = 10000h + 0240h (varsayılan DS)

Modlar Bir Arada

```
► MOV BX, 0200h ; dolaysız
2 MOV SI, 2
3 MOV AL, [0200h] ; doğrudan
4 MOV AH, [BX+1] ; taban+kaydırma
5 ADD AL, [BX+SI] ; taban+indis
6 MOV [BX+SI+2], AL ; taban+indis+kayd.
7 MOV DX, AX ; yazmaç
```

0200h 0201h 0202h 0203h 0204h

10h

20h

30h

00h

00h

Anlık mod: BX = **0200h** (sabit komutun içinde).

Adım 1 / 7

DEĞİŞKENLER

BX = **0200h**

SI = **0000h**

AX = **0000h**

DX = **0000h**

Modlar Bir Arada

```
1 MOV BX, 0200h ; dolaysız
▶ MOV SI, 2
3 MOV AL, [0200h] ; doğrudan
4 MOV AH, [BX+1] ; taban+kaydırma
5 ADD AL, [BX+SI] ; taban+indis
6 MOV [BX+SI+2], AL ; taban+indis+kayd.
7 MOV DX, AX ; yazmaç
```

0200h	0201h	0202h	0203h	0204h
10h	20h	30h	00h	00h

SI = 0002h.

Adım 2 / 7

DEĞİŞKENLER

BX = 0200h

SI = 0002h

AX = 0000h

DX = 0000h

Modlar Bir Arada

```
1 MOV BX, 0200h      ; dolaysız
2 MOV SI, 2
3 ► MOV AL, [0200h]   ; doğrudan
4 MOV AH, [BX+1]      ; taban+kaydırma
5 ADD AL, [BX+SI]      ; taban+indis
6 MOV [BX+SI+2], AL    ; taban+indis+kayd.
7 MOV DX, AX           ; yazmaç
```

0200h 0201h 0202h 0203h 0204h

10h

20h

30h

00h

00h

Doğrudan mod: EA = 0200h → AL = **10h**.

Adım 3 / 7

DEĞİŞKENLER

BX = 0200h

SI = 0002h

AX = 0010h

DX = 0000h

Modlar Bir Arada

```
1 MOV BX, 0200h      ; dolaysız
2 MOV SI, 2
3 MOV AL, [0200h]     ; doğrudan
4 ► MOV AH, [BX+1]    ; taban+kaydırma
5 ADD AL, [BX+SI]      ; taban+indis
6 MOV [BX+SI+2], AL    ; taban+indis+kayd.
7 MOV DX, AX           ; yazmaç
```

0200h	0201h	0202h	0203h	0204h
10h	20h	30h	00h	00h

Taban + yer değiştirme: $EA = BX + 1 = 0201h \rightarrow AH = 20h$; $AX = 2010h$.

Adım 4 / 7

DEĞİŞKENLER

BX = 0200h

SI = 0002h

AX = 2010h

DX = 0000h

Modlar Bir Arada

```
1 MOV BX, 0200h      ; dolaysız
2 MOV SI, 2
3 MOV AL, [0200h]     ; doğrudan
4 MOV AH, [BX+1]      ; taban+kaydırma
▶ 5 ADD AL, [BX+SI]    ; taban+indis
6 MOV [BX+SI+2], AL   ; taban+indis+kayd.
7 MOV DX, AX          ; yazmaç
```

0200h 0201h 0202h 0203h 0204h

10h

20h

30h

00h

00h

Taban indisli: $EA = BX + SI = 0202h \rightarrow AL = 10h + 30h = 40h$.

Adım 5 / 7

DEĞİŞKENLER

BX = 0200h

SI = 0002h

AX = 2040h

DX = 0000h

Modlar Bir Arada

```
1 MOV BX, 0200h      ; dolaysız
2 MOV SI, 2
3 MOV AL, [0200h]     ; doğrudan
4 MOV AH, [BX+1]      ; taban+kaydırma
5 ADD AL, [BX+SI]      ; taban+indis
▶ MOV [BX+SI+2], AL    ; taban+indis+kayd.
7 MOV DX, AX           ; yazmaç
```

0200h 0201h 0202h 0203h 0204h

10h

20h

30h

00h

40h

Taban indisli yer değiştirme: $EA = 0200h + 2 + 2 = 0204h \leftarrow AL (40h)$.

Adım 6 / 7

DEĞİŞKENLER

BX = 0200h

SI = 0002h

AX = 2040h

DX = 0000h

Modlar Bir Arada

```
1 MOV BX, 0200h      ; dolaysız
2 MOV SI, 2
3 MOV AL, [0200h]     ; doğrudan
4 MOV AH, [BX+1]      ; taban+kaydırma
5 ADD AL, [BX+SI]      ; taban+indis
6 MOV [BX+SI+2], AL    ; taban+indis+kayd.
▶ MOV DX, AX          ; yazmaç
```

0200h 0201h 0202h 0203h 0204h

10h

20h

30h

00h

40h

Yazmaç modu: DX = AX = **2040h**; bellek erişimi yok.

