Bölüm 2.2: 8086 Yazmaçları

Mikroişlemciler

Yazmaç kümesi | AX · BX · CX · DX | SP · BP · SI · DI | Kesim yazmaçları ve IP

Bayrak yazmacı | Durum ve kontrol bayrakları | Bayrakları değiştiren komutlar

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

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

CF, ZF, SF, OF, PF, AF bayraklarının hangi sonuçta kurulduğunu açıklamak; DF, IF, TF'nin işlemciyi nasıl yönettiğini bilmek.

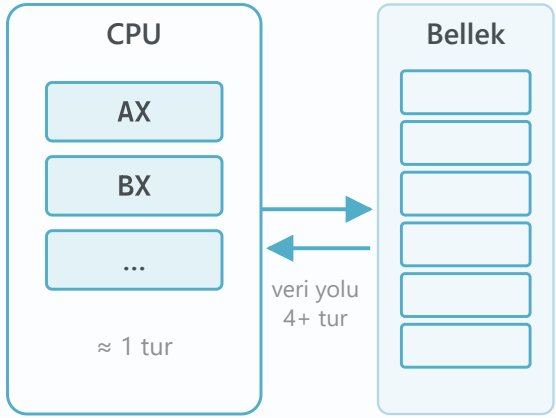
Yazmaç çiftleri

CS:IP, SS:SP, SS:BP, DS:SI ve ES:DI çiftlerinin hangi erişimde kullanıldığını bilmek.

YAZMAÇ KÜMESİ

Yazmaç Nedir?

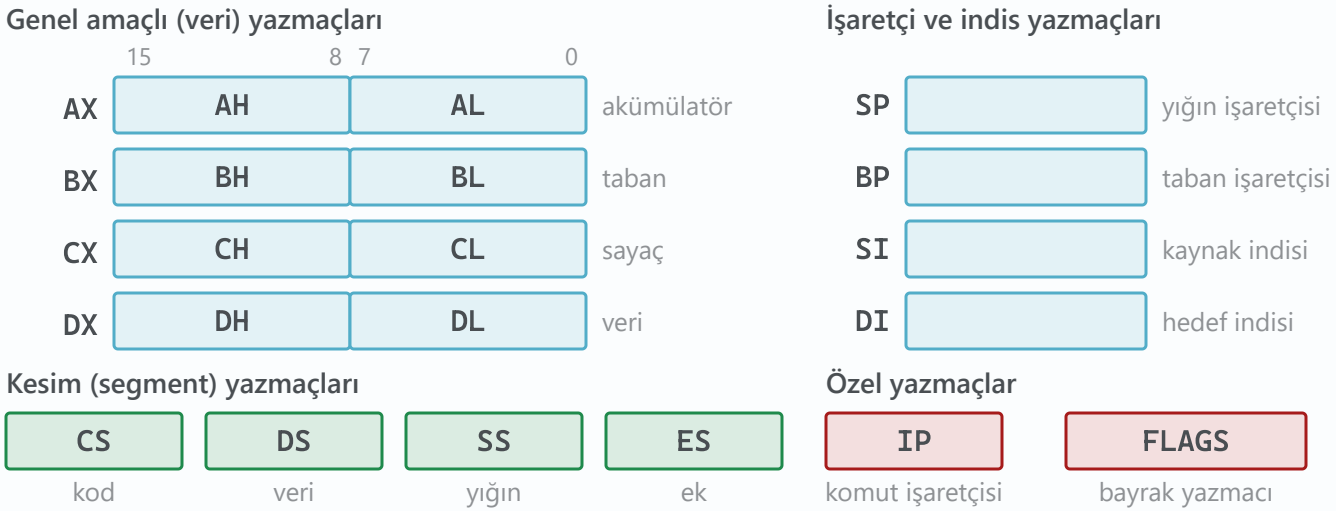
- **Yazmaç (register)**: işlemcinin *içinde* bulunan, çok hızlı, küçük bellek hücresidir.
- 8086'da yazmaçlar **16 bit** genişliğindedir; dört tanesi ayrıca 8 bitlik iki yarıya bölünebilir.
- Bellek erişimi dış veri yolu üzerinden birkaç saat turu sürer; yazmaç erişimi neredeyse **bedavadır**.
- Geçici 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



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.

