Bölüm 4.1: Bellek Erişimi

Mikroişlemciler

Adresleme modları · Yer değiştirme · Etkin adres · Kesim:bağıl konum · byte ptr / word ptr · MOV kuralları · Video belleği örneği · XCHG · 8/16 bit kopyalama · Sayı tabanları

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Adresleme modları

BX, SI, DI, BP ve yer değiştirme ile geçerli bellek işlenenleri oluşturmak.

Adres hesabı

Etkin adresi ve kesim:bağıl konumdan fiziksel adresi hesaplamak.

Veri boyutu

`byte ptr` / `word ptr` ile erişim boyutunu belirtmek; little-endian düzeni okumak.

MOV ve XCHG

Geçerli işlenen kombinasyonlarını ayırt etmek ve kopyalama/takas programlarını izlemek.

ADRESLEME MODLARI

Bellek Erişimi

- Belleğe erişmek için **BX, SI, DI, BP** yazmaçları kullanılır.
- Bu yazmaçlar (ve bir sabit) `[ ]` sembolleri içinde birleştirilerek farklı bellek konumlarına erişilir; bu birleştirmelere **adresleme modları** denir.

Yalnız yazmaç(lar)	+ d8 (8 bit yer değiştirme)	+ d16 (16 bit yer değiştirme)
`[BX + SI]`	`[BX + SI + d8]`	`[BX + SI + d16]`
`[BX + DI]`	`[BX + DI + d8]`	`[BX + DI + d16]`
`[BP + SI]`	`[BP + SI + d8]`	`[BP + SI + d16]`
`[BP + DI]`	`[BP + DI + d8]`	`[BP + DI + d16]`
`[SI]`	`[SI + d8]`	`[SI + d16]`
`[DI]`	`[DI + d8]`	`[DI + d16]`
`d16 (doğrudan adres)`	`[BP + d8]`	`[BP + d16]`
`[BX]`	`[BX + d8]`	`[BX + d16]`

24 kombinasyon, 8086 makine kodundaki mod/rm alanının tüm bellek seçenekleridir. `[BP]` tek başına yazılırsa assembler onu `[BP + 0]` (d8 = 0) olarak kodlar.

ADRESLEME MODLARI

Kombinasyon Kuralı

Taban		İndis		Yer değiştirme
BX	+	SI	+	d8
BP		DI		d16
yok		yok		yok

her sütundan en fazla bir seçim · BP seçilirse kesim SS, değilse DS

Kurallar

- **BX ve BP** aynı modda bir araya gelmez.
- **SI ve DI** aynı modda bir araya gelmez.
- SS kesimi BP ile, DS kesimi BP içermeyen modlarla kullanılır.

Örnekler

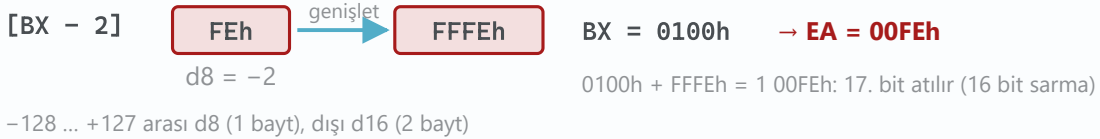
- `[BX+5]` → taban + d8 (DS)
- `[BX+SI]` → taban + indis (DS)
- `[DI+BX-4]` → `[BX+DI-4]`, sıra önemsiz (DS)
- `[BP+SI+300]` → taban + indis + d16 (SS)

Yer Değiştirme (Displacement)

d8
8 bit işaretli yer değiştirme (−128 … +127).
Örnek: `22`, `55h`, `-1`.

d16
16 bit işaretli yer değiştirme. Örnek: `300`, `5517h`, `-259`.

- Yer değiştirme bir sayı ya da bir **değişkenin bağıl konumu** (offset) olabilir: `[BX + dizi]`.
- Birden fazla değer varsa assembler **tek bir değer** hesaplar: `[BX + 2 + 3]` → `[BX + 5]`.
- Yer değiştirme `[ ]` içinde veya dışında yazılabilir; assembler aynı makine kodunu üretir: `[BX+5]` = `5[BX]` = `[BX]+5`.
- İşaretli bir değerdir; pozitif ya da negatif olabilir: `[SI-1]`.
- Assembler değer −128…127 aralığındaysa kısa (d8), değilse uzun (d16) biçimi seçer.



Etkin adres 16 bit ile hesaplanır ve **başa sarar**: BX = FFFFh iken `[BX+2]` → EA = 0001h.

Geçerli mi, Geçersiz mi?

Geçerli	Açıklama
`[BX]`	yazmaç dolaylı
`[1234h]`	doğrudan adres
`[SI+10]`	indisli
`[BP+DI-2]`	tabanlı-indisli (SS)
`ES:[BX+SI]`	kesim önekli

Geçersiz	Neden?
`[AX]`, `[CX]`, `[DX]`	adreslemede kullanılamaz
`[BL]`, `[BH]`	8 bitlik yazmaç adres olamaz
`[BX+BP]`	iki taban yazmacı
`[SI+DI]`	iki indis yazmacı
`[SP]`, `[BX*2]`	8086'da yok (80386+)

Tablo kullanılarak tüm geçerli kombinasyonlar oluşturulabilir: taban (BX/BP) + indis (SI/DI) + yer değiştirme, her biri isteğe bağlı.

ADRESLEME MODLARI

Adresleme Modu Türleri

Mod	Örnek	İşlenen nerede?
Anlık (immediate)	`MOV AX, 5`	komutun içinde (sabit)
Yazmaç (register)	`MOV AX, BX`	yazmaçta
Doğrudan (direct)	`MOV AX, [1000h]`	DS:1000h
Yazmaç dolaylı	`MOV AX, [BX]`	DS:BX
Tabanlı (based)	`MOV AX, [BP+4]`	SS:BP+4
İndisli (indexed)	`MOV AL, [SI+3]`	DS:SI+3
Tabanlı-indisli	`MOV AL, [BX+DI]`	DS:BX+DI
Tabanlı-indisli + yer değiştirme	`MOV AL, [BX+SI+8]`	DS:BX+SI+8

İlk ikisi belleğe gitmez; geri kalanlar etkin adres (EA) hesaplar. Fiziksel adres = kesim × 10h + EA.

ADRESLEME MODLARI

Dizgi, G/Ç ve Göreceli Mod

Üç mod daha vardır: işlenen komutta hiç yazılmaz, bellekten ayrı bir adres alanında bulunur ya da adres IP'ye göre hesaplanır.

Dizgi modu

Dizgi komutlarının işlenenleri **örtüktür**: kaynak `DS:SI`, hedef `ES:DI`. Her işlemden sonra SI ve DI, DF = 0 iken artar, DF = 1 iken azalır (bayt ±1, sözcük ±2).

```
MOVSB ; ES:[DI] ← DS:[SI]
```

G/Ç modu

Portlar bellekten ayrı, 64 K'lık bir adres alanındadır. Veri yalnız **AL** ya da **AX** üzerinden geçer; port 0–255 ise numarası komutta, daha büyükse **DX**'te verilir.

```
IN  AL, 45h
OUT DX, AX
```

Göreceli mod

Komutta hedefin kendisi değil, IP'ye eklenecek **işaretli uzaklık** bulunur: IP = sonraki komutun adresi + uzaklık. Kısa atlamalarda uzaklık 8 bittir (−128 … +127).

```
JNZ dongu ; 75 FA = −6
```

Komut adları bitişik yazılır (`MOVSB`, `MOVS B` değil). OUT'ta önce port, sonra veri yazmacı gelir: `OUT 50h, AL`; `IN A, 45` 8085 yazımıdır.

Kod İzleme: Altı Adresleme Modu

```
1  ORG 100h
▶  MOV BX, OFFSET tablo
3  MOV SI, 2
4  MOV AH, 7            ; anlık
5  MOV AL, [tablo+1]    ; doğrudan
6  MOV CL, [BX]         ; dolaylı
7  MOV CH, [BX+4]       ; tabanlı
8  MOV DL, [BX+SI]      ; taban+indis
9  MOV DH, [BX+SI+3]    ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,55h,66h
```

tablo (DS:0116h ...)

11h	22h	33h	44h	55h	66h
0116	0117	0118	0119	011A	011B

BX = 0116h SI = 0000h

AH	AL	CL
00h	00h	00h
CH	**DL**	**DH**
00h	00h	00h

BX = **0116h**: `tablo` 'nun bağıl konumu (OFFSET derleme anında hesaplanır).

Adım 1 / 8

DEĞİŞKENLER

BX = **0116h** SI = **0000h** AX = **0000h** CX = **0000h** DX = **0000h**

.COM programında DS = 0700h alındı; fiziksel adres = DS × 10h + EA.

Kod İzleme: Altı Adresleme Modu

```
1  ORG 100h
2  MOV BX, OFFSET tablo
▶  MOV SI, 2
4  MOV AH, 7            ; anlık
5  MOV AL, [tablo+1]    ; doğrudan
6  MOV CL, [BX]         ; dolaylı
7  MOV CH, [BX+4]       ; tabanlı
8  MOV DL, [BX+SI]      ; taban+indis
9  MOV DH, [BX+SI+3]    ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,55h,66h
```

tablo (DS:0116h ...)

11h	22h	33h	44h	55h	66h
0116	0117	0118	0119	011A	011B

BX = 0116h SI = 0002h

AH	AL	CL
00h	00h	00h
CH	DL	DH
00h	00h	00h

SI = 2: indis yazmacı.

Adım 2 / 8

DEĞİŞKENLER

BX = 0116h | SI = 0002h | AX = 0000h | CX = 0000h | DX = 0000h

.COM programında DS = 0700h alındı; fiziksel adres = DS × 10h + EA.

Kod İzleme: Altı Adresleme Modu

```
1  ORG 100h
2  MOV BX, OFFSET tablo
3  MOV SI, 2
▶  MOV AH, 7            ; anlık
5  MOV AL, [tablo+1]    ; doğrudan
6  MOV CL, [BX]         ; dolaylı
7  MOV CH, [BX+4]       ; tabanlı
8  MOV DL, [BX+SI]      ; taban+indis
9  MOV DH, [BX+SI+3]    ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,55h,66h
```

tablo (DS:0116h ...)

11h	22h	33h	44h	55h	66h
0116	0117	0118	0119	011A	011B

anlık

değer komutun içinde: 07h

AH	AL	CL
07h	00h	00h
CH	DL	DH
00h	00h	00h

Anlık mod: 7 komutun içindedir; belleğe gidilmez. AH = 07h.

Adım 3 / 8

DEĞİŞKENLER

BX = 0116h | SI = 0002h | AX = 0700h | CX = 0000h | DX = 0000h

.COM programında DS = 0700h alındı; fiziksel adres = DS × 10h + EA.

Kod İzleme: Altı Adresleme Modu

```
1  ORG 100h
2  MOV BX, OFFSET tablo
3  MOV SI, 2
4  MOV AH, 7           ; anlık
▶  MOV AL, [tablo+1]   ; doğrudan
6  MOV CL, [BX]        ; dolaylı
7  MOV CH, [BX+4]      ; tabanlı
8  MOV DL, [BX+SI]     ; taban+indis
9  MOV DH, [BX+SI+3]   ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,55h,66h
```

tablo (DS:0116h ...)

11h	22h	33h	44h	55h	66h
0116	0117	0118	0119	011A	011B

doğrudan

EA = tablo+1 = 0117h

fiziksel = 07000h + 0117h = 07117h

AH	AL	CL
07h	22h	00h
CH	**DL**	**DH**
00h	00h	00h

Doğrudan mod: adres komutun içinde: [0117h] → AL = **22h**.

Adım 4 / 8

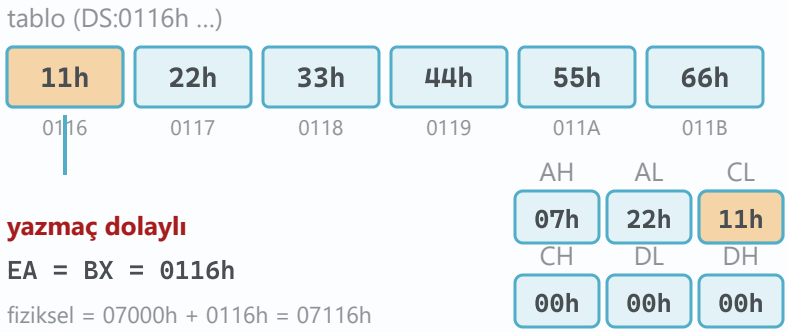
DEĞİŞKENLER

BX = 0116h | SI = 0002h | AX = 0722h | CX = 0000h | DX = 0000h

.COM programında DS = 0700h alındı; fiziksel adres = DS × 10h + EA.

Kod İzleme: Altı Adresleme Modu

```
1  ORG 100h
2  MOV BX, OFFSET tablo
3  MOV SI, 2
4  MOV AH, 7            ; anlık
5  MOV AL, [tablo+1]    ; doğrudan
▶  MOV CL, [BX]         ; dolaylı
7  MOV CH, [BX+4]       ; tabanlı
8  MOV DL, [BX+SI]      ; taban+indis
9  MOV DH, [BX+SI+3]    ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,55h,66h
```



Yazmaç dolaylı: EA = BX = 0116h → CL = **11h**.

Adım 5 / 8

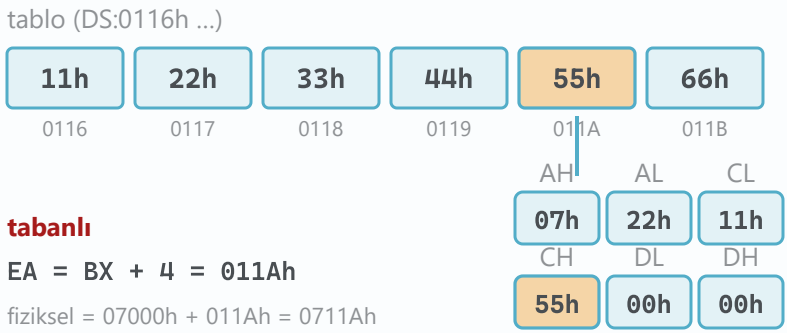
DEĞİŞKENLER

BX = 0116h | SI = 0002h | AX = 0722h | CX = 0011h | DX = 0000h

.COM programında DS = 0700h alındı; fiziksel adres = DS × 10h + EA.

Kod İzleme: Altı Adresleme Modu

```
1  ORG 100h
2  MOV BX, OFFSET tablo
3  MOV SI, 2
4  MOV AH, 7           ; anlık
5  MOV AL, [tablo+1]   ; doğrudan
6  MOV CL, [BX]        ; dolaylı
▶  MOV CH, [BX+4]      ; tabanlı
8  MOV DL, [BX+SI]     ; taban+indis
9  MOV DH, [BX+SI+3]   ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,55h,66h
```



tabanlı

`EA = BX + 4 = 011Ah`

fiziksel = 07000h + 011Ah = 0711Ah

Tabanlı: EA = BX + 4 = 011Ah → CH = **55h**.

Adım 6 / 8

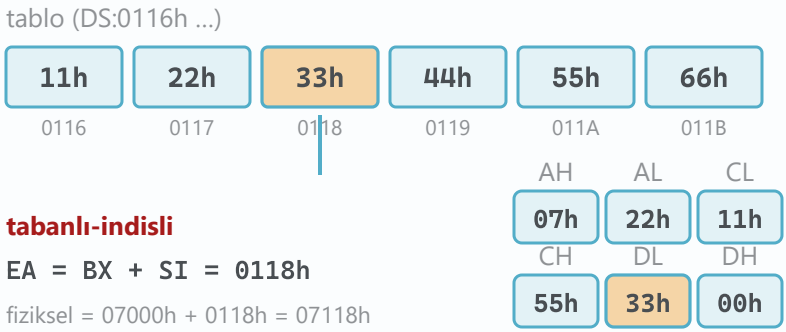
DEĞİŞKENLER

`BX = 0116h` `SI = 0002h` `AX = 0722h` `CX = 5511h` `DX = 0000h`

.COM programında DS = 0700h alındı; fiziksel adres = DS × 10h + EA.

Kod İzleme: Altı Adresleme Modu

```
1  ORG 100h
2  MOV BX, OFFSET tablo
3  MOV SI, 2
4  MOV AH, 7          ; anlık
5  MOV AL, [tablo+1]  ; doğrudan
6  MOV CL, [BX]       ; dolaylı
7  MOV CH, [BX+4]     ; tabanlı
▶  MOV DL, [BX+SI]    ; taban+indis
9  MOV DH, [BX+SI+3]  ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,55h,66h
```



Tabanlı-indisli: EA = BX + SI = 0118h → DL = **33h**.

Adım 7 / 8

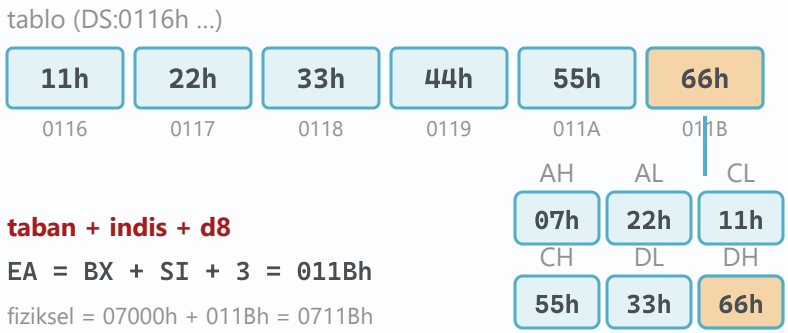
DEĞİŞKENLER

BX = **0116h** | SI = **0002h** | AX = **0722h** | CX = **5511h** | DX = **0033h**

.COM programında DS = 0700h alındı; fiziksel adres = DS × 10h + EA.