Adım 7 / 7

DEĞİŞKENLER

BX = 0200h

SI = 0002h

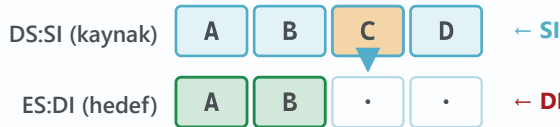
AX = 2040h

DX = 2040h

ADRESLEME MODLARI

Dize Modu

- Dize (string) komutları ile ilgilidir; işlenenler **örtüktür**, komutta yazılmaz.
- Kaynak **DS:SI** , hedef **ES:DI** .
- Her işlemden sonra SI ve DI, **yön bayrağına** bağlı olarak otomatik artar (DF = 0) ya da azalır (DF = 1).
- Örnekler: **MOVSB** (bayt), **MOVSW** (sözcük, ± 2).
- REP** öneki komutu CX kez tekrarlar;
REPE / **REPNE** ZF'ye de bakar.



MOVSB: ES:[DI] ← DS:[SI], sonra SI++ ve DI++ (DF = 0)

Komut	İşlem
MOVSB	ES:[DI] ← DS:[SI]
CMPSB	DS:[SI] – ES:[DI] (bayrak)
SCASB	AL – ES:[DI]
LODSB	AL ← DS:[SI]
STOSB	ES:[DI] ← AL

Komut adları assembler'da bitişik yazılır: MOVSB , MOVSW (MOVSB değil).

G/Ç Modu

- G/Ç (I/O) **portlarına** erişim içindir; 8086'nın bellekten ayrı 64 K'lık port adres alanı vardır.
- Veri her zaman **AL** (8 bit) ya da **AX** (16 bit) üzerinden aktarılır.
- **Sabit port** (0–255): port numarası komutta 8 bit olarak verilir.
- **Değişken port** (0–65535): port numarası **DX**'te verilir.

```
IN  AL, 45h      ; port 45h → AL
OUT 50h, AL      ; AL → port 50h
MOV DX, 03F8h
IN  AL, DX       ; port 03F8h → AL
OUT DX, AX       ; AX → port 03F8h
```

bellek: 00000h – FFFFFh (1 MB)

port 00h – FFh

100h – FFFFh

M/IO' = 1: MOV, ADD ...

M/IO' = 0: IN/OUT

sabit (0–255) ya da DX ile (0–65535)

8086'da doğru yazım `IN AL, 45h` ve `OUT 50h, AL` 'dir (`IN A, 45` 8085 tarzıdır) — OUT'ta önce port, sonra veri yazarı gelir.

Göreceli Mod

- Etkin adres, **komut işaretçisine** (IP) göre hesaplanır: komutta hedefin kendisi değil, **uzaklığı** bulunur.
- Kısa koşullu atlamalar (**JNZ** , **JC** ...) **işaretli 8 bit** uzaklık kullanır: -128 ... +127 bayt.
- Hesap, atlama komutundan **sonraki** komutun adresine göre yapılır: $IP = IP_{\text{sonraki}} + \text{kaydırma}$.

00FCh	DEC CX	
00FDh	...	
0100h	JNZ 00FCh	75 FAh
0102h	MOV AX, 1	

Döngünün sonundaki **JNZ** , 0100h adresinde 2 baytlık bir komuttur: 75h FAh.

Göreceli Mod

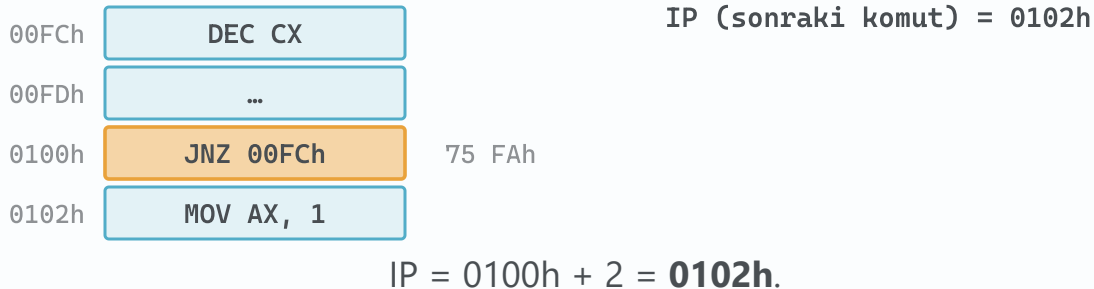
- Etkin adres, **komut işaretçisine** (IP) göre hesaplanır: komutta hedefin kendisi değil, **uzaklığı** bulunur.
- Kısa koşullu atlamalar (`JNZ` , `JC` ...) **işaretli 8 bit** uzaklık kullanır: -128 ... +127 bayt.
- Hesap, atlama komutundan **sonraki** komutun adresine göre yapılır: $IP = IP_{\text{sonraki}} + \text{kaydırma}$.

00FCh	DEC CX	
00FDh	...	
0100h	JNZ 00FCh	75 FAh
0102h	MOV AX, 1	

İşlemci JNZ'yi çalıştırırken IP zaten bir sonraki komutu gösterir.