YAZMAÇ KÜMESİ

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.

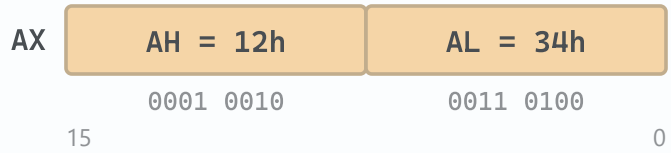
Yazmaç	Ad	8 bitlik yarılar	Örtük kullanım (örnek)
`AX`	Accumulator	AH, AL	MUL, DIV, IN, OUT, LODS/STOS
`BX`	Base	BH, BL	bellek adresleme `[BX]`, XLAT
`CX`	Counter	CH, CL	LOOP, REP, kaydırma sayısı CL
`DX`	Data	DH, DL	MUL/DIV yüksek yarısı, port adresi
`SP`	Stack Pointer	—	PUSH, POP, CALL, RET
`BP`	Base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

YAZMAÇ KÜMESİ · KODU ADIM ADIM ÇALIŞTIR

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

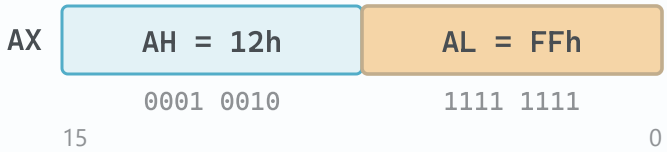
DEĞİŞKENLER

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

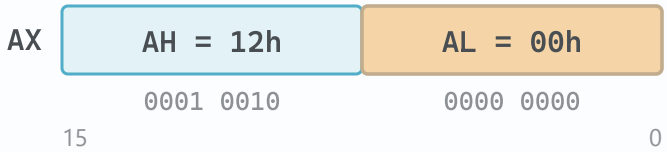
DEĞİŞKENLER

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
▶ 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.



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

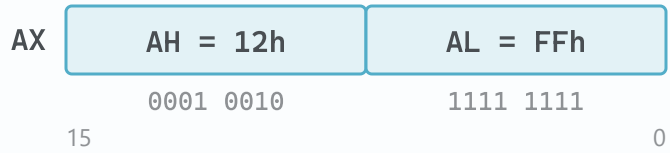
DEĞİŞKENLER

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

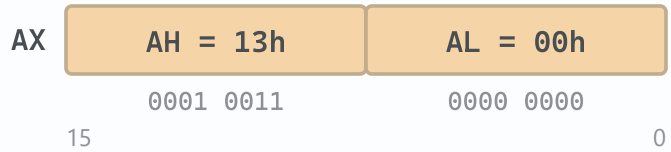
DEĞİŞKENLER

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`

GENEL AMAÇLI YAZMAÇLAR

AX — Akümülatör

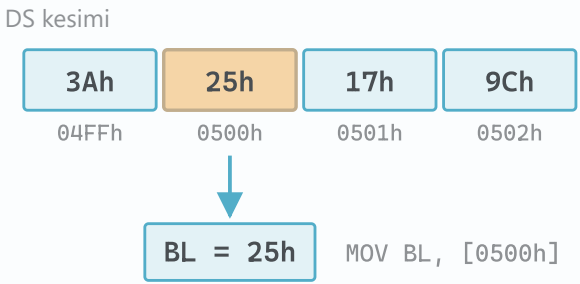
- 16 bit genişliğindedir; **AH** (yüksek sıra) ve **AL** (düşük sıra) olarak 8 bitlik iki bölüme ayrılır.
- 8 bitlik komutlarda AH ve AL ayrı ayrı kullanılabilir.
- Genellikle aritmetik ve mantıksal komutlarda kullanılır.
- Bazı komutlar AX/AL'yi **örtük** kullanır: `MUL`, `DIV`, `IN` / `OUT`, `CBW`, `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)

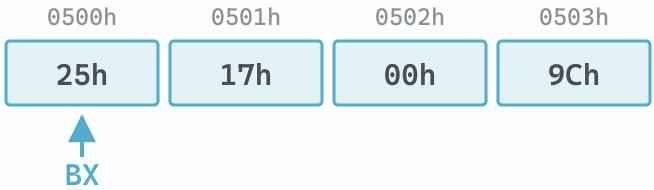
- 16 bit; **BH** ve **BL** olarak 8 bitlik iki yarıya ayrılır.
- Dolaylı bellek adreslemede **bağıl konum (offset)** değerini tutabilen tek veri yazmacıdır: `[BX]`, `[BX+SI]`, `[BX+4]`.
- Dizi ve tablo (matris) işlemlerinde başlangıç adresini tutmak için kullanılır.
- `[AX]`, `[CX]`, `[DX]` biçiminde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



`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.