Kod İzleme: Altı Adresleme Modu

```
1  ORG 100h
2  MOV BX, OFFSET tablo
3  MOV SI, 2
4  MOV AH, 7           ; anlık
5  MOV AL, [tablo+1]   ; doğrudan
6  MOV CL, [BX]        ; dolaylı
7  MOV CH, [BX+4]      ; tabanlı
8  MOV DL, [BX+SI]     ; taban+indis
▶  MOV DH, [BX+SI+3]   ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,55h,66h
```



taban + indis + d8

EA = BX + SI + 3 = 011Bh

fiziksel = 07000h + 011Bh = 0711Bh

Taban + indis + d8: EA = BX + SI + 3 = 011Bh → DH = **66h**. BP kullanılmadığı için kesim hep DS.

Adım 8 / 8

DEĞİŞKENLER

BX = 0116h | SI = 0002h | AX = 0722h | CX = 5511h | DX = 6633h

.COM programında DS = 0700h alındı; fiziksel adres = DS × 10h + EA.

Etkin Adres Adım Adım

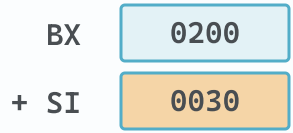
`MOV AL, [BX + SI + 4]` · DS = 1000h, BX = 0200h, SI = 0030h

BX `0200`

Taban yazmacı BX = 0200h.

Etkin Adres Adım Adım

`MOV AL, [BX + SI + 4]` · DS = 1000h, BX = 0200h, SI = 0030h



İndis yazmacı SI = 0030h eklenir.

Etkin Adres Adım Adım

`MOV AL, [BX + SI + 4]` · DS = 1000h, BX = 0200h, SI = 0030h



8 bitlik yer değiştirme: +4.

Etkin Adres Adım Adım

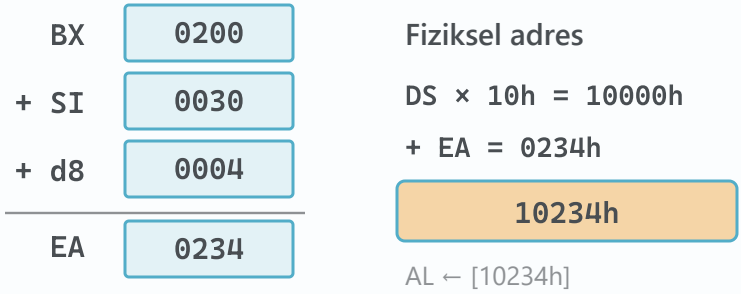
`MOV AL, [BX + SI + 4]` · DS = 1000h, BX = 0200h, SI = 0030h

BX	0200
+ SI	0030
+ d8	0004
EA	0234

Etkin adres EA = 0200h + 0030h + 0004h = **0234h** (16 bit).

Etkin Adres Adım Adım

`MOV AL, [BX + SI + 4]` · DS = 1000h, BX = 0200h, SI = 0030h



BP yok → kesim **DS**. Fiziksel adres = 10000h + 0234h = **10234h**. AL bu adresteki baytı alır.

BP ile Erişim

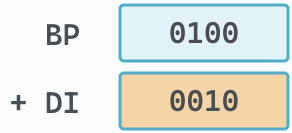
`MOV AX, [BP + DI − 2]` · SS = 2000h, BP = 0100h, DI = 0010h



Taban yazmacı BP = 0100h.

BP ile Erişim

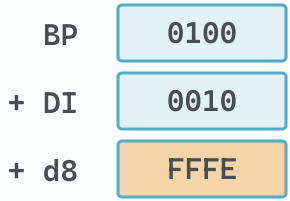
`MOV AX, [BP + DI − 2]` · SS = 2000h, BP = 0100h, DI = 0010h



İndis yazmacı DI = 0010h eklenir.

BP ile Erişim

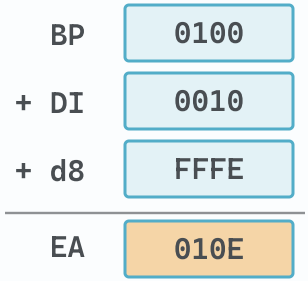
`MOV AX, [BP + DI − 2]` · SS = 2000h, BP = 0100h, DI = 0010h



İşaretli yer değiştirme −2 (makine kodunda FEh).

BP ile Erişim

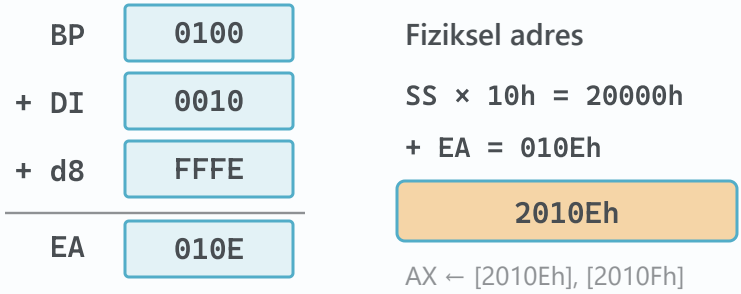
`MOV AX, [BP + DI − 2]` · SS = 2000h, BP = 0100h, DI = 0010h



EA = 0100h + 0010h − 2 = **010Eh**.

BP ile Erişim

`MOV AX, [BP + DI − 2]` · SS = 2000h, BP = 0100h, DI = 0010h



BP kullanıldığı için kesim **SS**: 20000h + 010Eh = **2010Eh**. AX ← [2010Eh] (2 bayt: 2010Eh ve 2010Fh).

ADRESLEME MODLARI

Etkin Adres Hesaplayıcı

[BX + SI + 4] varsayılan kesim

BX	BP	SI	DI
0200	0100	0030	0010

DS	ES	SS	CS
1000	B800	2000	0700

[BX + SI + 0004h]

EA = BX (0200h) + SI (0030h) + 0004h = **0234h**

DS × 10h + EA = 10000h + 0234h = **10234h**

Mod: **tabanlı-indisli + kaydırma** · kaydırma: d8 · kesim: **DS** (varsayılan DS)

Yazmaç değerleri onaltılık; yer değiştirme onluk (`-2`) ya da onaltılık (`1Ah`) girilebilir.

ADRESLEME MODLARI

Veri Yapısına Göre Adresleme Modu

Erişilen veri	Uygun mod	Örnek
Tek bir değişken	doğrudan	`MOV AX, sayac`
İşaretçinin gösterdiği eleman	yazmaç dolaylı	`MOV AL, [SI]`
Dizinin i. elemanı (dizi sabit, indis değişken)	indisli + d16	`MOV AL, dizi[SI]`
Bir kaydın (yapının) alanı	tabanlı + d8	`MOV AX, [BX+2]`
Matris ya da kayıt dizisi	taban + indis + d	`MOV AL, mat[BX+SI]`
Yığındaki parametre ya da yerel değişken	tabanlı, BP ile (SS)	`MOV AX, [BP+4]`
Başka kesimdeki tampon	kesim önekli	`MOV AL, ES:[DI]`

Taban = hangi kayıt / satır

BX ya da BP, yapının başlangıcını tutar; satır değişince yalnızca taban yazmacı güncellenir.

İndis = kayıt içindeki eleman

SI ya da DI döngüde artırılır; yer değiştirme dizinin ya da alanın sabit konumudur.

Aynı veriye birden çok modla ulaşılabilir; seçim, hangi parçanın sabit, hangisinin çalışma anında değişken olduğuna göre yapılır.

Kod İzleme: Matris Elemanına Erişim

```
1  org 100h
▶  mov bx, 2*4          ; satır 2
3  mov si, 1            ; sütun 1
4  mov al, mat[bx+si]   ; mat[2][1]
5  mov si, 3            ; sütun 3
6  mov ah, mat[bx+si]   ; mat[2][3]
7  sub bx, 4            ; satır 1
8  mov dl, mat[bx+si]   ; mat[1][3]
9  ret
10 mat db 11h,12h,13h,14h
11     db 21h,22h,23h,24h
12     db 31h,32h,33h,34h
```



BX = 2 × 4 = **0008h**: satır 2, matrisin başından 8 bayt ileride başlar (satır × satır uzunluğu).

Adım 1 / 8

DEĞİŞKENLER

`BX = 0008h` `SI = 0000h` `AX = 0000h` `DX = 0000h`

Satır öncelikli (row-major) düzen: eleman[satır][sütun] adresi = mat + satır × sütun sayısı + sütun. Sözcük matrisinde her iki terim ×2 alınır.

Kod İzleme: Matris Elemanına Erişim

```
1  org 100h
2  mov bx, 2*4          ; satır 2
3  mov si, 1            ; sütun 1
4  mov al, mat[bx+si]   ; mat[2][1]
5  mov si, 3            ; sütun 3
6  mov ah, mat[bx+si]   ; mat[2][3]
7  sub bx, 4            ; satır 1
8  mov dl, mat[bx+si]   ; mat[1][3]
9  ret
10 mat db 11h,12h,13h,14h
11     db 21h,22h,23h,24h
12     db 31h,32h,33h,34h
```



SI = 1: satır içindeki sütun. Taban yazmacı satırı, indis yazmacı sütunu seçer.

Adım 2 / 8

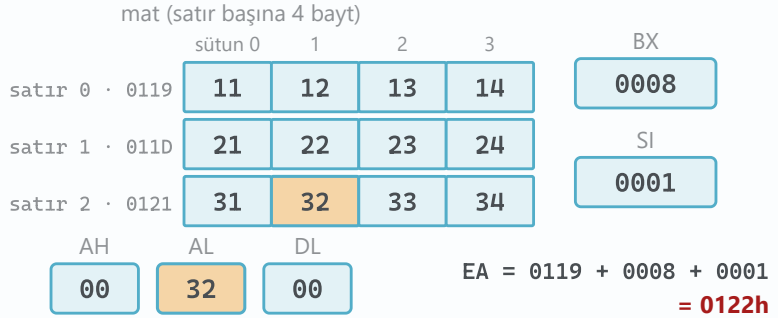
DEĞİŞKENLER

BX = 0008h | SI = 0001h | AX = 0000h | DX = 0000h

Satır öncelikli (row-major) düzen: eleman[satır][sütun] adresi = mat + satır × sütun sayısı + sütun. Sözcük matrisinde her iki terim ×2 alınır.

Kod İzleme: Matris Elemanına Erişim

```
1  org 100h
2  mov bx, 2*4          ; satır 2
3  mov si, 1            ; sütun 1
▶  mov al, mat[bx+si]   ; mat[2][1]
5  mov si, 3            ; sütun 3
6  mov ah, mat[bx+si]   ; mat[2][3]
7  sub bx, 4            ; satır 1
8  mov dl, mat[bx+si]   ; mat[1][3]
9  ret
10 mat db 11h,12h,13h,14h
11    db 21h,22h,23h,24h
12    db 31h,32h,33h,34h
```



`mat[bx+si]` : EA = 0119h + 8 + 1 = **0122h** → AL = **32h** (satır 2, sütun 1). Yer değiştirme `mat` 'ın bağıl konumudur (d16).

Adım 3 / 8

DEĞİŞKENLER

BX = **0008h** | SI = **0001h** | AX = **0032h** | DX = **0000h**

Satır öncelikli (row-major) düzen: eleman[satır][sütun] adresi = mat + satır × sütun sayısı + sütun. Sözcük matrisinde her iki terim ×2 alınır.

Kod İzleme: Matris Elemanına Erişim

```
1  org 100h
2  mov bx, 2*4          ; satır 2
3  mov si, 1            ; sütun 1
4  mov al, mat[bx+si]   ; mat[2][1]
▶  mov si, 3            ; sütun 3
6  mov ah, mat[bx+si]   ; mat[2][3]
7  sub bx, 4            ; satır 1
8  mov dl, mat[bx+si]   ; mat[1][3]
9  ret
10 mat db 11h,12h,13h,14h
11     db 21h,22h,23h,24h
12     db 31h,32h,33h,34h
```



SI = 3: aynı satırda başka bir sütun. BX değişmediği için satır sabit kaldı.

Adım 4 / 8

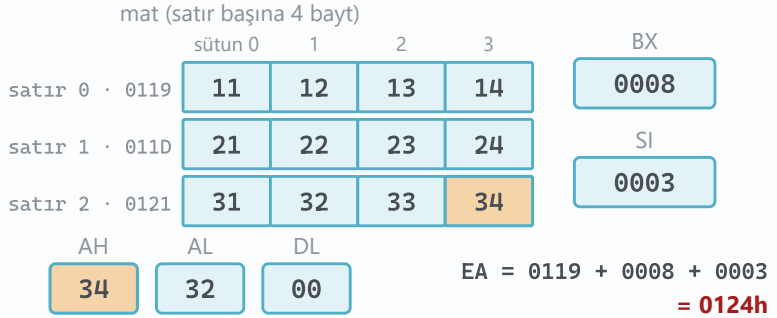
DEĞİŞKENLER

BX = 0008h | SI = 0003h | AX = 0032h | DX = 0000h

Satır öncelikli (row-major) düzen: eleman[satır][sütun] adresi = mat + satır × sütun sayısı + sütun. Sözcük matrisinde her iki terim ×2 alınır.

Kod İzleme: Matris Elemanına Erişim

```
1  org 100h
2  mov bx, 2*4          ; satır 2
3  mov si, 1            ; sütun 1
4  mov al, mat[bx+si]   ; mat[2][1]
5  mov si, 3            ; sütun 3
▶  mov ah, mat[bx+si]   ; mat[2][3]
7  sub bx, 4            ; satır 1
8  mov dl, mat[bx+si]   ; mat[1][3]
9  ret
10 mat db 11h,12h,13h,14h
11     db 21h,22h,23h,24h
12     db 31h,32h,33h,34h
```



EA = 0124h → AH = **34h**. Komut aynı, yalnızca indis değişti.

Adım 5 / 8

DEĞİŞKENLER

BX = 0008h | SI = 0003h | AX = 3432h | DX = 0000h

Satır öncelikli (row-major) düzen: eleman[satır][sütun] adresi = mat + satır × sütun sayısı + sütun. Sözcük matrisinde her iki terim ×2 alınır.

Kod İzleme: Matris Elemanına Erişim

```
1  org 100h
2  mov bx, 2*4          ; satır 2
3  mov si, 1            ; sütun 1
4  mov al, mat[bx+si]   ; mat[2][1]
5  mov si, 3            ; sütun 3
6  mov ah, mat[bx+si]   ; mat[2][3]
▶  sub bx, 4            ; satır 1
8  mov dl, mat[bx+si]   ; mat[1][3]
9  ret
10 mat db 11h,12h,13h,14h
11    db 21h,22h,23h,24h
12    db 31h,32h,33h,34h
```



`sub bx, 4` : bir satır yukarı (BX = 0004h). Sütun (SI = 3) korunur.

Adım 6 / 8

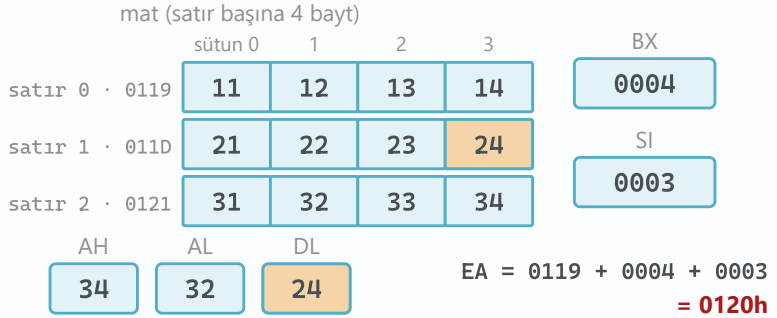
DEĞİŞKENLER

BX = 0004h | SI = 0003h | AX = 3432h | DX = 0000h

Satır öncelikli (row-major) düzen: eleman[satır][sütun] adresi = mat + satır × sütun sayısı + sütun. Sözcük matrisinde her iki terim ×2 alınır.

Kod İzleme: Matris Elemanına Erişim

```
1  org 100h
2  mov bx, 2*4          ; satır 2
3  mov si, 1            ; sütun 1
4  mov al, mat[bx+si]   ; mat[2][1]
5  mov si, 3            ; sütun 3
6  mov ah, mat[bx+si]   ; mat[2][3]
7  sub bx, 4            ; satır 1
▶  mov dl, mat[bx+si]   ; mat[1][3]
9  ret
10 mat db 11h,12h,13h,14h
11    db 21h,22h,23h,24h
12    db 31h,32h,33h,34h
```



EA = 0119h + 4 + 3 = 0120h → DL = **24h** (satır 1, sütun 3).

Adım 7 / 8

DEĞİŞKENLER

BX = **0004h** | SI = **0003h** | AX = **3432h** | DX = **0024h**

Satır öncelikli (row-major) düzen: eleman[satır][sütun] adresi = mat + satır × sütun sayısı + sütun. Sözcük matrisinde her iki terim ×2 alınır.

Kod İzleme: Matris Elemanına Erişim

```
1  org 100h
2  mov bx, 2*4          ; satır 2
3  mov si, 1            ; sütun 1
4  mov al, mat[bx+si]   ; mat[2][1]
5  mov si, 3            ; sütun 3
6  mov ah, mat[bx+si]   ; mat[2][3]
7  sub bx, 4            ; satır 1
8  mov dl, mat[bx+si]   ; mat[1][3]
▶  ret
10 mat db 11h,12h,13h,14h
11    db 21h,22h,23h,24h
12    db 31h,32h,33h,34h
```



`ret` : AX = **3432h**, DL = 24h. Üç eleman tek bir adresleme kalıbıyla okundu.

Adım 8 / 8

DEĞİŞKENLER

`BX = 0004h` `SI = 0003h` `AX = 3432h` `DX = 0024h`

Satır öncelikli (row-major) düzen: eleman[satır][sütun] adresi = mat + satır × sütun sayısı + sütun. Sözcük matrisinde her iki terim ×2 alınır.