Göreceli Mod

- Etkin adres, **komut işaretçisine** (IP) göre hesaplanır: komutta hedefin kendisi değil, **uzaklığı** bulunur.
- Kısa koşullu atlamalar (**JNZ** , **JC** ...) **işaretli 8 bit** uzaklık kullanır: -128 ... +127 bayt.
- Hesap, atlama komutundan **sonraki** komutun adresine göre yapılır: $IP = IP_{\text{sonraki}} + \text{kaydırma}$.



Göreceli Mod

- Etkin adres, **komut işaretçisine** (IP) göre hesaplanır: komutta hedefin kendisi değil, **uzaklığı** bulunur.
- Kısa koşullu atlamalar (`JNZ` , `JC` ...) **işaretli 8 bit** uzaklık kullanır: $-128 \dots +127$ bayt.
- Hesap, atlama komutundan **sonraki** komutun adresine göre yapılır: $IP = IP_{\text{sonraki}} + \text{kaydırma}$.

00FCh	DEC CX	
00FDh	...	
0100h	JNZ 00FCh	75 FAh
0102h	MOV AX, 1	

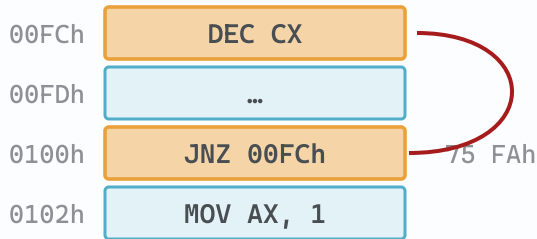
IP (sonraki komut) = 0102h

kaydırma FAh = -6

Uzaklık baytı FAh işaretli olarak -6'dır (Bölüm 3: ikiye tümleyen).

Göreceli Mod

- Etkin adres, **komut işaretçisine** (IP) göre hesaplanır: komutta hedefin kendisi değil, **uzaklığı** bulunur.
- Kısa koşullu atlamalar (`JNZ` , `JC` ...) **işaretli 8 bit** uzaklık kullanır: -128 ... +127 bayt.
- Hesap, atlama komutundan **sonraki** komutun adresine göre yapılır: $IP = IP_{\text{sonraki}} + \text{kaydırma}$.



IP (sonraki komut) = 0102h

kaydırma FAh = -6

$0102h + (-6) = 00FCh$

ZF = 0 ise $IP = 0102h - 6 = 00FCh$: döngü başına dönülür. ZF = 1 ise IP 0102h'de kalır.

ÖZET

Özet

Yazmaçlar

AX, BX, CX, DX ikiye bölünebilen veri yazmaçlarıdır; SP/BP yığını, SI/DI dizeleri ve indisleri, CS/DS/SS/ES kesimleri, IP sonraki komutu tutar.

Örtük görevler

MUL/DIV $\rightarrow$ AX, DX $\cdot$ LOOP/REP $\rightarrow$ CX $\cdot$ [BX]
 $\rightarrow$ DS $\cdot$ [BP] $\rightarrow$ SS $\cdot$ dize $\rightarrow$ DS:SI, ES:DI $\cdot$ port
 $\rightarrow$ DX.

Bayraklar

CF işaretsiz, OF işaretli taşıma; ZF sıfır; SF MSB; PF düşük baytın eşliği; AF nibble eldesi. DF, IF, TF kontrol bayraklarıdır.

Adresleme

EA = [BX|BP] + [SI|DI] + kaydırma; fiziksel adres = kesim $\times$ 16 + EA. Bellekten belleğe ve kesime sabit aktarım yoktur.

ÖZET

Alıştırmalar

1. AX = 3A7Fh iken `MOV AL, 0` ve ardından `INC AH` çalışınca AX kaç olur?
2. `MOV AL, 9Ch` / `ADD AL, 64h` sonrası AL, CF, ZF, SF, OF, PF değerlerini bulun.
3. `MOV AL, 50h` / `ADD AL, 50h` : CF ve OF kaçtır? Neden farklılar?
4. DS = 3000h, SS = 4000h, BX = 0100h, BP = 0020h, SI = 0008h için `[BX+SI+4]` ve `[BP+SI]` fiziksel adreslerini hesaplayın.
5. SP = 0100h iken 3 PUSH ve 1 POP sonrası SP kaçtır?
6. `MOV CX, 0` ile başlayan bir LOOP döngüsü kaç kez döner?
7. `MOV [BX], [SI]` neden derlenmez? İki komutla yazın.

► Cevaplar

Kaynaklar

- Intel, *The 8086 Family User's Manual*, 1979.
- Barry B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009.
- Muhammad Ali Mazidi, Janice G. Mazidi, *The x86 PC: Assembly Language, Design, and Interfacing*, 5. baskı, 2010.
- emu8086 belgeleri: *8086 instructions* ve *Addressing modes* bölümleri.
- Mehmet Bodur, *Mikroişlemcilere Giriş: Assembler ile Yazılım ve Arayüz*.

Son

Bölüm 2: 8086 Yazmaçlar

Sonraki bölüm: Bölüm 3: Sayı Sistemleri · [Terim sözlüğü](#)