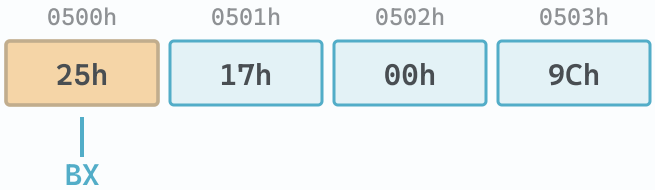
Adım 1 / 5

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**.

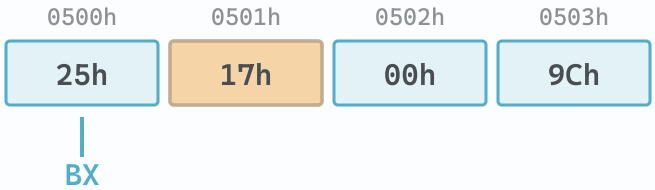
Adım 2 / 5

DEĞİŞKENLER

BX = **0500h** AL = **25h** DL = **00h**

BX ile Dolaylı Erişim

```
1 MOV BX, 0500h      ; BX = adres
2 MOV AL, [BX]       ; AL = [0500h]
▶ 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**.

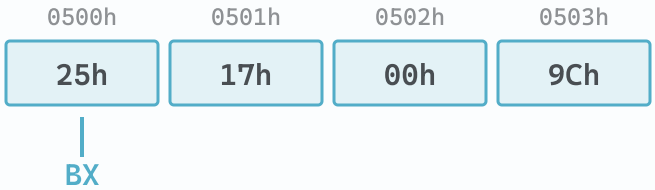
Adım 3 / 5

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
▶ ADD AL, DL         ; AL = 25h + 17h
5 MOV [BX+2], AL     ; [0502h] = AL
```



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

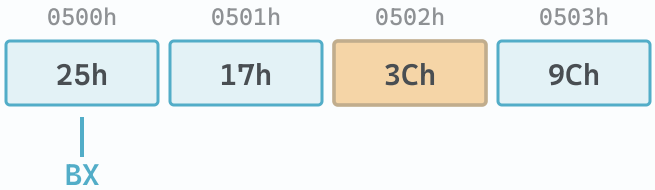
Adım 4 / 5

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
```

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
```

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

CX = 5: döngü 5 kez dönecek.

Adım 1 / 13

DEĞİŞKENLER

CX = 0005h | AX = 0000h | DX = 0000h

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
```

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

Toplam AX = 0 ile başlar.

Adım 2 / 13

DEĞİŞKENLER

CX = **0005h** AX = **0000h** DX = **0000h**

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
```

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

AX = AX + CX → 0 + 5 = **5**.

Adım 3 / 13

DEĞİŞKENLER

CX = **0005h** | AX = **0005h** | DX = **0000h**

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
```

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

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

Adım 4 / 13

DEĞİŞKENLER

CX = 0004h | AX = 0005h | DX = 0000h

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
```

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

AX = AX + CX → 5 + 4 = **9**.

Adım 5 / 13

DEĞİŞKENLER

CX = **0004h** AX = **0009h** DX = **0000h**

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
```

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

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

Adım 6 / 13

DEĞİŞKENLER

CX = 0003h | AX = 0009h | DX = 0000h

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
```

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

AX = AX + CX → 9 + 3 = **12**.

Adım 7 / 13

DEĞİŞKENLER

CX = **0003h** AX = **000Ch** DX = **0000h**

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
```

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

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

Adım 8 / 13

DEĞİŞKENLER

CX = 0002h | AX = 000Ch | DX = 0000h

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
```

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

AX = AX + CX → 12 + 2 = **14**.