ADRESLEME MODLARI

Etkin Adres Hesap Süresi

Etkin adres bileşimi	Saat turu
`[BX]`, `[BP]`, `[SI]`, `[DI]`	5
doğrudan `[d16]`	6
`[BP+DI]`, `[BX+SI]`	7
`[BP+SI]`, `[BX+DI]`	8
tek yazmaç + d8/d16	9
`[BP+DI+d]`, `[BX+SI+d]`	11
`[BP+SI+d]`, `[BX+DI+d]`	12
kesim öneki	+2

8086'da etkin adres, yürütme biriminde ayrı bir toplama ile hesaplanır; bu süre komutun kendi süresine eklenir.

- `MOV AX, BX` : 2 tur (bellek yok).
- `MOV CL, [BX]` : 8 + 5 = **13** tur.
- `MOV CL, [BX+DI+4]` : 8 + 12 = **20** tur.
- Tek adresteki bir sözcük, ikinci yol döngüsü için +4 tur daha ister.

Döngü içinde sık okunan değeri bir yazmaçta tutmak, her turda bu hesabı ortadan kaldırır.

Değerler Intel 8086 kullanıcı kılavuzundaki tablodandır; 80286 ve sonrasında adres hesabı ayrı bir birimde yapıldığı için bu farklar küçülür.

Kesim ve Bağıl Konum

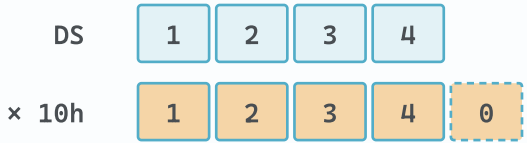
- Kesim yazmacındaki (CS, DS, SS, ES) değere **kesim** (segment), adresi veren yazmaçlardaki (BX, SI, DI, BP) değere **bağıl konum** (offset) denir.
- Kesim belleğin bir **bloğunu**, bağıl konum o bloktaki **belirli bir konumu** temsil eder; ikisi birleşerek fiziksel adresi oluşturur.



DS = **1234h**, SI = **7890h**. Gösterim: `1234:7890`.

Kesim ve Bağıl Konum

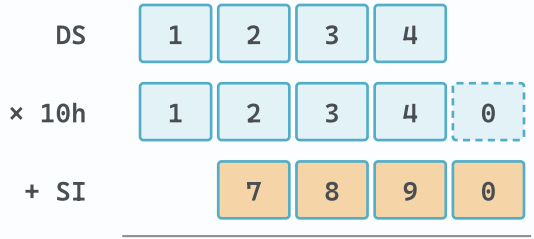
- Kesim yazmacındaki (CS, DS, SS, ES) değere **kesim** (segment), adresi veren yazmaçlardaki (BX, SI, DI, BP) değere **bağıl konum** (offset) denir.
- Kesim belleğin bir **bloğunu**, bağıl konum o bloktaki **belirli bir konumu** temsil eder; ikisi birleşerek fiziksel adresi oluşturur.



Kesim 10h ile çarpılır (4 bit sola kaydırma): **12340h**.

Kesim ve Bağıl Konum

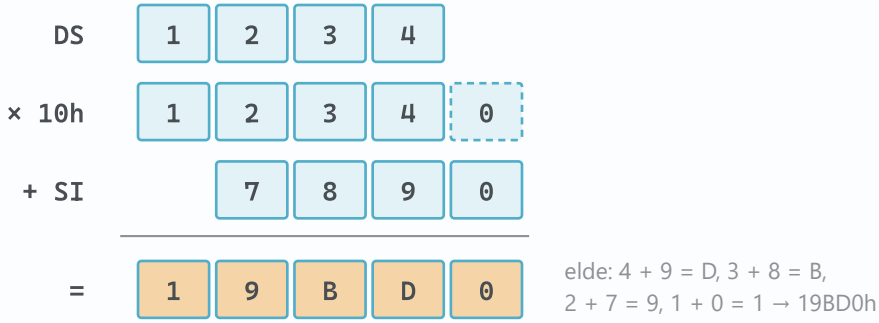
- Kesim yazmacındaki (CS, DS, SS, ES) değere **kesim** (segment), adresi veren yazmaçlardaki (BX, SI, DI, BP) değere **bağıl konum** (offset) denir.
- Kesim belleğin bir **bloğunu**, bağıl konum o bloktaki **belirli bir konumu** temsil eder; ikisi birleşerek fiziksel adresi oluşturur.



Bağıl konum eklenmek üzere hizalanır: **7890h**.

Kesim ve Bağıl Konum

- Kesim yazmacındaki (CS, DS, SS, ES) değere **kesim** (segment), adresi veren yazmaçlardaki (BX, SI, DI, BP) değere **bağıl konum** (offset) denir.
- Kesim belleğin bir **bloğunu**, bağıl konum o bloktaki **belirli bir konumu** temsil eder; ikisi birleşerek fiziksel adresi oluşturur.



12340h + 7890h = **19BD0h** fiziksel adres.

KESİM VE BAĞIL KONUM

Adres Hatları ve Bellek Kapasitesi

n adres hattı 2^n farklı konum seçer. Bir konumdaki bit sayısını veri hattı sayısı belirler:

kapasite = 2^n × veri genişliği

- 8086: 20 hat → 2^{20} = 1 048 576 konum = **1 MB** (00000h–FFFFFh).
- Bir kesimin sınırı bağıl konumdan gelir: 16 bit → 2^{16} = **64 KB**.
- Ters soru: en az kaç hat? Kapasiteyi aşan ilk 2'nin kuvveti alınır: 6 GB için 2^{32} = 4 GB yetmez, **33** hat (8 GB).

Adres / veri hattı	Kapasite
12 / 8	4096 × 1 B = 4 KB
16 / 4	64 K × 4 bit = 32 KB
16 / 8	64 KB (bir kesim)
20 / 8	1 MB (8086)
24 / 16	16 M × 2 B = 32 MB
32 / 8	4 GB (80386)

Bellek yongasında veri hattı sayısı sözcük genişliğidir; kapasite bayt cinsinden istenirse bit sayısı 8'e bölünür.

KESİM VE BAĞIL KONUM

Ters Hesap: Fiziksel Adresten DS:SI

Fiziksel adres **2A5C3h**'e DS:SI ile erişmek isteniyor. Tek bir cevap yoktur; DS'nin alabileceği aralık bulunur.

En büyük DS

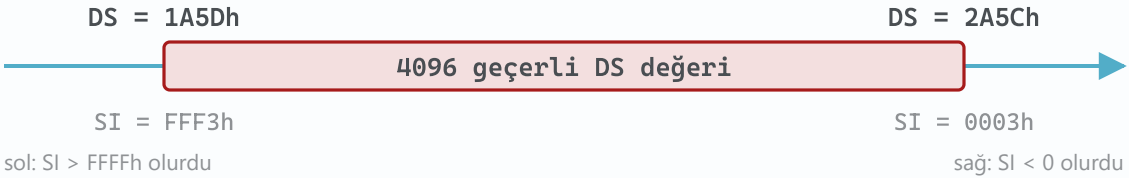
SI en küçük olsun: DS = ilk dört basamak = **2A5Ch**, SI = 2A5C3h − 2A5C0h = **0003h**.

En küçük DS

SI en çok FFFFh olabilir: DS × 10h ≥ 2A5C3h − FFFFh = 1A5C4h → DS ≥ 1A5C,4h → **1A5Dh**, SI = **FFF3h**.

Kaç farklı çift?

DS her artışta SI 10h azalır: 2A5Ch − 1A5Dh + 1 = 1000h = **4096** çift.



Denetim her zaman ileri hesapla yapılır: 1A5Dh × 10h + FFF3h = 1A5D0h + FFF3h = 2A5C3h.

Kesim Öneki

Varsayılan kesim, köşeli parantez önüne **kesim öneki** yazılarak değiştirilebilir:

```
MOV AX, 0B800h
MOV ES, AX          ; ES = video belleği
MOV BX, 0
MOV AL, ES:[BX]     ; ES:0000'dan oku
MOV AL, [BX]        ; DS:0000'dan oku
MOV DL, SS:[SI]     ; SS:SI'dan oku
```

Önek, makine kodunda komutun önüne eklenen 1 baytlık bir koddur (ES: → 26h, CS: → 2Eh, SS: → 36h, DS: → 3Eh).

Ne zaman gerekir?

- DS'yi değiştirmeden başka bir kesimdeki veriye (ör. video belleği) erişmek.
- Yığındaki veriye BP dışındaki bir yazmaçla erişmek: `SS:[BX]`.
- Koddaki bir tabloya erişmek: `CS:[SI]`.

IP her zaman CS ile, SP her zaman SS ile kullanılır; bunlar değiştirilemez.

Kod İzleme: ES ile Video Belleği

```
1 ORG 100h
▶ MOV AX, 0B800h
3 MOV ES, AX          ; ES = video
4 MOV BX, 00A0h       ; satır 1
5 MOV BYTE PTR ES:[BX], 'K'
6 MOV BYTE PTR ES:[BX+1], 1Eh
7 MOV AL, [BX]        ; DS:00A0h
8 MOV DL, ES:[BX]     ; B800:00A0h
9 RET
```



AX = B800h: renkli metin ekranının kesimi. Video belleğinde bir hücre = karakter + renk.

Adım 1 / 7

DEĞİŞKENLER

DS = 0700h · ES = 0700h · BX = 0000h · AL = 00h · DL = 00h

Başlangıçta video hücresi 20h 07h (boşluk, gri) kabul edildi. Önek yalnızca o komutu etkiler; DS değişmez.

Kod İzleme: ES ile Video Belleği

```
1 ORG 100h
2 MOV AX, 0B800h
▶ MOV ES, AX          ; ES = video
4 MOV BX, 00A0h       ; satır 1
5 MOV BYTE PTR ES:[BX], 'K'
6 MOV BYTE PTR ES:[BX+1], 1Eh
7 MOV AL, [BX]        ; DS:00A0h
8 MOV DL, ES:[BX]     ; B800:00A0h
9 RET
```



ES = **B800h**. DS hâlâ programın kesiminde (0700h): değişkenlere erişim bozulmaz.

Adım 2 / 7

DEĞİŞKENLER

DS = **0700h** | ES = **B800h** | BX = **0000h** | AL = **00h** | DL = **00h**

Başlangıçta video hücresi 20h 07h (boşluk, gri) kabul edildi. Önek yalnızca o komutu etkiler; DS değişmez.

Kod İzleme: ES ile Video Belleği

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV ES, AX          ; ES = video
▶ MOV BX, 00A0h       ; satır 1
5 MOV BYTE PTR ES:[BX], 'K'
6 MOV BYTE PTR ES:[BX+1], 1Eh
7 MOV AL, [BX]        ; DS:00A0h
8 MOV DL, ES:[BX]     ; B800:00A0h
9 RET
```

DS = 0700h

07000h + 00A0h = 070A0h

00h
+0

ES = B800h

B8000h + 00A0h = B80A0h

20h	07h
+0	+1

ekran, satır 1:

AL: 00h

DL: 00h

BX = 00A0h = 160 = (1 × 80 + 0) × 2: 1. satırın ilk hücresi.

Adım 3 / 7

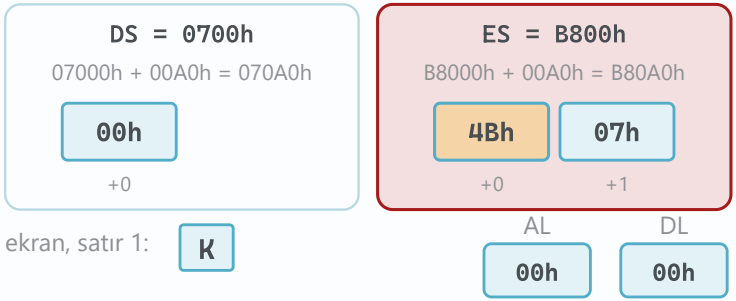
DEĞİŞKENLER

DS = 0700h · ES = B800h · BX = 00A0h · AL = 00h · DL = 00h

Başlangıçta video hücresi 20h 07h (boşluk, gri) kabul edildi. Önek yalnızca o komutu etkiler; DS değişmez.

Kod İzleme: ES ile Video Belleği

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV ES, AX          ; ES = video
4 MOV BX, 00A0h       ; satır 1
▶ MOV BYTE PTR ES:[BX], 'K'
6 MOV BYTE PTR ES:[BX+1], 1Eh
7 MOV AL, [BX]        ; DS:00A0h
8 MOV DL, ES:[BX]     ; B800:00A0h
9 RET
```



`ES:` öneki (makine kodunda 26h baytı): B8000h + 00A0h = **B80A0h** ← 'K' (4Bh).

Adım 4 / 7

DEĞİŞKENLER

`DS = 0700h` `ES = B800h` `BX = 00A0h` `AL = 00h` `DL = 00h`

Başlangıçta video hücresi 20h 07h (boşluk, gri) kabul edildi. Önek yalnızca o komutu etkiler; DS değişmez.

Kod İzleme: ES ile Video Belleği

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV ES, AX          ; ES = video
4 MOV BX, 00A0h       ; satır 1
5 MOV BYTE PTR ES:[BX], 'K'
▶ MOV BYTE PTR ES:[BX+1], 1Eh
7 MOV AL, [BX]        ; DS:00A0h
8 MOV DL, ES:[BX]     ; B800:00A0h
9 RET
```



B80A1h ← **1Eh**: mavi zemin üzerine sarı yazı. Ekranda sarı bir K belirir.

Adım 5 / 7

DEĞİŞKENLER

DS = **0700h** · ES = **B800h** · BX = **00A0h** · AL = **00h** · DL = **00h**

Başlangıçta video hücresi 20h 07h (boşluk, gri) kabul edildi. Önek yalnızca o komutu etkiler; DS değişmez.

Kod İzleme: ES ile Video Belleği

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV ES, AX          ; ES = video
4 MOV BX, 00A0h       ; satır 1
5 MOV BYTE PTR ES:[BX], 'K'
6 MOV BYTE PTR ES:[BX+1], 1Eh
▶ MOV AL, [BX]        ; DS:00A0h
8 MOV DL, ES:[BX]     ; B800:00A0h
9 RET
```



Önek yok → varsayılan **DS**: 07000h + 00A0h = 070A0h okunur (programın PSP alanı, burada 00h). Aynı bağıl konum, başka bayt.

Adım 6 / 7

DEĞİŞKENLER

DS = **0700h** · ES = **B800h** · BX = **00A0h** · AL = **00h** · DL = **00h**

Başlangıçta video hücresi 20h 07h (boşluk, gri) kabul edildi. Önek yalnızca o komutu etkiler; DS değişmez.

Kod İzleme: ES ile Video Belleği

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV ES, AX          ; ES = video
4 MOV BX, 00A0h       ; satır 1
5 MOV BYTE PTR ES:[BX], 'K'
6 MOV BYTE PTR ES:[BX+1], 1Eh
7 MOV AL, [BX]        ; DS:00A0h
▶ MOV DL, ES:[BX]     ; B800:00A0h
9 RET
```



`ES:[BX]` yeniden video belleğine gider: DL = **4Bh** ('K').

Adım 7 / 7

DEĞİŞKENLER

DS = **0700h** · ES = **B800h** · BX = **00A0h** · AL = **00h** · DL = **4Bh**

Başlangıçta video hücresi 20h 07h (boşluk, gri) kabul edildi. Önek yalnızca o komutu etkiler; DS değişmez.

Kod İzleme: BP ile Veriye Erişmek

```
 1 veri segment
 2 bos db 3 dup(0)
 3 x db 'D'
 4 veri ends
 5 yigin segment stack
 6 dw 64 dup(0)
 7 yigin ends
 8 kod segment
 9 assume cs:kod, ds:veri, ss:yigin
10 basla:
▶ mov ax, veri
12 mov ds, ax
13 mov bp, offset x
14 mov al, [bp]          ; SS:BP okunur!
15 mov bl, ds:[bp]       ; DS:BP
16 mov si, offset x
17 mov cl, [si]          ; DS:SI
18 mov ax, 4C00h
19 int 21h
20 kod ends
21 end basla
```

veri: DS = 0700h

fiz. 07100h…

0000	0001	0002	0003
00	00	00	44

yığın: SS = 0711h

fiz. 07110h…

0000	0001	0002	0003
00	00	00	00

BP	SI	AL	BL	CL
0000	0000	10	00	00

AX = **0710h**: `veri` kesiminin paragraf numarası (yükleyici belirler). EXE başlarken DS veriyi göstermez.

Adım 1 / 8

DEĞİŞKENLER

`DS = 0700h` `SS = 0711h` `BP = 0000h` `SI = 0000h` `AL = 10h` `BL = 00h` `CL = 00h`

.COM programında CS = DS = SS olduğu için bu hata görünmez; ayrı yığın kesimli .EXE programında ortaya çıkar.

Kod İzleme: BP ile Veriye Erişmek

```
 1 veri segment
 2 bos db 3 dup(0)
 3 x db 'D'
 4 veri ends
 5 yigin segment stack
 6 dw 64 dup(0)
 7 yigin ends
 8 kod segment
 9 assume cs:kod, ds:veri, ss:yigin
10 basla:
11 mov ax, veri
▶ mov ds, ax
13 mov bp, offset x
14 mov al, [bp]          ; SS:BP okunur!
15 mov bl, ds:[bp]       ; DS:BP
16 mov si, offset x
17 mov cl, [si]          ; DS:SI
18 mov ax, 4C00h
19 int 21h
20 kod ends
21 end basla
```

veri: DS = 0710h

fiz. 07100h…

0000	0001	0002	0003
00	00	00	44

yığın: SS = 0711h

fiz. 07110h…

0000	0001	0002	0003
00	00	00	00

BP	SI	AL	BL	CL
0000	0000	10	00	00

DS = 0710h. Yığın kesimi SS = **0711h** ayrı bir bloktur; bu programda DS ≠ SS.

Adım 2 / 8

DEĞİŞKENLER

DS = 0710h | SS = 0711h | BP = 0000h | SI = 0000h | AL = 10h | BL = 00h | CL = 00h

.COM programında CS = DS = SS olduğu için bu hata görünmez; ayrı yığın kesimli .EXE programında ortaya çıkar.

Kod İzleme: BP ile Veriye Erişmek

```
 1 veri segment
 2 bos db 3 dup(0)
 3 x db 'D'
 4 veri ends
 5 yigin segment stack
 6 dw 64 dup(0)
 7 yigin ends
 8 kod segment
 9 assume cs:kod, ds:veri, ss:yigin
10 basla:
11 mov ax, veri
12 mov ds, ax
▶ mov bp, offset x
14 mov al, [bp]          ; SS:BP okunur!
15 mov bl, ds:[bp]       ; DS:BP
16 mov si, offset x
17 mov cl, [si]          ; DS:SI
18 mov ax, 4C00h
19 int 21h
20 kod ends
21 end basla
```

veri: DS = 0710h

fiz. 07100h…

0000	0001	0002	0003
00	00	00	44

yığın: SS = 0711h

fiz. 07110h…

0000	0001	0002	0003
00	00	00	00

BP	SI	AL	BL	CL
0003	0000	10	00	00

BP = **0003h**: `x`'in bağıl konumu. Adres doğru, sorun kesim seçiminde.

Adım 3 / 8

DEĞİŞKENLER

DS = 0710h | SS = 0711h | BP = 0003h | SI = 0000h | AL = 10h | BL = 00h | CL = 00h

.COM programında CS = DS = SS olduğu için bu hata görünmez; ayrı yığın kesimli .EXE programında ortaya çıkar.

Kod İzleme: BP ile Veriye Erişmek

```
 1 veri segment
 2 bos db 3 dup(0)
 3 x db 'D'
 4 veri ends
 5 yigin segment stack
 6 dw 64 dup(0)
 7 yigin ends
 8 kod segment
 9 assume cs:kod, ds:veri, ss:yigin
10 basla:
11 mov ax, veri
12 mov ds, ax
13 mov bp, offset x
▶ mov al, [bp]          ; SS:BP okunur!
15 mov bl, ds:[bp]      ; DS:BP
16 mov si, offset x
17 mov cl, [si]         ; DS:SI
18 mov ax, 4C00h
19 int 21h
20 kod ends
21 end basla
```



`[bp]` varsayılan olarak **SS** ile birleşir: 07113h okunur, AL = **00h**. 'D' değil, yığındaki bir bayt geldi.

Adım 4 / 8

DEĞİŞKENLER

DS = 0710h | SS = 0711h | BP = 0003h | SI = 0000h | AL = 00h | BL = 00h | CL = 00h

.COM programında CS = DS = SS olduğu için bu hata görünmez; ayrı yığın kesimli .EXE programında ortaya çıkar.

Kod İzleme: BP ile Veriye Erişmek

```
 1 veri segment
 2 bos db 3 dup(0)
 3 x db 'D'
 4 veri ends
 5 yigin segment stack
 6 dw 64 dup(0)
 7 yigin ends
 8 kod segment
 9 assume cs:kod, ds:veri, ss:yigin
10 basla:
11 mov ax, veri
12 mov ds, ax
13 mov bp, offset x
14 mov al, [bp]          ; SS:BP okunur!
▶ mov bl, ds:[bp]        ; DS:BP
16 mov si, offset x
17 mov cl, [si]          ; DS:SI
18 mov ax, 4C00h
19 int 21h
20 kod ends
21 end basla
```

veri: DS = 0710h

fiz. 07100h…

0000	0001	0002	0003
00	00	00	44

yığın: SS = 0711h

fiz. 07110h…

0000	0001	0002	0003
00	00	00	00

BP	SI	AL	BL	CL
0003	0000	00	44	00

DS:0003 → 07103h

`ds:[bp]` : önek varsayılanı değiştirir, 07103h okunur → BL = **44h** ('D').

Adım 5 / 8

DEĞİŞKENLER

DS = 0710h | SS = 0711h | BP = 0003h | SI = 0000h | AL = 00h | BL = 44h | CL = 00h

.COM programında CS = DS = SS olduğu için bu hata görünmez; ayrı yığın kesimli .EXE programında ortaya çıkar.

Kod İzleme: BP ile Veriye Erişmek

```
 1 veri segment
 2 bos db 3 dup(0)
 3 x db 'D'
 4 veri ends
 5 yigin segment stack
 6 dw 64 dup(0)
 7 yigin ends
 8 kod segment
 9 assume cs:kod, ds:veri, ss:yigin
10 basla:
11 mov ax, veri
12 mov ds, ax
13 mov bp, offset x
14 mov al, [bp]          ; SS:BP okunur!
15 mov bl, ds:[bp]       ; DS:BP
▶ mov si, offset x
17 mov cl, [si]          ; DS:SI
18 mov ax, 4C00h
19 int 21h
20 kod ends
21 end basla
```

veri: DS = 0710h

fiz. 07100h…

0000	0001	0002	0003
00	00	00	44

yığın: SS = 0711h

fiz. 07110h…

0000	0001	0002	0003
00	00	00	00

BP	SI	AL	BL	CL
0003	0003	00	44	00

Aynı bağıl konum SI'ya alınır: SI = 0003h.

Adım 6 / 8

DEĞİŞKENLER

DS = 0710h · SS = 0711h · BP = 0003h · SI = 0003h · AL = 00h · BL = 44h · CL = 00h

.COM programında CS = DS = SS olduğu için bu hata görünmez; ayrı yığın kesimli .EXE programında ortaya çıkar.

Kod İzleme: BP ile Veriye Erişmek

```
 1 veri segment
 2 bos db 3 dup(0)
 3 x db 'D'
 4 veri ends
 5 yigin segment stack
 6 dw 64 dup(0)
 7 yigin ends
 8 kod segment
 9 assume cs:kod, ds:veri, ss:yigin
10 basla:
11 mov ax, veri
12 mov ds, ax
13 mov bp, offset x
14 mov al, [bp]          ; SS:BP okunur!
15 mov bl, ds:[bp]       ; DS:BP
16 mov si, offset x
▶ mov cl, [si]           ; DS:SI
18 mov ax, 4C00h
19 int 21h
20 kod ends
21 end basla
```

veri: DS = 0710h

fiz. 07100h…

0000	0001	0002	0003
00	00	00	44

yığın: SS = 0711h

fiz. 07110h…

0000	0001	0002	0003
00	00	00	00

BP	SI	AL	BL	CL
0003	0003	00	44	44

DS:0003 → 07103h

`[si]` varsayılan olarak **DS** kullanır: CL = **44h**. Veri için BX/SI/DI, yığın için BP seçilir.

Adım 7 / 8

DEĞİŞKENLER

DS = 0710h · SS = 0711h · BP = 0003h · SI = 0003h · AL = 00h · BL = 44h · CL = 44h

.COM programında CS = DS = SS olduğu için bu hata görünmez; ayrı yığın kesimli .EXE programında ortaya çıkar.

Kod İzleme: BP ile Veriye Erişmek

```
 1 veri segment
 2 bos db 3 dup(0)
 3 x db 'D'
 4 veri ends
 5 yigin segment stack
 6 dw 64 dup(0)
 7 yigin ends
 8 kod segment
 9 assume cs:kod, ds:veri, ss:yigin
10 basla:
11 mov ax, veri
12 mov ds, ax
13 mov bp, offset x
14 mov al, [bp]          ; SS:BP okunur!
15 mov bl, ds:[bp]       ; DS:BP
16 mov si, offset x
17 mov cl, [si]          ; DS:SI
▶ mov ax, 4C00h
▶ int 21h
20 kod ends
21 end basla
```

veri: DS = 0710h

fiz. 07100h…

0000	0001	0002	0003
00	00	00	44

yığın: SS = 0711h

fiz. 07110h…

0000	0001	0002	0003
00	00	00	00

BP	SI	AL	BL	CL
0003	0003	00	44	44

AH = 4Ch, AL = 0: program dönüş kodu 0 ile DOS'a döner.

Adım 8 / 8

DEĞİŞKENLER

DS = 0710h | SS = 0711h | BP = 0003h | SI = 0003h | AL = 00h | BL = 44h | CL = 44h

.COM programında CS = DS = SS olduğu için bu hata görünmez; ayrı yığın kesimli .EXE programında ortaya çıkar.

KESİM VE BAĞIL KONUM

Kesim Yazmacını Değiştirmek

```
PUSH DS            ; DS'yi sakla
MOV  AX, 0B800h
MOV  DS, AX        ; geçici kesim
MOV  BYTE PTR [0], 'Z'
POP  DS            ; değişkenler geri

MOV  AX, yeni_yigin
MOV  SS, AX        ; ardından hemen
MOV  SP, 0400h     ; SP kurulur
```

DS

Değişken adları yalnızca bağıl konumdur; DS değişince aynı ad başka kesimdeki bir bayta gider. Geçici değişiklik PUSH DS / POP DS ile sarılır ya da ES kullanılır.

SS ve SP

Birlikte değişir. 8086, kesim yazmacına MOV/POP yapan komuttan sonraki komut bitene kadar kesme kabul etmez; böylece SS:SP çifti yarım kalmaz.

CS

MOV ile yazılmaz; uzak JMP/CALL/RET ya da INT/IRET CS ve IP'yi birlikte değiştirir.

ES, varsayılan olarak yalnızca dizgi hedefinde kullanıldığı için ikinci bir veri bloğuna (video belleği, tampon) erişmenin en güvenli yoludur.

Veri Tipi Belirtme

- Derleyiciye erişilecek verinin boyutunu bildirmek için **ön ekler** kullanılır:
  - `byte ptr [BX]` : bir bayt (8 bit) erişimi.
  - `word ptr [BX]` : iki bayt (16 bit, bir sözcük) erişimi.
- emu8086 daha kısa önekleri de destekler:
  - `b.[SI]` : SI'nın gösterdiği adresteki değeri bir bayt olarak yorumlar.
  - `w.[DI]` : DI'nın gösterdiği adresteki değeri iki bayt olarak yorumlar.
- Bazı durumlarda assembler türü **kendisi** çıkarır: diğer işlenen bir yazmaçsa boyutu odur.



`byte ptr [BX]` → `34h`

`word ptr [BX]` → `1234h` (düşük bayt düşük adreste)

Komut	Boyut nereden belli?
`MOV [BX], AL`	AL 8 bit → bayt yazılır
`MOV [BX], AX`	AX 16 bit → sözcük yazılır
`MOV [BX], 5`	**belirsiz!** → `MOV BYTE PTR [BX], 5` ya da `MOV WORD PTR [BX], 5`

Kod İzleme: byte ptr ve word ptr

```
▶ MOV BX, 0200h
2 MOV WORD PTR [BX], 1234h
3 MOV BYTE PTR [BX], 0FFh
4 MOV WORD PTR [BX], 0FFh
5 MOV AX, [BX]          ; sözcük
6 MOV CL, [BX+1]        ; bayt
```



BX = 0200h. Bellekte önceki değerler (AA BB CC DD) duruyor.

Adım 1 / 6

DEĞİŞKENLER

`BX = 0200h` `AX = 0000h` `CL = 00h` `w[0200] = BBAAh` `w[0202] = DDCCh`

Kod İzleme: byte ptr ve word ptr

```
1 MOV BX, 0200h
▶ MOV WORD PTR [BX], 1234h
3 MOV BYTE PTR [BX], 0FFh
4 MOV WORD PTR [BX], 0FFh
5 MOV AX, [BX]          ; sözcük
6 MOV CL, [BX+1]        ; bayt
```

BX	02	00
	BH	BL

AX	00	00
	AH	AL

CX	00	00
	CH	CL

DS:0200 ... DS:0203

0200	0201	0202	0203
34	12	CC	DD

word ptr: 2 bayt yazılır. Düşük bayt (34h) düşük adrese: **34 12**.

Adım 2 / 6

DEĞİŞKENLER

`BX = 0200h` `AX = 0000h` `CL = 00h` `w[0200] = 1234h` `w[0202] = DDCCh`

Kod İzleme: byte ptr ve word ptr

```
1 MOV BX, 0200h
2 MOV WORD PTR [BX], 1234h
▶ MOV BYTE PTR [BX], 0FFh
4 MOV WORD PTR [BX], 0FFh
5 MOV AX, [BX]          ; sözcük
6 MOV CL, [BX+1]        ; bayt
```

BX	02	00
	BH	BL

AX	00	00
	AH	AL

CX	00	00
	CH	CL

DS:0200 ... DS:0203

0200	0201	0202	0203
FF	12	CC	DD

byte ptr: yalnızca 0200h değişir → FF 12. 0201h'ye dokunulmaz.

Adım 3 / 6

DEĞİŞKENLER

`BX = 0200h` `AX = 0000h` `CL = 00h` `w[0200] = 12FFh` `w[0202] = DDCCh`

Kod İzleme: byte ptr ve word ptr

```
1 MOV BX, 0200h
2 MOV WORD PTR [BX], 1234h
3 MOV BYTE PTR [BX], 0FFh
▶ MOV WORD PTR [BX], 0FFh
5 MOV AX, [BX]          ; sözcük
6 MOV CL, [BX+1]        ; bayt
```

BX	02	00
	BH	BL

AX	00	00
	AH	AL

CX	00	00
	CH	CL

DS:0200 ... DS:0203

0200	0201	0202	0203
FF	00	CC	DD

Aynı sabit, **word ptr**: 00FFh yazılır → **FF 00**. 0201h de değişti!

Adım 4 / 6

DEĞİŞKENLER

`BX = 0200h` `AX = 0000h` `CL = 00h` `w[0200] = 00FFh` `w[0202] = DDCCh`

Kod İzleme: byte ptr ve word ptr

```
1 MOV BX, 0200h
2 MOV WORD PTR [BX], 1234h
3 MOV BYTE PTR [BX], 0FFh
4 MOV WORD PTR [BX], 0FFh
▶ MOV AX, [BX]          ; sözcük
6 MOV CL, [BX+1]        ; bayt
```

BX		AX		CX	
02	00	00	FF	00	00
BH	BL	AH	AL	CH	CL

DS:0200 ... DS:0203

0200	0201	0202	0203
FF	00	CC	DD

AX 16 bit olduğundan boyut belli: AX = **00FFh** (0201h yüksek bayt).

Adım 5 / 6

DEĞİŞKENLER

`BX = 0200h` `AX = 00FFh` `CL = 00h` `w[0200] = 00FFh` `w[0202] = DDCCh`

Kod İzleme: byte ptr ve word ptr

```
1 MOV BX, 0200h
2 MOV WORD PTR [BX], 1234h
3 MOV BYTE PTR [BX], 0FFh
4 MOV WORD PTR [BX], 0FFh
5 MOV AX, [BX]          ; sözcük
▶ MOV CL, [BX+1]        ; bayt
```

BX		AX		CX	
02	00	00	FF	00	00
BH	BL	AH	AL	CH	CL

DS:0200 ... DS:0203

0200	0201	0202	0203
FF	00	CC	DD

CL 8 bit: [BX+1] = 0201h'deki bayt → CL = **00h**.

Adım 6 / 6

DEĞİŞKENLER

`BX = 0200h` `AX = 00FFh` `CL = 00h` `w[0200] = 00FFh` `w[0202] = DDCCh`

VERİ TİPİ

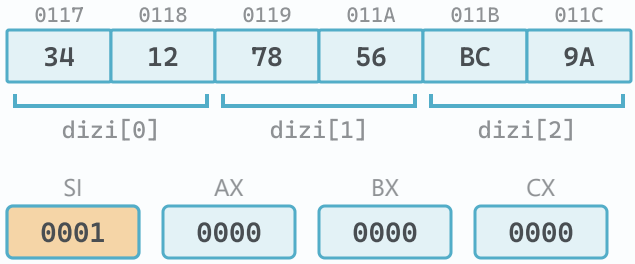
Little-Endian Düzen



- 8086 **little-endian**dir: 16 bitlik bir değerin **düşük baytı düşük adreste**, yüksek baytı bir sonraki adreste saklanır.
- `MOV [0200h], AX` → [0200h] = AL, [0201h] = AH. `MOV AX, [0200h]` tersini yapar.
- emu8086'nın bellek penceresinde sözcükler bu yüzden "ters" görünür: 1234h → `34 12`.
- Bu kural makine kodundaki sabitler için de geçerlidir: `MOV AX, 1234h` → `B8 34 12`.

Kod İzleme: Sözcük Dizisinde İndis Hatası

```
1  org 100h
▶  mov si, 1              ; indis 1
3  mov ax, dizi[si]       ; HATA!
4  shl si, 1              ; indis × 2
5  mov bx, dizi[si]       ; dizi[1]
6  mov si, 2              ; indis 2
7  shl si, 1              ; SI = 4
8  mov cx, dizi[si]       ; dizi[2]
9  ret
10 dizi dw 1234h, 5678h, 9ABCh
```



SI = 1: "ikinci eleman" düşüncesiyle indis 1 verildi. Ama SI bir **bayt** uzaklığıdır.

Adım 1 / 7

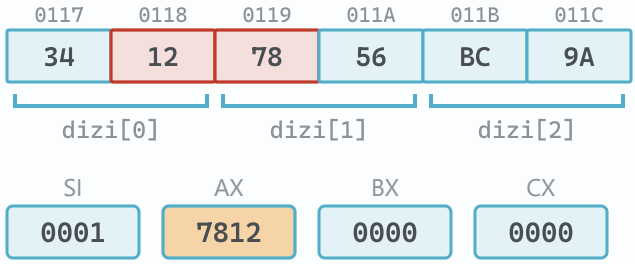
DEĞİŞKENLER

`SI = 0001h` `AX = 0000h` `BX = 0000h` `CX = 0000h`

Döngüde sözcük dizisi gezilirken SI her turda `ADD SI, 2` ile ilerletilir; `INC SI` yalnızca bayt dizileri içindir.

Kod İzleme: Sözcük Dizisinde İndis Hatası