Adım 9 / 13

DEĞİŞKENLER

CX = **0002h** AX = **000Eh** DX = **0000h**

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
```

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

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

Adım 10 / 13

DEĞİŞKENLER

CX = 0001h | AX = 000Eh | DX = 0000h

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
```

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

AX = AX + CX → 14 + 1 = **15**.

Adım 11 / 13

DEĞİŞKENLER

CX = **0001h** AX = **000Fh** DX = **0000h**

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
```

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

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

Adım 12 / 13

DEĞİŞKENLER

CX = 0000h | AX = 000Fh | DX = 0000h

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
```

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

Sonuç DX'e kopyalanır: 5+4+3+2+1 = 15 = **000Fh**.

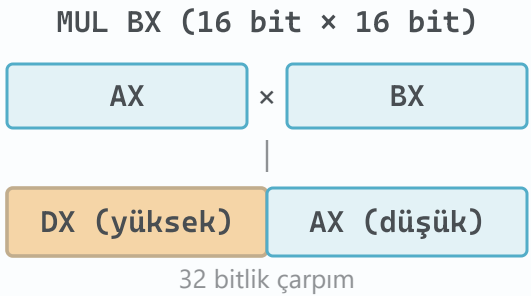
Adım 13 / 13

DEĞİŞKENLER

CX = 0000h | AX = 000Fh | DX = 000Fh

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) dizgi 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
```

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

1234h × 0100h = **00123400h** → 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 ile DX:AX

```
1 MOV AX, 1234h
2 MOV BX, 0100h
3 MUL BX          ; DX:AX = AX*BX
▶ 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
```

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

BL = 02h (BX = 0102h, BH değişmedi).

Adım 5 / 8

DEĞİŞKENLER

AX = **00FFh** | BX = **0102h** | DX = **0012h** | CF = **1** | OF = **1**

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
```

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

8 bitlik çarpım: FFh × 02h = **01FEh** → AX. DX değişmez; AH ≠ 0 olduğundan CF = OF = 1.

Adım 6 / 8

DEĞİŞKENLER

AX = **01FEh** | BX = **0102h** | DX = **0012h** | CF = **1** | OF = **1**

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

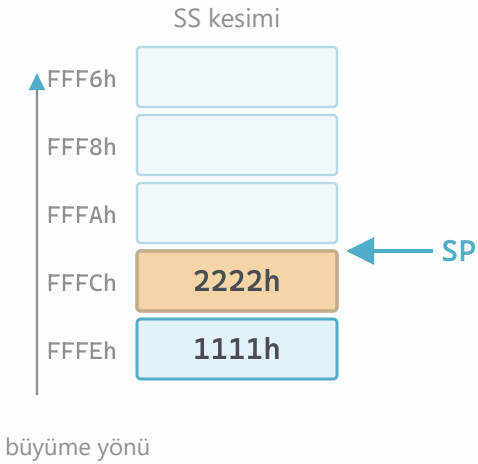
DEĞİŞKENLER

AX = **01FEh** BX = **0102h** DX = **03F8h** CF = **1** OF = **1**

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

SP — Yığın İşaretçisi

- 16 bitlik **Stack Pointer**; yığının **en üst** öğesinin bağıl konumunu (offset) tutar.
- Adres her zaman **yığın kesimine** göredir: fiziksel adres `SS:SP`.
- Yığın **LIFO** yapısıdır: `PUSH` ile eklenir, `POP` ile çıkarılır; 8086'da öğeler hep **16 bit**tir.
- 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.



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`.