```
1  org 100h
2  mov si, 1            ; indis 1
▶  mov ax, dizi[si]     ; HATA!
4  shl si, 1            ; indis × 2
5  mov bx, dizi[si]     ; dizi[1]
6  mov si, 2            ; indis 2
7  shl si, 1            ; SI = 4
8  mov cx, dizi[si]     ; dizi[2]
9  ret
10 dizi dw 1234h, 5678h, 9ABCh
```



`EA = 0117 + 0001 = 0118`

`dizi[si]` = 0118h: dizi[0]'ın yüksek baytı ile dizi[1]'in düşük baytı okunur → AX = **7812h**. Ne 1234h ne 5678h.

Adım 2 / 7

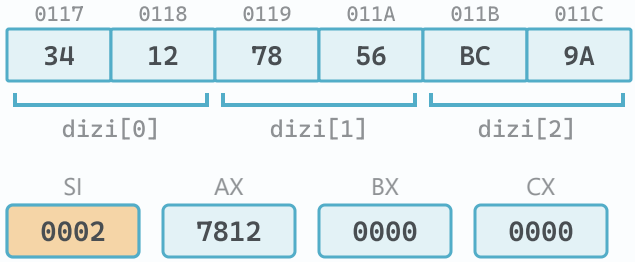
DEĞİŞKENLER

`SI = 0001h` `AX = 7812h` `BX = 0000h` `CX = 0000h`

Döngüde sözcük dizisi gezilirken SI her turda `ADD SI, 2` ile ilerletilir; `INC SI` yalnızca bayt dizileri içindir.

Kod İzleme: Sözcük Dizisinde İndis Hatası

```
1  org 100h
2  mov si, 1            ; indis 1
3  mov ax, dizi[si]     ; HATA!
▶  shl si, 1            ; indis × 2
5  mov bx, dizi[si]     ; dizi[1]
6  mov si, 2            ; indis 2
7  shl si, 1            ; SI = 4
8  mov cx, dizi[si]     ; dizi[2]
9  ret
10 dizi dw 1234h, 5678h, 9ABCh
```



`shl si, 1` : indis × 2 → SI = 2. Sözcük dizisinde bayt uzaklığı = indis × 2.

Adım 3 / 7

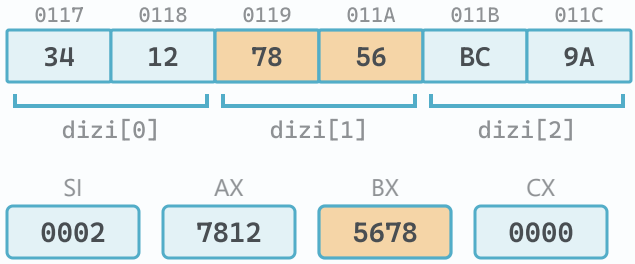
DEĞİŞKENLER

`SI = 0002h` `AX = 7812h` `BX = 0000h` `CX = 0000h`

Döngüde sözcük dizisi gezilirken SI her turda `ADD SI, 2` ile ilerletilir; `INC SI` yalnızca bayt dizileri içindir.

Kod İzleme: Sözcük Dizisinde İndis Hatası

```
1  org 100h
2  mov si, 1            ; indis 1
3  mov ax, dizi[si]     ; HATA!
4  shl si, 1            ; indis × 2
▶  mov bx, dizi[si]     ; dizi[1]
6  mov si, 2            ; indis 2
7  shl si, 1            ; SI = 4
8  mov cx, dizi[si]     ; dizi[2]
9  ret
10 dizi dw 1234h, 5678h, 9ABCh
```



`EA = 0117 + 0002 = 0119`

EA = 0119h → BX = **5678h** = dizi[1]. Doğru eleman.

Adım 4 / 7

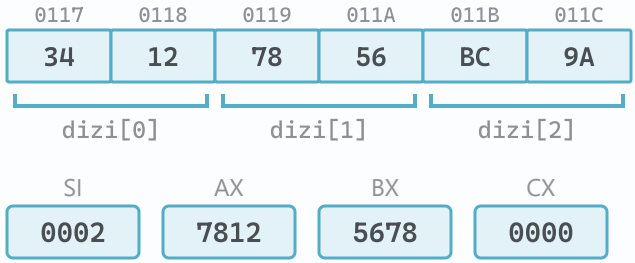
DEĞİŞKENLER

`SI = 0002h` `AX = 7812h` `BX = 5678h` `CX = 0000h`

Döngüde sözcük dizisi gezilirken SI her turda `ADD SI, 2` ile ilerletilir; `INC SI` yalnızca bayt dizileri içindir.

Kod İzleme: Sözcük Dizisinde İndis Hatası

```
1  org 100h
2  mov si, 1            ; indis 1
3  mov ax, dizi[si]     ; HATA!
4  shl si, 1            ; indis × 2
5  mov bx, dizi[si]     ; dizi[1]
▶  mov si, 2            ; indis 2
7  shl si, 1            ; SI = 4
8  mov cx, dizi[si]     ; dizi[2]
9  ret
10 dizi dw 1234h, 5678h, 9ABCh
```



SI = 2: bu kez indis 2 (üçüncü eleman).

Adım 5 / 7

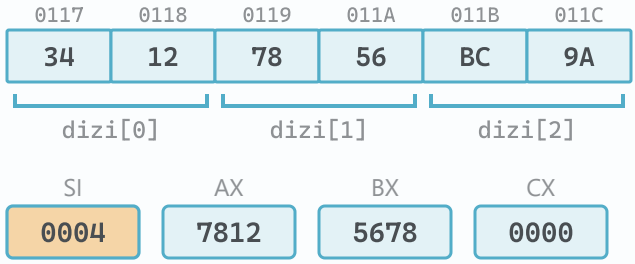
DEĞİŞKENLER

`SI = 0002h` `AX = 7812h` `BX = 5678h` `CX = 0000h`

Döngüde sözcük dizisi gezilirken SI her turda `ADD SI, 2` ile ilerletilir; `INC SI` yalnızca bayt dizileri içindir.

Kod İzleme: Sözcük Dizisinde İndis Hatası

```
1  org 100h
2  mov si, 1            ; indis 1
3  mov ax, dizi[si]     ; HATA!
4  shl si, 1            ; indis × 2
5  mov bx, dizi[si]     ; dizi[1]
6  mov si, 2            ; indis 2
▶  shl si, 1            ; SI = 4
8  mov cx, dizi[si]     ; dizi[2]
9  ret
10 dizi dw 1234h, 5678h, 9ABCh
```



`shl si, 1` : SI = **4**. Kaydırma, 2 ile çarpmanın en kısa yoludur.

Adım 6 / 7

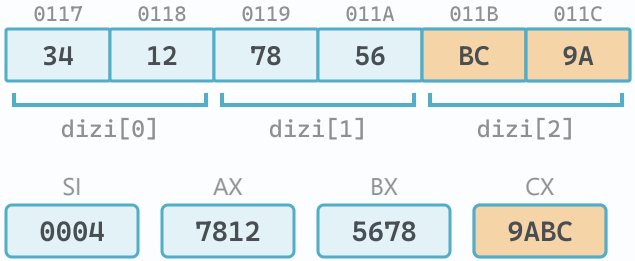
DEĞİŞKENLER

`SI = 0004h` `AX = 7812h` `BX = 5678h` `CX = 0000h`

Döngüde sözcük dizisi gezilirken SI her turda `ADD SI, 2` ile ilerletilir; `INC SI` yalnızca bayt dizileri içindir.

Kod İzleme: Sözcük Dizisinde İndis Hatası

```
1  org 100h
2  mov si, 1            ; indis 1
3  mov ax, dizi[si]     ; HATA!
4  shl si, 1            ; indis × 2
5  mov bx, dizi[si]     ; dizi[1]
6  mov si, 2            ; indis 2
7  shl si, 1            ; SI = 4
▶  mov cx, dizi[si]     ; dizi[2]
9  ret
10 dizi dw 1234h, 5678h, 9ABCh
```



`EA = 0117 + 0004 = 011B`

EA = 011Bh → CX = **9ABCh** = dizi[2].

Adım 7 / 7

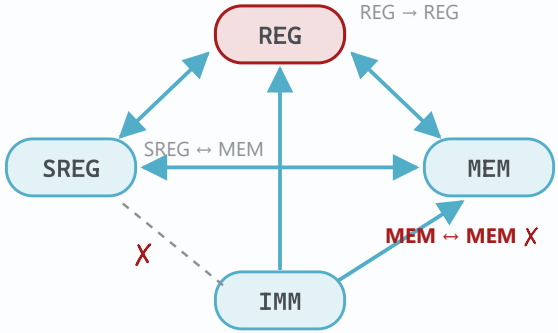
DEĞİŞKENLER

`SI = 0004h` `AX = 7812h` `BX = 5678h` `CX = 9ABCh`

Döngüde sözcük dizisi gezilirken SI her turda `ADD SI, 2` ile ilerletilir; `INC SI` yalnızca bayt dizileri içindir.

MOV Komutu

- `MOV hedef, kaynak` : kaynaktaki değeri hedefe **kopyalar**; kaynak değişmez.
- **Kaynak**: anlık değer (sabit), genel amaçlı yazmaç, kesim yazmacı ya da bellek konumu.
- **Hedef**: genel amaçlı yazmaç, kesim yazmacı (CS hariç) ya da bellek konumu.
- **CS ve IP** yazmaçlarına değer atamak için kullanılmaz.
- İki işlenenin **boyutu aynı** olmalıdır: ikisi de bayt ya da ikisi de sözcük.
- İki işlenen birden bellek olamaz.
- MOV **bayrakları etkilemez**.



MOV AX, BX

BX'teki değeri AX'e kopyalar. BX değişmez.

MOV [SI], 10h

SI'nın gösterdiği adrese (DS:SI) 10h değerini yazar. Boyut belirsiz olduğundan doğrusu `MOV BYTE PTR [SI], 10h`.

Desteklenen İşlenenler

Genel amaçlı

```
MOV REG, memory
MOV memory, REG
MOV REG, REG
MOV memory, immediate
MOV REG, immediate
```

Kesim yazmaçları

```
MOV SREG, memory
MOV memory, SREG
MOV REG, SREG
MOV SREG, REG
```

REG	AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP
SREG	DS, ES, SS ve CS (CS yalnızca **ikinci** işlenen, yani kaynak olarak)
memory	`[BX]`, `[BX+SI+7]`, değişken...
immediate	`5`, `-24`, `3Fh`, `10001101b`...

`MOV SREG, immediate` ve `MOV SREG, SREG` yoktur; değer önce bir genel yazmaca alınır.

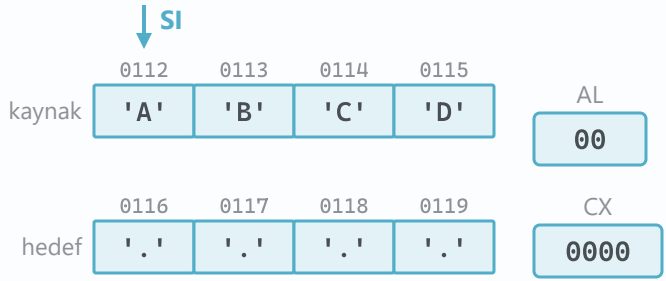
MOV: Geçerli mi?

Komut	?	Komut	?
`MOV [BX], [SI]`	1	`MOV AX, CS`	6
`MOV DS, ES`	2	`MOV SP, BX`	7
`MOV DS, 100h`	3	`MOV BL, DX`	8
`MOV CS, AX`	4	`MOV AL, [SI+BP]`	9
`MOV [BX], 5`	5	`MOV 5, AX`	10

▶ **Cevaplar**

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
▶ 2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
10 loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



SI = **0112h**: kaynağın bağıl konumu.

Adım 1 / 12

DEĞİŞKENLER

`SI = 0112h` `DI = 0000h` `AL = 00h` `CX = 0000h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB`: DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
▶  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
10 loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



DI = **0116h**: hedefin bağıl konumu. İki işaretçi iki ayrı tampon gösterir.

Adım 2 / 12

DEĞİŞKENLER

`SI = 0112h` `DI = 0116h` `AL = 00h` `CX = 0000h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB`: DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
▶ 4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
10 loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



CX = 4: kopyalanacak bayt sayısı (LOOP sayacı).

Adım 3 / 12

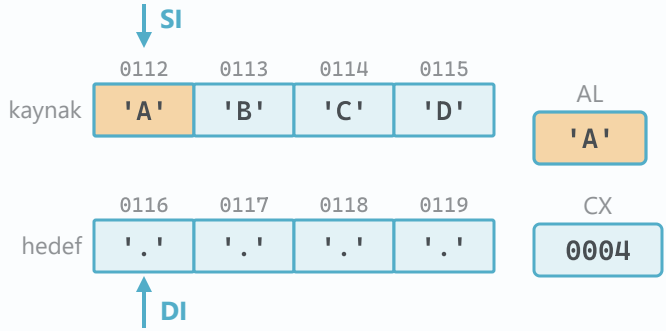
DEĞİŞKENLER

`SI = 0112h` `DI = 0116h` `AL = 00h` `CX = 0004h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB`: DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
▶  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
10 loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



`mov al, [si]` : bellekten yazmaca → AL = 'A' (41h). Bellekten belleğe MOV olmadığı için ara durak AL.

Adım 4 / 12

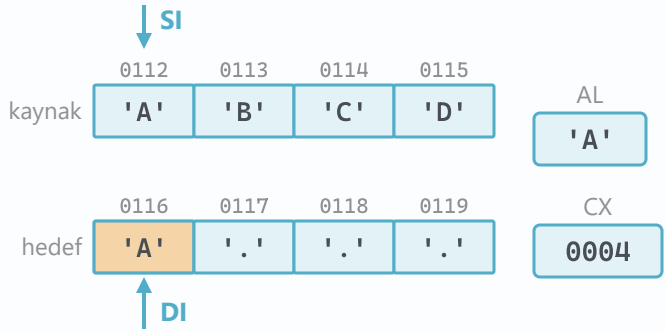
DEĞİŞKENLER

`SI = 0112h` `DI = 0116h` `AL = 41h` `CX = 0004h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB` : DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
▶  mov [di], al          ; AL → bellek
8  inc si
9  inc di
10 loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



`mov [di], al` : yazmaçtan belleğe → [0116] = 41h.

Adım 5 / 12

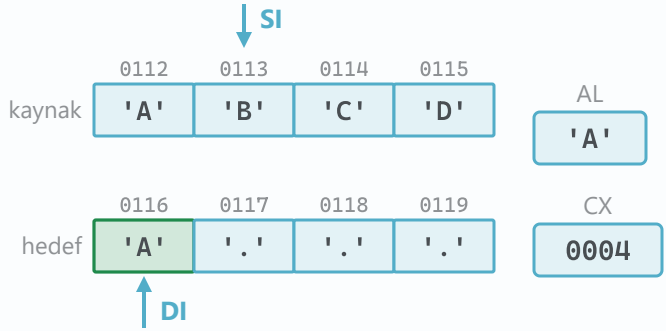
DEĞİŞKENLER

`SI = 0112h` `DI = 0116h` `AL = 41h` `CX = 0004h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB` : DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
▶  inc si
9  inc di
10 loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



`inc si` : kaynak işaretçisi sonraki bayta.

Adım 6 / 12

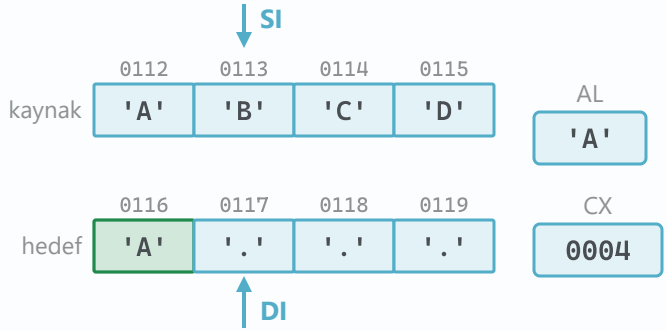
DEĞİŞKENLER

`SI = 0113h` `DI = 0116h` `AL = 41h` `CX = 0004h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB` : DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
▶  inc di
10 loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



`inc di` : hedef işaretçisi de sonraki bayta. İki işaretçi birlikte ilerler.

Adım 7 / 12

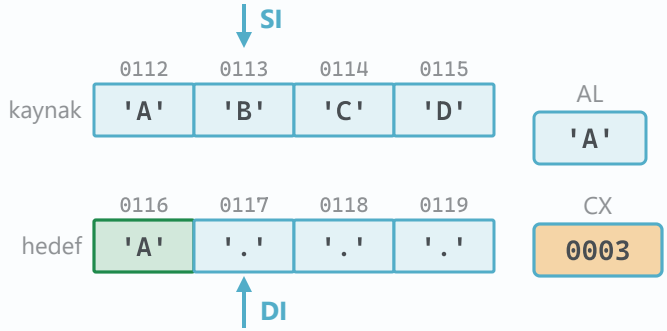
DEĞİŞKENLER

`SI = 0113h` `DI = 0117h` `AL = 41h` `CX = 0004h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB` : DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
▶  loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



`loop kopya` : CX = 3 ≠ 0 → başa dön.

Adım 8 / 12

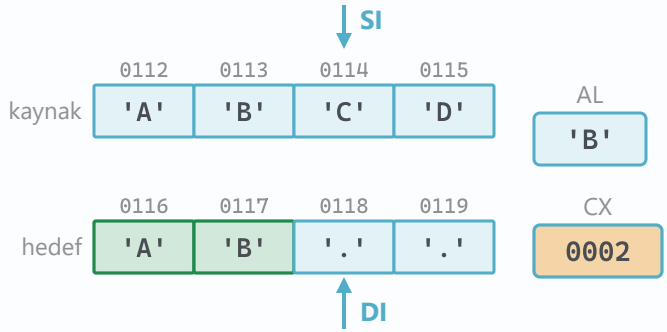
DEĞİŞKENLER

`SI = 0113h` `DI = 0117h` `AL = 41h` `CX = 0003h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB` : DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
▶ loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



Tur 2 (özet): 'B' kopyalandı, SI = 0114h, DI = 0118h, CX = 2.

Adım 9 / 12

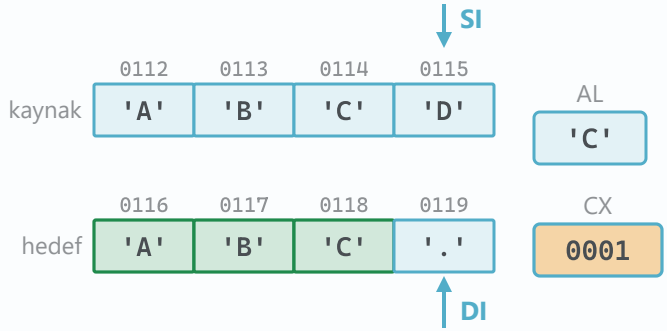
DEĞİŞKENLER

`SI = 0114h` `DI = 0118h` `AL = 42h` `CX = 0002h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB`: DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
▶  loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



Tur 3 (özet): 'C' kopyalandı, CX = 1.

Adım 10 / 12

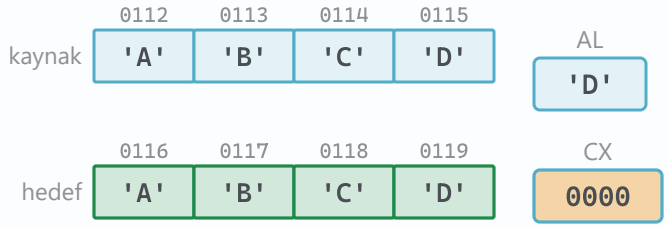
DEĞİŞKENLER

`SI = 0115h` `DI = 0119h` `AL = 43h` `CX = 0001h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB`: DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
▶  loop kopya
11 ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



Tur 4 (özet): 'D' kopyalandı, CX = **0** → LOOP düşer. Hedef artık **ABCD**.

Adım 11 / 12

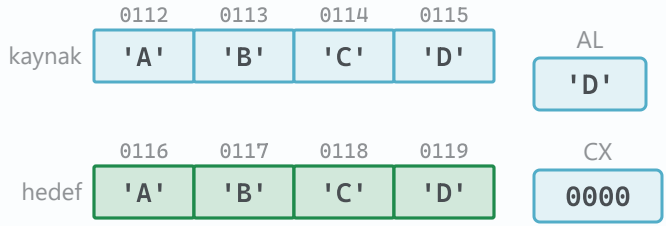
DEĞİŞKENLER

`SI = 0116h` `DI = 011Ah` `AL = 44h` `CX = 0000h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB`: DS:SI → ES:DI) tek komutta yapılır.

Kod İzleme: Yazmaç Üzerinden Blok Kopyalama

```
1  org 100h
2  mov si, offset kaynak
3  mov di, offset hedef
4  mov cx, 4
5  kopya:
6  mov al, [si]          ; bellek → AL
7  mov [di], al          ; AL → bellek
8  inc si
9  inc di
10 loop kopya
▶  ret
12 kaynak db 'ABCD'
13 hedef  db 4 dup('.')
```



`ret` : SI ve DI tamponların hemen arkasını gösteriyor (0116h, 011Ah).

Adım 12 / 12

DEĞİŞKENLER

`SI = 0116h` `DI = 011Ah` `AL = 44h` `CX = 0000h`

Her bayt için iki bellek erişimi ve bir yazmaç gerekir. Aynı iş dizgi komutlarıyla (`REP MOVSB` : DS:SI → ES:DI) tek komutta yapılır.

PROGRAM ÖRNEĞİ

Program Örneği

```
ORG 100h            ; .com programı için
MOV AX, 0B800h      ; AX = B800h
MOV DS, AX          ; DS = B800h
MOV CL, 'A'         ; CL = 41h
MOV CH, 11011111b   ; CH = DFh
MOV BX, 15Eh        ; BX = 015Eh
MOV [BX], CX        ; B800:015E ← CX
RET                 ; DOS'a dön
```

- Program, **video belleği** üzerinde belirli bir konuma 'A' harfini yazar.
- `MOV`, kaynak ve hedef arasında değer kopyalar.
- `ORG`, programın başlangıç adresini belirler (makine koduna çevrilmez).
- `;` açıklama satırları için kullanılır.
- DS'ye sabit doğrudan atanamadığı için B800h önce AX'e yüklenir.

Komut	Makine kodu	Komut	Makine kodu
`MOV AX, 0B800h`	`B8 00 B8`	`MOV CH, 11011111b`	`B5 DF`
`MOV DS, AX`	`8E D8`	`MOV BX, 15Eh`	`BB 5E 01`
`MOV CL, 'A'`	`B1 41`	`MOV [BX], CX` · `RET`	`89 0F` · `C3`

PROGRAM ÖRNEĞİ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Ekrana 'A' Yazmak

```
▶ ORG 100h
2 MOV AX, 0B800h
3 MOV DS, AX
4 MOV CL, 'A'
5 MOV CH, 11011111b
6 MOV BX, 15Eh
7 MOV [BX], CX
8 RET
```



Ekran boş: her hücre **20h** (boşluk) + **07h** (siyah zemin üzerine gri yazı).

Adım 1 / 8

DEĞİŞKENLER

`AX = 0000h` `DS = 0700h` `CH = 00h` `CL = 00h` `BX = 0000h`

`[B800:015E] = 20h` `[B800:015F] = 07h`

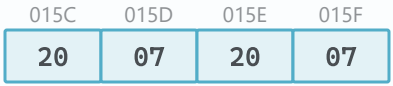
PROGRAM ÖRNEĞİ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Ekrana 'A' Yazmak

```
1  ORG 100h
▶  MOV AX, 0B800h
3  MOV DS, AX
4  MOV CL, 'A'
5  MOV CH, 11011111b
6  MOV BX, 15Eh
7  MOV [BX], CX
8  RET
```



B800:015C ... 015F (video belleği)



AX = **B800h** (renkli metin ekranının kesimi).

Adım 2 / 8

DEĞİŞKENLER

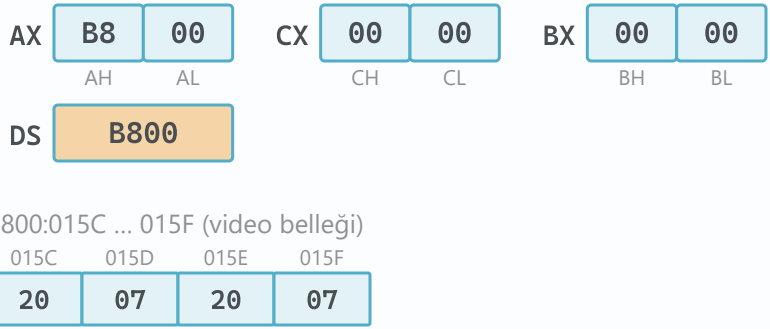
`AX = B800h` `DS = 0700h` `CH = 00h` `CL = 00h` `BX = 0000h`

`[B800:015E] = 20h` `[B800:015F] = 07h`

PROGRAM ÖRNEĞİ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Ekrana 'A' Yazmak

```
1 ORG 100h
2 MOV AX, 0B800h
▶ MOV DS, AX
4 MOV CL, 'A'
5 MOV CH, 11011111b
6 MOV BX, 15Eh
7 MOV [BX], CX
8 RET
```



DS = AX = **B800h**. Artık [ ] erişimleri video belleğine gider.

Adım 3 / 8

DEĞİŞKENLER

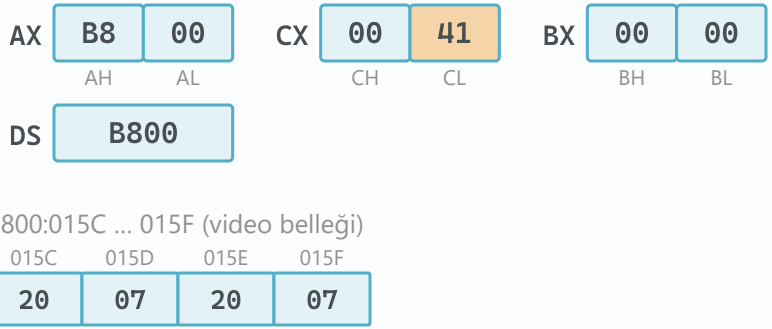
`AX = B800h` `DS = B800h` `CH = 00h` `CL = 00h` `BX = 0000h`

`[B800:015E] = 20h` `[B800:015F] = 07h`

PROGRAM ÖRNEĞİ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Ekrana 'A' Yazmak

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV DS, AX
▶ MOV CL, 'A'
5 MOV CH, 11011111b
6 MOV BX, 15Eh
7 MOV [BX], CX
8 RET
```



CL = 'A' = **41h** (ASCII kodu).

Adım 4 / 8

DEĞİŞKENLER

`AX = B800h` `DS = B800h` `CH = 00h` `CL = 41h` `BX = 0000h`

`[B800:015E] = 20h` `[B800:015F] = 07h`

PROGRAM ÖRNEĞİ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Ekrana 'A' Yazmak

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV DS, AX
4 MOV CL, 'A'
▶ MOV CH, 11011111b
6 MOV BX, 15Eh
7 MOV [BX], CX
8 RET
```

	AH	AL
AX	B8	00

	CH	CL
CX	DF	41

	BH	BL
BX	00	00

DS
B800

B800:015C ... 015F (video belleği)

015C	015D	015E	015F
20	07	20	07

CH = 11011111b = **DFh**: renk (öznitelik) baytı.

Adım 5 / 8

DEĞİŞKENLER

AX = **B800h** | DS = **B800h** | CH = **DFh** | CL = **41h** | BX = **0000h** | [B800:015E] = **20h** | [B800:015F] = **07h**

PROGRAM ÖRNEĞİ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Ekrana 'A' Yazmak

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV DS, AX
4 MOV CL, 'A'
5 MOV CH, 11011111b
▶ MOV BX, 15Eh
7 MOV [BX], CX
8 RET
```

AX	B8	00	CX	DF	41	BX	01	5E
	AH	AL		CH	CL		BH	BL

DS = B800

B800:015C ... 015F (video belleği)

015C	015D	015E	015F
20	07	20	07

BX = **015Eh** = 350: ekrandaki konumun bağıl adresi.

Adım 6 / 8

DEĞİŞKENLER

AX = **B800h** · DS = **B800h** · CH = **DFh** · CL = **41h** · BX = **015Eh**

[B800:015E] = **20h** · [B800:015F] = **07h**

PROGRAM ÖRNEĞİ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Ekrana 'A' Yazmak

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV DS, AX
4 MOV CL, 'A'
5 MOV CH, 11011111b
6 MOV BX, 15Eh
▶ MOV [BX], CX
8 RET
```

AX	B8	00	CX	DF	41	BX	01	5E
	AH	AL		CH	CL		BH	BL

DS	B800

B800:015C ... 015F (video belleği)

015C	015D	015E	015F
20	07	41	DF

CX = DF41h sözcük olarak yazılır: [015Eh] = **41h** ('A'), [015Fh] = **DFh** (renk). Fiziksel: B8000h + 15Eh = **B815Eh**.

Adım 7 / 8

DEĞİŞKENLER

AX = **B800h** DS = **B800h** CH = **DFh** CL = **41h** BX = **015Eh**

[B800:015E] = **41h** [B800:015F] = **DFh**

PROGRAM ÖRNEĞİ · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Ekrana 'A' Yazmak

```
1 ORG 100h
2 MOV AX, 0B800h
3 MOV DS, AX
4 MOV CL, 'A'
5 MOV CH, 11011111b
6 MOV BX, 15Eh
7 MOV [BX], CX
▶ RET
```

AX	B8	00	CX	DF	41	BX	01	5E
	AH	AL		CH	CL		BH	BL

DS = B800

B800:015C ... 015F (video belleği)

015C	015D	015E	015F
20	07	41	DF

RET: DOS'a dönülür; 'A' harfi ekranda kalır.

Adım 8 / 8

DEĞİŞKENLER

AX = **B800h** · DS = **B800h** · CH = **DFh** · CL = **41h** · BX = **015Eh**

[B800:015E] = **41h** · [B800:015F] = **DFh**

PROGRAM ÖRNEĞİ

Metin Modu Video Belleği



Konum

80×25 metin ekranında her hücre **2 bayttır**: karakter + renk. Bağıl adres = (satır × 80 + sütun) × 2 → (2 × 80 + 15) × 2 = 350 = **15Eh**.

Renk baytı DFh = 1101 1111b

Bit 7: yanıp sönme (emu8086'da parlak zemin) · bit 6–4 = 101: macenta zemin · bit 3–0 = 1111: parlak beyaz yazı.

Bellekten Bir Değeri Yazmaca Taşıma

```
1 ORG 100h
2 MOV AX, [1000h]   ; A1 00 10
3 RET
```



DS = 0700h. DS:1000h'de **CD AB** baytları var. IP = 0100h.

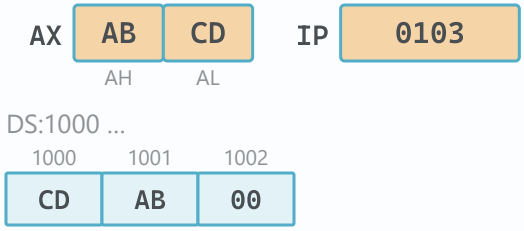
Adım 1 / 3

DEĞİŞKENLER

`AX = 0000h` `IP = 0100h` `w[1000h] = ABCDh`

Bellekten Bir Değeri Yazmaca Taşıma

```
1 ORG 100h
▶ MOV AX, [1000h]   ; A1 00 10
3 RET
```



`MOV AX,[1000h]`: iki bayt okunur, little-endian → AX = **ABCDh**. Komut 3 bayt → IP = 0103h.

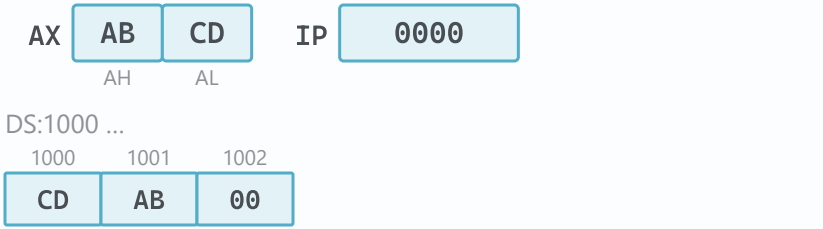
Adım 2 / 3

DEĞİŞKENLER

`AX = ABCDh` `IP = 0103h` `w[1000h] = ABCDh`

Bellekten Bir Değeri Yazmaca Taşıma

```
1 ORG 100h
2 MOV AX, [1000h]   ; A1 00 10
▶ RET
```



RET: IP = 0000h (PSP'deki INT 20h). Bellek değişmedi; MOV yalnızca **kopyalar**. Fiziksel adres: 07000h + 1000h = 08000h.

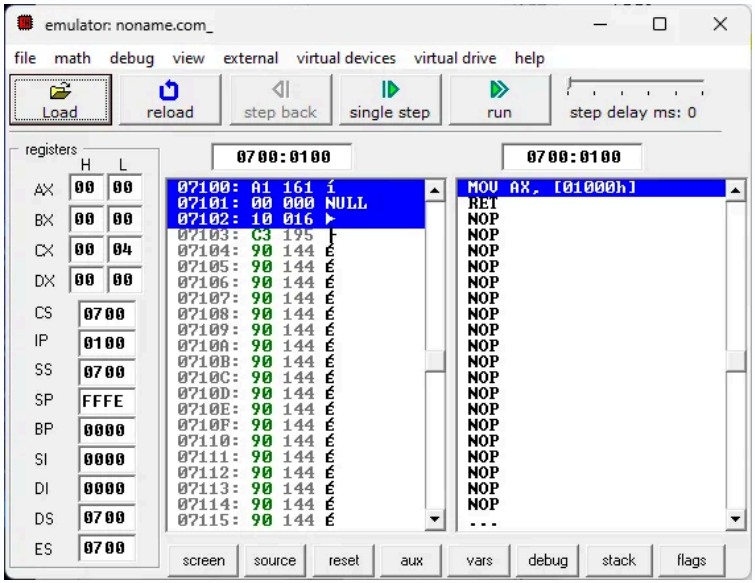
Adım 3 / 3

DEĞİŞKENLER

`AX = ABCDh` `IP = 0000h` `w[1000h] = ABCDh`

emu8086'da Görünüm

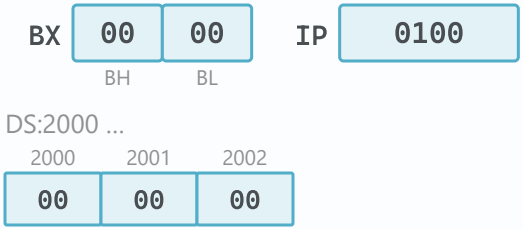
- Sol: yazmaçlar (H/L yarıları ayrı). `CS = DS = 0700h`, `IP = 0100h`, `SP = FFFEh`.
- Orta: bellek (fiziksel adres `07100h` = 0700:0100). `A1 00 10` komutun 3 baytıdır.
- Sağ: ayrıştırılmış (disassembled) kod: `MOV AX, [01000h]`, ardından `RET` (C3).
- **single step** ile komut yürütülür; değişen yazmaçlar renklenir.
- Bellek boş olduğundan bu ekranda AX = 0000h kalır.



emu8086 öykünücü penceresi

Yazmaçtaki Değeri Belleğe Taşıma