SI ve DI — İndis Yazmaçları

SI — Source Index (kaynak indisi)

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

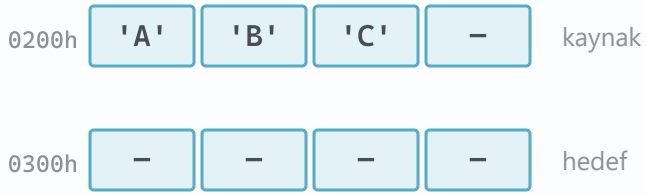
DI — Destination Index (hedef indisi)

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

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

SI ve DI ile Dizgi 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

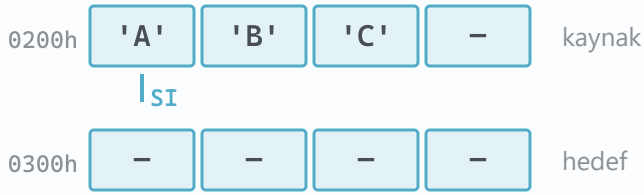
DEĞİŞKENLER

SI = **0000h** DI = **0000h** CX = **0000h**

DF = **0**

SI ve DI ile Dizgi Kopyalama

```
1 CLD              ; DF=0: artan adres
▶ 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” dizgisinin başı.

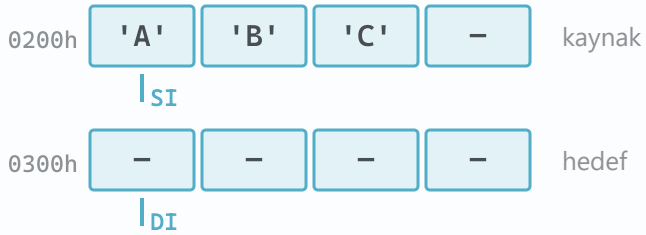
Adım 2 / 7

DEĞİŞKENLER

SI = 0200h DI = 0000h CX = 0000h DF = 0

SI ve DI ile Dizgi Kopyalama

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



DI = 0300h: hedef alanın başı.

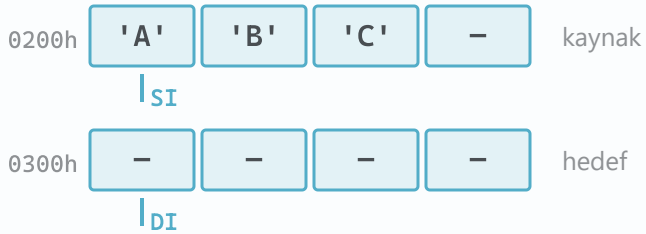
Adım 3 / 7

DEĞİŞKENLER

SI = 0200h DI = 0300h CX = 0000h DF = 0

SI ve DI ile Dizgi 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.

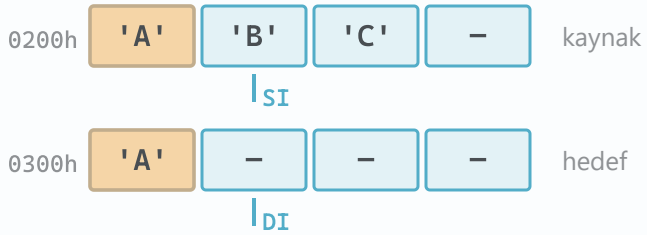
Adım 4 / 7

DEĞİŞKENLER

SI = 0200h DI = 0300h CX = 0003h DF = 0

SI ve DI ile Dizgi 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

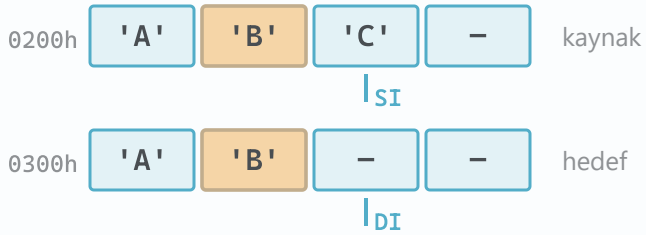
DEĞİŞKENLER

SI = **0201h** DI = **0301h** CX = **0002h**

DF = **0**

SI ve DI ile Dizgi 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.

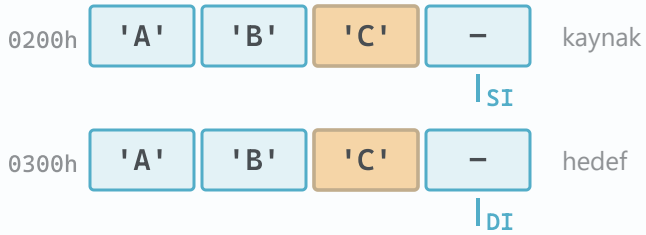
Adım 6 / 7

DEĞİŞKENLER

SI = 0202h DI = 0302h CX = 0001h DF = 0

SI ve DI ile Dizgi 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

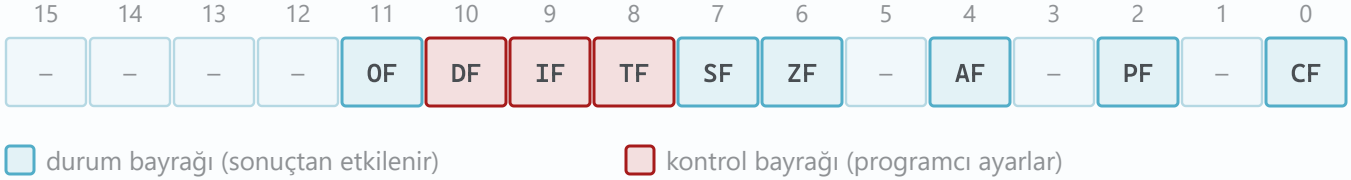
- **CS** (kod), **DS** (veri), **SS** (yığın), **ES** (ek): her biri 64 KB'lık bir kesimin başlangıcını tutar.
- **IP** (Instruction Pointer): CS içinde çalıştırılacak **sonraki komutun** bağıl konumu.
- IP'ye `MOV` ile yazılamaz; yalnızca JMP, CALL, RET, kesmeler değiştirir.
- Kesim yazmacına anlık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

Fiziksel adres (20 bit) = kesim × 16 + bağıl konum. 16 ile çarpmak, onaltılık sayının sonuna bir **0** eklemektir: DS = 2000h, BX = 0500h ise `[BX]` → 20000h + 0500h = **20500h**.