```
1 ORG 100h
2 MOV BX, 1234h      ; BB 34 12
3 MOV [2000h], BX    ; 89 1E 00 20
4 RET
```



DS:2000h boş (00 00).

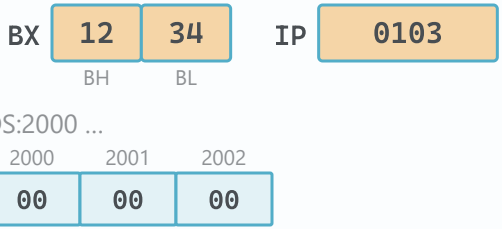
Adım 1 / 4

DEĞİŞKENLER

`BX = 0000h` `IP = 0100h` `w[2000h] = 0000h`

Yazmaçtaki Değeri Belleğe Taşıma

```
1 ORG 100h
▶ MOV BX, 1234h      ; BB 34 12
3 MOV [2000h], BX    ; 89 1E 00 20
4 RET
```



BX = **1234h**; IP = 0103h.

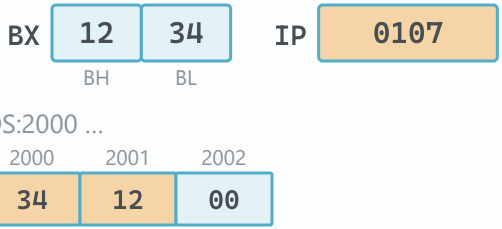
Adım 2 / 4

DEĞİŞKENLER

BX = 1234h IP = 0103h w[2000h] = 0000h

Yazmaçtaki Değeri Belleğe Taşıma

```
1 ORG 100h
2 MOV BX, 1234h      ; BB 34 12
▶ MOV [2000h], BX    ; 89 1E 00 20
4 RET
```



BX belleğe yazılır: [2000h] = **34h**, [2001h] = **12h**. Komut 4 bayt → IP = 0107h.

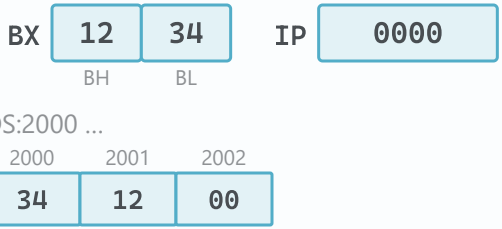
Adım 3 / 4

DEĞİŞKENLER

`BX = 1234h` `IP = 0107h` `w[2000h] = 1234h`

Yazmaçtaki Değeri Belleğe Taşıma

```
1 ORG 100h
2 MOV BX, 1234h      ; BB 34 12
3 MOV [2000h], BX    ; 89 1E 00 20
▶ RET
```



RET: IP = 0000h. BX değişmedi. Fiziksel adres: 07000h + 2000h = **09000h**.

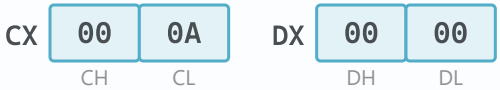
Adım 4 / 4

DEĞİŞKENLER

`BX = 1234h` `IP = 0000h` `w[2000h] = 1234h`

İki Yazmaç Değerini Takas

```
▶ MOV CX, 10   ; CX = 000Ah
2 MOV DX, 20   ; DX = 0014h
3 XCHG CX, DX  ; takas
```



CX = 10 = **000Ah** (sayılar varsayılan olarak onluktur).

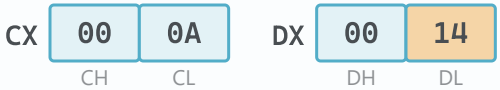
Adım 1 / 3

DEĞİŞKENLER

`CX = 000Ah` `DX = 0000h`

İki Yazmaç Değerini Takas

```
1 MOV CX, 10   ; CX = 000Ah
▶ MOV DX, 20   ; DX = 0014h
3 XCHG CX, DX  ; takas
```



DX = 20 = **0014h**.

Adım 2 / 3

DEĞİŞKENLER

CX = 000Ah DX = 0014h

İki Yazmaç Değerini Takas

```
1 MOV CX, 10   ; CX = 000Ah
2 MOV DX, 20   ; DX = 0014h
▶ XCHG CX, DX  ; takas
```



`XCHG` iki işleneni **tek komutta** değiştirir: CX = 0014h, DX = 000Ah. Bayraklar etkilenmez.

Adım 3 / 3

DEĞİŞKENLER

`CX = 0014h` `DX = 000Ah`

XCHG Kuralları

XCHG hedef, kaynak

- Yazmaç ↔ yazmaç: `XCHG AX, BX`
- Yazmaç ↔ bellek: `XCHG AL, [SI]`
- Bellek ↔ bellek **yok**.
- Kesim yazmaçları ve sabitler kullanılamaz.
- Boyutlar aynı olmalı; bayraklar değişmez.
- `XCHG AX, reg` tek baytlık komuttur (`XCHG AX, AX` = `NOP` = 90h).

XCHG olmasaydı takas için **geçici** bir yazmaç gerekirdi:

```
MOV AX, CX     ; AX ← CX (yedek)
MOV CX, DX     ; CX ← DX
MOV DX, AX     ; DX ← eski CX
```

3 komut ve bir yazmaç daha harcanır; XCHG bunu tek komutta yapar.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
▶  MOV SI, OFFSET sayi
3  MOV AL, 5Ah
4  XCHG AL, [SI]      ; AL ⇔ [SI]
5  MOV CX, 1111h
6  MOV DX, 2222h
7  MOV AX, CX         ; yedek
8  MOV CX, DX
9  MOV DX, AX         ; takas
10 XCHG CX, DX        ; tek komut
11 RET
12 sayi DB 0A5h
```

AX	CX	DX	sayi · 0116h
0000h	0000h	0000h	A5h

SI = 0116h: `sayi` 'nın adresi; bellekte sayi = **A5h**.

Adım 1 / 9

DEĞİŞKENLER

`SI = 0116h` `AX = 0000h` `CX = 0000h` `DX = 0000h` `sayi = A5h`

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
2  MOV SI, OFFSET sayi
▶  MOV AL, 5Ah
4  XCHG AL, [SI]       ; AL ⇔ [SI]
5  MOV CX, 1111h
6  MOV DX, 2222h
7  MOV AX, CX          ; yedek
8  MOV CX, DX
9  MOV DX, AX          ; takas
10 XCHG CX, DX         ; tek komut
11 RET
12 sayi DB 0A5h
```

AX	CX	DX	sayi · 0116h
005Ah	0000h	0000h	A5h

AL = **5Ah**.

Adım 2 / 9

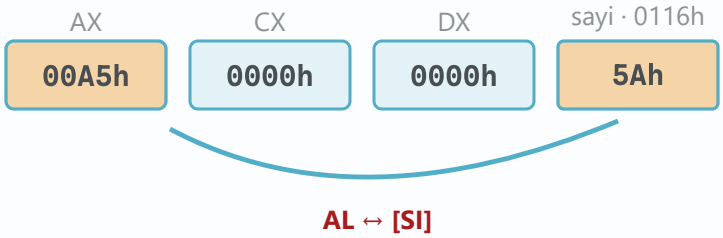
DEĞİŞKENLER

SI = **0116h** · AX = **005Ah** · CX = **0000h** · DX = **0000h** · sayi = **A5h**

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
2  MOV SI, OFFSET sayi
3  MOV AL, 5Ah
▶ XCHG AL, [SI]       ; AL ⇔ [SI]
5  MOV CX, 1111h
6  MOV DX, 2222h
7  MOV AX, CX         ; yedek
8  MOV CX, DX
9  MOV DX, AX         ; takas
10 XCHG CX, DX        ; tek komut
11 RET
12 sayi DB 0A5h
```



`XCHG AL, [SI]` : AL ↔ bellek. AL = **A5h**, sayi = **5Ah**. Ara yazmaç gerekmedi.

Adım 3 / 9

DEĞİŞKENLER

`SI = 0116h` `AX = 00A5h` `CX = 0000h` `DX = 0000h` `sayi = 5Ah`

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
2  MOV SI, OFFSET sayi
3  MOV AL, 5Ah
4  XCHG AL, [SI]       ; AL ⇔ [SI]
▶  MOV CX, 1111h
6  MOV DX, 2222h
7  MOV AX, CX          ; yedek
8  MOV CX, DX
9  MOV DX, AX          ; takas
10 XCHG CX, DX         ; tek komut
11 RET
12 sayi DB 0A5h
```

AX	CX	DX	sayi · 0116h
00A5h	1111h	0000h	5Ah

CX = 1111h.

Adım 4 / 9

DEĞİŞKENLER

SI = 0116h | AX = 00A5h | CX = 1111h | DX = 0000h | sayi = 5Ah

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
2  MOV SI, OFFSET sayi
3  MOV AL, 5Ah
4  XCHG AL, [SI]       ; AL ⇔ [SI]
5  MOV CX, 1111h
▶  MOV DX, 2222h
7  MOV AX, CX          ; yedek
8  MOV CX, DX
9  MOV DX, AX          ; takas
10 XCHG CX, DX         ; tek komut
11 RET
12 sayi DB 0A5h
```

AX	CX	DX	sayi · 0116h
00A5h	1111h	2222h	5Ah

DX = 2222h. Şimdi aynı takas XCHG olmadan yapılacak.

Adım 5 / 9

DEĞİŞKENLER

SI = 0116h · AX = 00A5h · CX = 1111h · DX = 2222h · sayi = 5Ah

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
2  MOV SI, OFFSET sayi
3  MOV AL, 5Ah
4  XCHG AL, [SI]       ; AL ⇔ [SI]
5  MOV CX, 1111h
6  MOV DX, 2222h
▶  MOV AX, CX          ; yedek
8  MOV CX, DX
9  MOV DX, AX          ; takas
10 XCHG CX, DX         ; tek komut
11 RET
12 sayi DB 0A5h
```

AX	CX	DX	sayi · 0116h
1111h	1111h	2222h	5Ah

1. AX ← CX

3 komut + yardımcı yazmaç

`MOV AX, CX` : CX'in yedeği AX'te (AX'in eski değeri kayboldu).

Adım 6 / 9

DEĞİŞKENLER

SI = 0116h | AX = 1111h | CX = 1111h | DX = 2222h | sayi = 5Ah

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
2  MOV SI, OFFSET sayi
3  MOV AL, 5Ah
4  XCHG AL, [SI]       ; AL ⇔ [SI]
5  MOV CX, 1111h
6  MOV DX, 2222h
7  MOV AX, CX          ; yedek
▶  MOV CX, DX
9  MOV DX, AX          ; takas
10 XCHG CX, DX         ; tek komut
11 RET
12 sayi DB 0A5h
```

AX	CX	DX	sayi · 0116h
1111h	2222h	2222h	5Ah

2. CX ← DX

3 komut + yardımcı yazmaç

`MOV CX, DX` : CX = 2222h. CX'in eski değeri yalnızca AX'te duruyor.

Adım 7 / 9

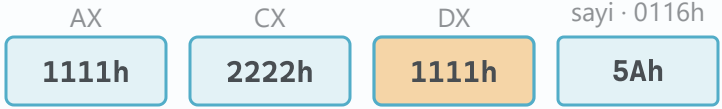
DEĞİŞKENLER

SI = 0116h · AX = 1111h · CX = 2222h · DX = 2222h · sayi = 5Ah

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
2  MOV SI, OFFSET sayi
3  MOV AL, 5Ah
4  XCHG AL, [SI]       ; AL ⇔ [SI]
5  MOV CX, 1111h
6  MOV DX, 2222h
7  MOV AX, CX          ; yedek
8  MOV CX, DX
▶  MOV DX, AX          ; takas
10 XCHG CX, DX         ; tek komut
11 RET
12 sayi DB 0A5h
```



3. DX ← AX

3 komut + yardımcı yazmaç

`MOV DX, AX` : DX = 1111h. Takas 3 komut ve bir yazmaç harcadı.

Adım 8 / 9

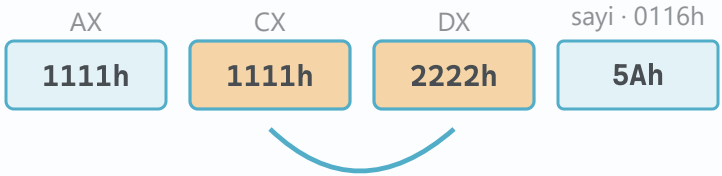
DEĞİŞKENLER

`SI = 0116h` `AX = 1111h` `CX = 2222h` `DX = 1111h` `sayi = 5Ah`

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Kod İzleme: XCHG ile Takas

```
1  ORG 100h
2  MOV SI, OFFSET sayi
3  MOV AL, 5Ah
4  XCHG AL, [SI]      ; AL ⇔ [SI]
5  MOV CX, 1111h
6  MOV DX, 2222h
7  MOV AX, CX         ; yedek
8  MOV CX, DX
9  MOV DX, AX         ; takas
▶  XCHG CX, DX        ; tek komut
11 RET
12 sayi DB 0A5h
```



`XCHG CX, DX`: tek komutta geri takas: CX = **1111h**, DX = **2222h**; AX dokunulmadan kaldı.

Adım 9 / 9

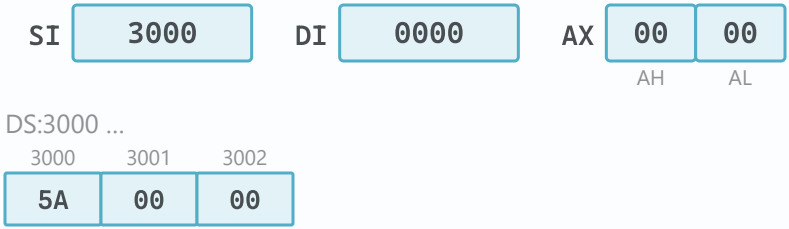
DEĞİŞKENLER

`SI = 0116h` `AX = 1111h` `CX = 1111h` `DX = 2222h` `sayi = 5Ah`

XCHG bayrakları değiştirmez; bellek ↔ bellek ve kesim yazmacı ile takas yoktur.

Bir Yazmaca Başka Yazmacın İçeriğini Kopyalama

```
▶ MOV SI, 3000h  ; SI = adres
2 MOV DI, SI     ; DI = SI
3 MOV AL, [SI]   ; DS:3000h
4 MOV [DI+1], AL ; DS:3001h
```



SI'ya bir **bellek adresi** (bağıl konum) atanır: 3000h.

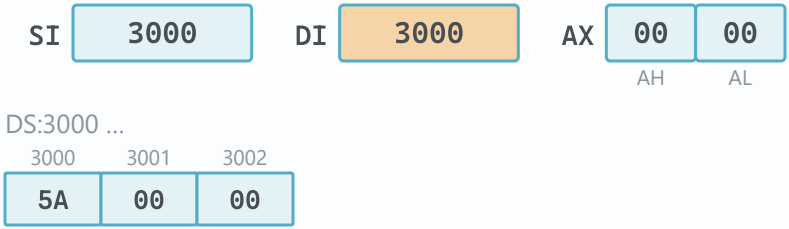
Adım 1 / 4

DEĞİŞKENLER

`SI = 3000h` `DI = 0000h` `AL = 00h`

Bir Yazmaca Başka Yazmacın İçeriğini Kopyalama

```
1 MOV SI, 3000h  ; SI = adres
▶ MOV DI, SI     ; DI = SI
3 MOV AL, [SI]   ; DS:3000h
4 MOV [DI+1], AL ; DS:3001h
```



DI = SI = **3000h**. Kopyalanan şey adrestir, bellekteki veri değil.

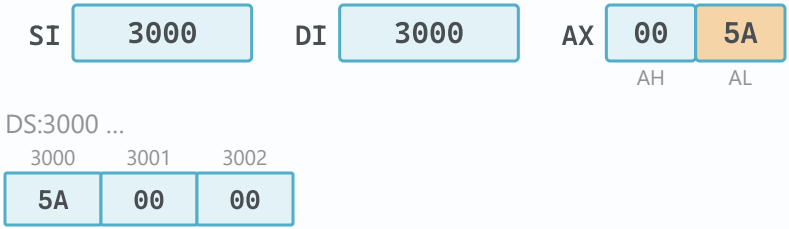
Adım 2 / 4

DEĞİŞKENLER

SI = 3000h | DI = 3000h | AL = 00h

Bir Yazmaca Başka Yazmacın İçeriğini Kopyalama

```
1 MOV SI, 3000h  ; SI = adres
2 MOV DI, SI     ; DI = SI
▶ MOV AL, [SI]   ; DS:3000h
4 MOV [DI+1], AL ; DS:3001h
```



Artık SI ve DI aynı hücreyi gösterir: AL = [3000h] = **5Ah**.

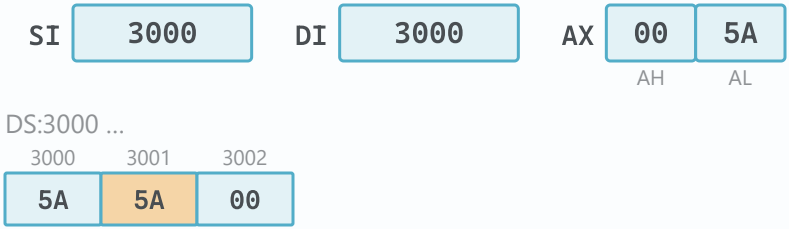
Adım 3 / 4

DEĞİŞKENLER

`SI = 3000h` `DI = 3000h` `AL = 5Ah`

Bir Yazmaca Başka Yazmacın İçeriğini Kopyalama