Bayrak Yazmacı

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



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.

BAYRAK YAZMACI

SF — İşaret Bayrağı

- Sign Flag, işlem sonucunun **negatif mi pozitif mi** olduğunu belirtir.
- Sonucun **en önemli biti** (MSB) kopyalanır:
- MSB = 1 → sayı (işaretli yorumda) negatiftir → **SF = 1**.
- MSB = 0 → sayı pozitif ya da sıfırdır → **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 ikiye tümleyen olarak (işaretli) yorumlamak programcının işidir.

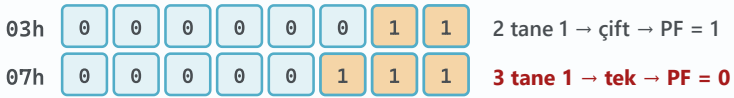
ZF ve PF

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)

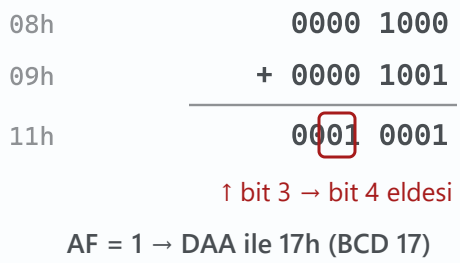
- Sonucun **düşük 8 bitindeki** 1'lerin sayısı **çift** ise PF = 1; tek ise PF = 0.
- 16 bitlik işlemde de yalnızca düşük bayta bakılır.
- Örnek: 03h = 0000 0011b → iki 1 → PF = 1.
- Örnek: 07h = 0000 0111b → üç 1 → PF = 0.



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.



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.

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 taşma 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

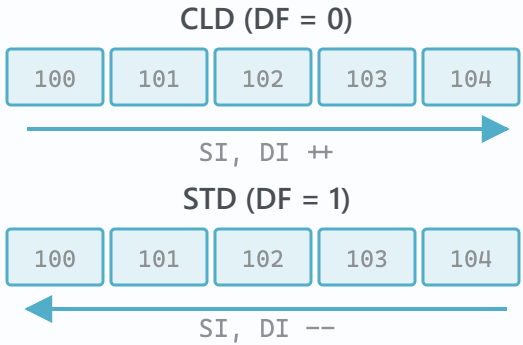
- Overflow Flag: sonuç, **işaretli** olarak yorumlandığında kullanılan bit sayısına **sığmıyorsa** OF = 1; aksi hâlde 0.
- 8 bitte işaretli aralık −128 … +127, 16 bitte −32768 … +32767'dir.
- Kural (toplama): iki **pozitif** toplanıp negatif, ya da iki **negatif** toplanıp pozitif sonuç çıkarsa OF = 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 ≠ −128	80h	0	**1**
`FFh + 01h`	255 + 1 ≠ 0	−1 + 1 = 0	00h	**1**	0
`80h + 80h`	128 + 128 ≠ 0	−128 + (−128) ≠ 0	00h	**1**	**1**

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

DF — Yön Bayrağı

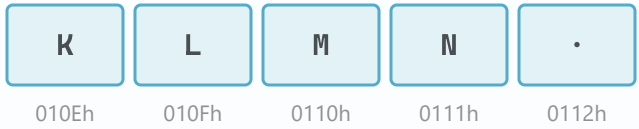
- Direction Flag, **dizgi (string)** komutlarında veri erişim yönünü belirler.
- **DF = 0**: düşük adresten yükseğe; her adımda SI/DI **artar**.
- **DF = 1**: yüksek adresten düşüğe; SI/DI **azalır**.
- Adım büyüklüğü bayt komutlarında 1, sözcük komutlarında 2'dir.
- `CLD` → DF = 0, `STD` → DF = 1.
-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



Kod İzleme: DF = 1 ile Örtüşen Kopyalama

```
1 org 100h
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

"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
▶ 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
```



DF = 0 · CX = 0

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

Adım 2 / 9

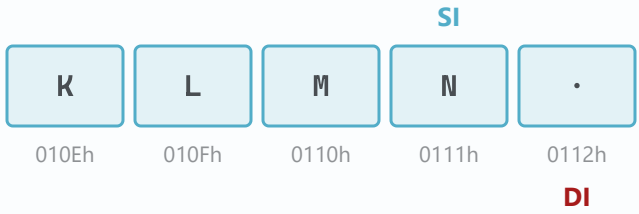
DEĞİŞKENLER

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

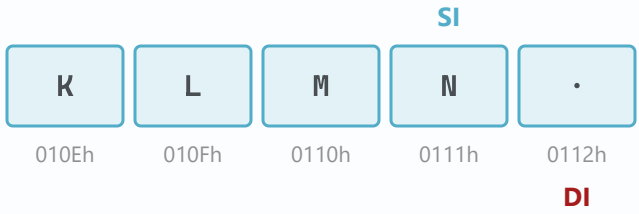
DEĞİŞKENLER

`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
▶ std                   ; DF = 1
6 rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh–0112h)



`DF = 1 · 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
▶ rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```

dizi (010Eh–0112h)



DF = 1 · CX = 3

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

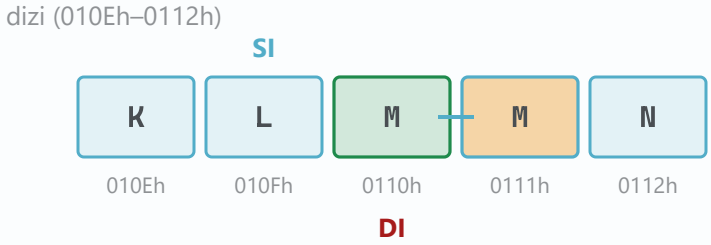
Adım 5 / 9

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
▶ rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```



DF = 1 · CX = 2

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

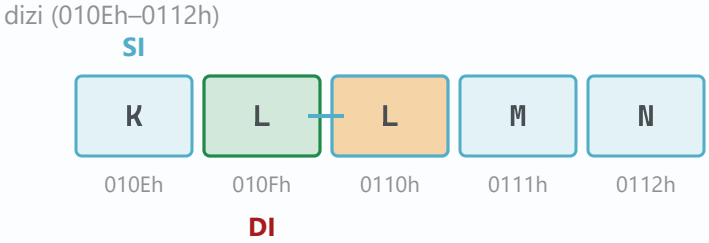
Adım 6 / 9

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
▶ rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```



DF = 1 · CX = 1

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

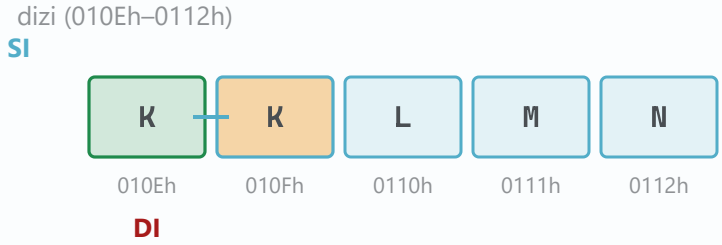
Adım 7 / 9

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
▶ rep movsb
7 cld
8 ret
9 dizi db 'KLMN', 0
```



`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
▶ cld
8 ret
9 dizi db 'KLMN', 0
```



`DF = 0 · CX = 0`

`cld` ile DF = 0'a dönülür. 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 değiştir → `POPF` .



Bayrakları Değiştiren Komutlar

Komut	Etki	Komut	Etki
`CLC`	CF = 0	`CLD`	DF = 0
`STC`	CF = 1	`STD`	DF = 1
`CMC`	CF = ¬CF	`CLI` / `STI`	IF = 0 / 1
`LAHF`	AH ← SF ZF – AF – PF – CF	`SAHF`	AH → 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
▶  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	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
▶  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
▶  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
▶  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
▶  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.

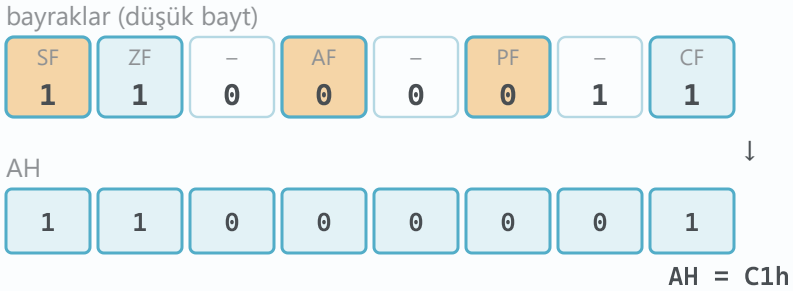
Adım 7 / 8

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
▶  sahf             ; bayraklar ← AH
10 ret
```



`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**

Laboratuvar: Yazmaçlar ve Bayraklar

AX = AH : AL

KOD · **14 bayt**

```
1 ORG 100h
2 MOV AX, 1234h   ; AH=12h, …
3 MOV AL, 0FFh    ; yalnız a…
4 ADD AL, 1       ; 8 bitlik…
5 MOV AX, 12FFh
6 ADD AX, 1       ; 16 bitli…
7 RET
```

YAZMAÇLAR

AX	0000	AH	00	AL	00
BX	0000	BH	00	BL	00
CX	000E	CH	00	CL	0E
DX	0000	DH	00	DL	00
SI	0000	DI	0000	BP	0000
SP	FFFE	CS	0700	IP	0100
DS	0700	ES	0700	SS	0700

BAYRAKLAR

O	D	I	T	S	Z	A	P	C
0	0	1	0	0	0	0	0	0

CS:IP = 0700:0100 · adım 0

BELLEK DS = 0700h

```
0100 B8 34 12 B0 FF 04 01 B8
0108 FF 12 05 01 00 C3 00 00
0110 00 00 00 00 00 00 00 00
0118 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00
```

YIĞIN SS:SP

```
FFF6 0000
FFF8 0000
FFFA 0000
FFFC 0000
FFFE 0000 ◂ SP
```

ÇIKTI

Program yüklendi: 14 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (000Eh) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

Yazmaçlar

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

Örtük görevler

MUL/DIV → AX, DX · LOOP/REP → CX · `[BX]` → DS · `[BP]` → SS · dizgi → DS:SI, ES:DI · port → DX.

Bayraklar

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

Yazmaç çiftleri

CS:IP sonraki komut, SS:SP yığının üstü, SS:BP yığın çerçevesi, DS:SI → ES:DI dizgi kopyalama. Kesim yazmacına sabit yüklenemez.

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. SP = 0100h iken 3 PUSH ve 1 POP sonrası SP kaçtır?
5. `MOV CX, 0` ile başlayan bir LOOP döngüsü kaç kez döner?

▶ **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 2.2: 8086 Yazmaçları

Sonraki bölüm: Bölüm 3.1: Programlama Dilleri · Terim sözlüğü