```
1 MOV SI, 3000h  ; SI = adres
2 MOV DI, SI     ; DI = SI
3 MOV AL, [SI]   ; DS:3000h
▶ MOV [DI+1], AL ; DS:3001h
```



[DI+1] = 3001h'ye AL yazılır → **5A 5A**.

Adım 4 / 4

DEĞİŞKENLER

SI = **3000h** DI = **3000h** AL = **5Ah**

Bir Bellek Adresine Sabit Bir Değeri Yazma

```
1 ORG 100h
2 MOV BYTE PTR [4000h], 65  ; 'A'
3 MOV WORD PTR [4002h], 65
4 RET
```

IP 0100

DS:4000 ...

4000	4001	4002	4003
00	99	00	99

DS:4000h ... 4003h: 00 99 00 99.

Adım 1 / 4

DEĞİŞKENLER

IP = 0100h [4000h] = 00h [4003h] = 99h

Bir Bellek Adresine Sabit Bir Değeri Yazma

```
1 ORG 100h
▶ MOV BYTE PTR [4000h], 65  ; 'A'
3 MOV WORD PTR [4002h], 65
4 RET
```

IP 0105

DS:4000 ...

4000	4001	4002	4003
41	99	00	99

65 = 41h = 'A' tek bayt olarak yazılır. Makine kodu **C6 06 00 40 41** (5 bayt) → IP = 0105h. [4001h] değişmez.

Adım 2 / 4

DEĞİŞKENLER

IP = 0105h [4000h] = 41h [4003h] = 99h

Bir Bellek Adresine Sabit Bir Değeri Yazma

```
1 ORG 100h
2 MOV BYTE PTR [4000h], 65  ; 'A'
▶ MOV WORD PTR [4002h], 65
4 RET
```

IP **010B**

DS:4000 …

4000	4001	4002	4003
41	99	41	00

WORD PTR: 0041h yazılır → [4002h] = 41h, [4003h] = **00h** (99h ezildi). Komut 6 bayt → IP = 010Bh.

Adım 3 / 4

DEĞİŞKENLER

`IP = 010Bh` `[4000h] = 41h` `[4003h] = 00h`

Bir Bellek Adresine Sabit Bir Değeri Yazma

```
1 ORG 100h
2 MOV BYTE PTR [4000h], 65  ; 'A'
3 MOV WORD PTR [4002h], 65
▶ RET
```

IP 0000

DS:4000 ...

4000	4001	4002	4003
41	99	41	00

RET: IP = 0000h. Sabitin boyutu yoktur; boyutu **PTR** öneki belirler.

Adım 4 / 4

DEĞİŞKENLER

IP = 0000h [4000h] = 41h [4003h] = 00h

Bir Yazmacın Düşük Baytını Diğerine Kopyalama

```
▶ MOV AX, 1234h   ; AH = 12h, AL = 34h
2 MOV BX, 0FFFFh
3 MOV BL, AL      ; yalnız BL
```



AX = **1234h**: AH = 12h, AL = 34h.

Adım 1 / 3

DEĞİŞKENLER

`AX = 1234h` `BX = 0000h` `BL = 00h`

Bir Yazmacın Düşük Baytını Diğerine Kopyalama

```
1 MOV AX, 1234h    ; AH = 12h, AL = 34h
▶ MOV BX, 0FFFFh
3 MOV BL, AL       ; yalnız BL
```



Etkiyi görmek için BX = FFFFh.

Adım 2 / 3

DEĞİŞKENLER

AX = **1234h** BX = **FFFFh** BL = **FFh**

Bir Yazmacın Düşük Baytını Diğerine Kopyalama

```
1 MOV AX, 1234h    ; AH = 12h, AL = 34h
2 MOV BX, 0FFFFh
▶ MOV BL, AL       ; yalnız BL
```



AL (AX'in **düşük baytı**) BL'ye kopyalanır: BL = 34h. BH değişmez → BX = **FF34h**.

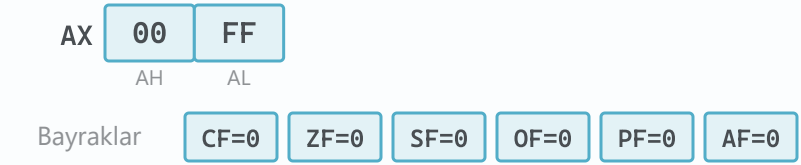
Adım 3 / 3

DEĞİŞKENLER

AX = **1234h** BX = **FF34h** BL = **34h**

Bir Yazmacın Yüksek Baytına Ekleme

```
▶ MOV AX, 00FFh   ; AH = 00h, AL = FFh
2 ADD AH, 01h     ; yalnız AH
3 ADD AL, 01h     ; yalnız AL
4 MOV AX, 01FFh
5 ADD AX, 1       ; 16 bit
```



AX = **00FFh**.

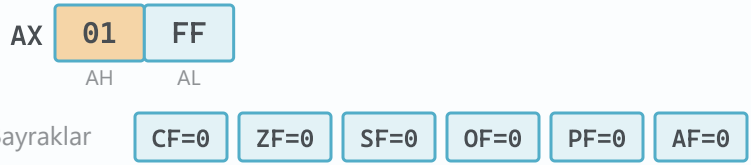
Adım 1 / 5

DEĞİŞKENLER

AX = **00FFh** | AH = **00h** | AL = **FFh** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **0** | AF = **0**

Bir Yazmacın Yüksek Baytına Ekleme

```
1 MOV AX, 00FFh    ; AH = 00h, AL =
  FFh
▶ ADD AH, 01h      ; yalnız AH
3 ADD AL, 01h      ; yalnız AL
4 MOV AX, 01FFh
5 ADD AX, 1        ; 16 bit
```



AH = 00h + 01h = 01h → AX = **01FFh**. Sonuç sıfır değil, elde yok.

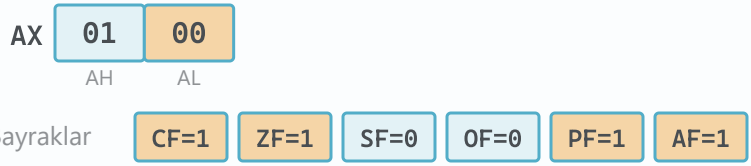
Adım 2 / 5

DEĞİŞKENLER

AX = **01FFh** | AH = **01h** | AL = **FFh** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **0** | AF = **0**

Bir Yazmacın Yüksek Baytına Ekleme

```
1 MOV AX, 00FFh    ; AH = 00h, AL = FFh
2 ADD AH, 01h      ; yalnız AH
▶ ADD AL, 01h      ; yalnız AL
4 MOV AX, 01FFh
5 ADD AX, 1        ; 16 bit
```



AL = FFh + 1 = 00h, **CF = 1**, ZF = 1. Elde AH'ye **geçmez**: AX = **0100h**.

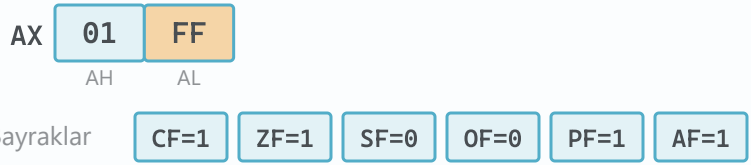
Adım 3 / 5

DEĞİŞKENLER

AX = **0100h** | AH = **01h** | AL = **00h** | CF = **1** | ZF = **1** | SF = **0** | OF = **0** | PF = **1** | AF = **1**

Bir Yazmacın Yüksek Baytına Ekleme

```
1 MOV AX, 00FFh    ; AH = 00h, AL = FFh
2 ADD AH, 01h      ; yalnız AH
3 ADD AL, 01h      ; yalnız AL
▶ MOV AX, 01FFh
5 ADD AX, 1        ; 16 bit
```



Karşılaştırma için AX yeniden 01FFh. MOV bayrakları değiştirmez.

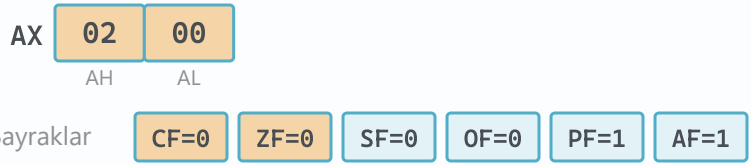
Adım 4 / 5

DEĞİŞKENLER

AX = 01FFh | AH = 01h | AL = FFh | CF = 1 | ZF = 1 | SF = 0 | OF = 0 | PF = 1 | AF = 1

Bir Yazmacın Yüksek Baytına Ekleme

```
1 MOV AX, 00FFh    ; AH = 00h, AL = FFh
2 ADD AH, 01h      ; yalnız AH
3 ADD AL, 01h      ; yalnız AL
4 MOV AX, 01FFh
▶ ADD AX, 1        ; 16 bit
```



16 bitlik toplama: 01FFh + 1 = **0200h**; elde AL'den AH'ye geçti, CF = 0.

Adım 5 / 5

DEĞİŞKENLER

AX = 0200h | AH = 02h | AL = 00h | CF = 0 | ZF = 0 | SF = 0 | OF = 0 | PF = 1 | AF = 1

İkili, Sekizlik ve Onaltılık Değer Atama

```
1 ▶ mov al, 00000101b ; ikili
2   mov bl, 0ah       ; onaltılık
3   mov cl, 10o       ; sekizli
4   add al, bl        ; 5 + 10 = 15
5   sub al, cl        ; 15 - 8 = 7
```

AX	00	05		BX	00	00		CX	00	00	
	AH	AL			BH	BL			CH	CL	

Bayraklar: CF=0 | ZF=0 | SF=0 | OF=0 | PF=0 | AF=0

00000101b = **5** → AL = 05h.

Adım 1 / 5

DEĞİŞKENLER

AL = **05h** | BL = **00h** | CL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **0** | AF = **0**

İkili, Sekizlik ve Onaltılık Değer Atama

```
1 mov al, 00000101b ; ikili
▶ mov bl, 0ah       ; onaltılık
3 mov cl, 10o       ; sekizli
4 add al, bl        ; 5 + 10 = 15
5 sub al, cl        ; 15 - 8 = 7
```

AX	00	05	BX	00	0A	CX	00	00
	AH	AL		BH	BL		CH	CL

Bayraklar: CF=0 ZF=0 SF=0 OF=0 PF=0 AF=0

0Ah = **10** → BL = 0Ah. Harfle başladığı için başta 0 var.

Adım 2 / 5

DEĞİŞKENLER

AL = 05h | BL = 0Ah | CL = 00h | CF = 0 | ZF = 0 | SF = 0 | OF = 0 | PF = 0 | AF = 0

İkili, Sekizlik ve Onaltılık Değer Atama

```
1 mov al, 00000101b ; ikili
2 mov bl, 0ah       ; onaltılık
▶ mov cl, 10o       ; sekizli
4 add al, bl        ; 5 + 10 = 15
5 sub al, cl        ; 15 - 8 = 7
```

	AH	AL	BH	BL	CH	CL
AX	00	05				
BX			00	0A		
CX					00	08

Bayraklar: CF=0 ZF=0 SF=0 OF=0 PF=0 AF=0

10o (sekizli) = 1×8 + 0 = **8** → CL = 08h.

Adım 3 / 5

DEĞİŞKENLER

AL = 05h | BL = 0Ah | CL = 08h | CF = 0 | ZF = 0 | SF = 0 | OF = 0 | PF = 0 | AF = 0

İkili, Sekizlik ve Onaltılık Değer Atama

```
1 mov al, 00000101b ; ikili
2 mov bl, 0ah       ; onaltılık
3 mov cl, 10o       ; sekizli
▶ add al, bl        ; 5 + 10 = 15
5 sub al, cl        ; 15 - 8 = 7
```

AX	00	0F	BX	00	0A	CX	00	08
	AH	AL		BH	BL		CH	CL

Bayraklar: CF=0 | ZF=0 | SF=0 | OF=0 | PF=1 | AF=0

AL = 5 + 10 = 15 = **0Fh**. PF = 1 (0000 1111: dört adet 1).

Adım 4 / 5

DEĞİŞKENLER

AL = 0Fh | BL = 0Ah | CL = 08h | CF = 0 | ZF = 0 | SF = 0 | OF = 0 | PF = 1 | AF = 0

İkili, Sekizlik ve Onaltılık Değer Atama

```
1 mov al, 00000101b ; ikili
2 mov bl, 0ah       ; onaltılık
3 mov cl, 10o       ; sekizli
4 add al, bl        ; 5 + 10 = 15
▶ sub al, cl        ; 15 - 8 = 7
```



AL = 15 − 8 = 7 = **07h**. Ödünç yok → CF = 0; PF = 0 (üç adet 1).

Adım 5 / 5

DEĞİŞKENLER

AL = 07h | BL = 0Ah | CL = 08h | CF = 0 | ZF = 0 | SF = 0 | OF = 0 | PF = 0 | AF = 0

MOV ÖRNEKLERİ

Sayı Yazımı

Taban	Sonek	Örnek	Onluk değer
Onluk (decimal)	yok ya da `d`	`25` , `25d`	25
İkili (binary)	`b`	`00011001b`	25
Sekizli (octal)	`o` ya da `q`	`31o`	25
Onaltılık (hexadecimal)	`h`	`19h` , `0FFh`	25, 255
Karakter	`' '`	`'A'`	65 (41h)

Onaltılık sayı bir harfle (A–F) başlıyorsa başına **0** eklenir: `0Ah` , `0B800h` . Aksi hâlde `Ah` , `AH` yazmacıyla; `B800h` bir etiketle karışır. emu8086 ayrıca `0x1000` biçimini de kabul eder.

MOV ÖRNEKLERİ

Bellek Erişiminde Sık Yapılan Hatalar

Hatalı	Ne olur?	Doğrusu
`MOV SI, dizi`	SI'ya dizinin adresi değil ilk elemanın değeri yüklenir.	`MOV SI, OFFSET dizi` ya da `LEA SI, dizi`
`INC SI` (sözcük dizisi)	Sözcüğün ortasından okunur: iki elemanın yarıları birleşir.	`ADD SI, 2`
`MOV AL, [BP]` ile değişken	SS kesiminden okunur; .EXE'de yanlış bayt gelir.	`[BX]`, `[SI]` ya da `DS:[BP]`
`MOV [BX], 5`	Boyut belirsiz; derleyici hata verir ya da tahmin eder.	`MOV BYTE PTR [BX], 5`
`MOV AX, B800h`	B800h bir etiket adı sanılır.	`MOV AX, 0B800h`
DS'yi değiştirip bırakmak	Sonraki tüm değişken erişimleri başka kesime gider.	`PUSH DS` … `POP DS` ya da ES
`MOV AX, [0200h]` → 3412h beklemek	Little-endian unutuldu: bellekte **34 12** ise AX = 1234h.	Düşük adres = düşük bayt

Bu hataların çoğu derleme hatası vermez; program çalışır ama yanlış adresten okur. emu8086'da bellek penceresinde EA ve fiziksel adres kontrol edilerek bulunur.

Laboratuvar: Bellek Erişimi

Adresleme modları

KOD · **28 bayt**

```
 1 ORG 100h
 2 MOV BX, OFFSET tablo
 3 MOV SI, 2
 4 MOV AH, 7           ; anlık
 5 MOV AL, [tablo+1]   ; doğru…
 6 MOV CL, [BX]        ; dolay…
 7 MOV CH, [BX+4]      ; taban…
 8 MOV DL, [BX+SI]     ; taban…
 9 MOV DH, [BX+SI+3]   ; + d8
10 RET
11 tablo DB 11h,22h,33h,44h,5…
```

YAZMAÇLAR

AX	0000	AH	00	AL	00
BX	0000	BH	00	BL	00
CX	001C	CH	00	CL	1C
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**

```
0110 8A 10 8A 70 03 C3 11 22
0118 33 44 55 66 00 00 00 00
0120 00 00 00 00 00 00 00 00
0128 00 00 00 00 00 00 00 00
0130 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: 28 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (001Ch) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

Adresleme
[taban + indis + yer değiştirme]; taban BX/BP, indis SI/DI, yer değiştirme d8/d16 — her biri isteğe bağlı.

Kesim seçimi
BP varsa SS, yoksa DS; önekle değiştirilebilir (ES:[BX]).

Fiziksel adres
kesim × 10h + EA; 1234:7890 → 19BD0h.

Boyut
byte ptr / word ptr (b. / w.); yazmaç işlenen boyutu belirler. Little-endian.

MOV
Kopyalar, bayrakları etkilemez; bellek↔bellek, kesim↔sabit, CS/IP hedef yok.

XCHG
İki işleneni tek komutta takas eder; kesim ve sabit yok.

ÖZET

Alıştırmalar

1. DS = 3000h, BX = 0010h, SI = 0005h: `[BX+SI+20h]` için EA ve fiziksel adres?
2. SS = 4000h, BP = FFFEh: `[BP+4]` için EA ve fiziksel adres?
3. `[BX+BP]`, `[DI+SI]`, `[BX+DI+300]`, `[DX]` modlarından hangileri geçerli?
4. [0500h] = 78h, [0501h] = 56h ise `MOV AX,[0500h]` sonrası AX? `MOV AL,[0501h]` sonrası AL?
5. Ekranın satır 10, sütun 40 konumuna yeşil zemin üzerine sarı 'X' yazan kodu yazın (renk 2Eh).
6. DS'yi değiştirmeden ES ile video belleğine yazmak için önceki programı nasıl değiştirirsiniz?

▶ **Cevaplar**

Kaynaklar

- emu8086 belgeleri, *8086 assembler tutorial for beginners*, bölüm 2: memory access, MOV instruction.
- Intel, *8086/8088 User's Manual*, adresleme modları ve komut kodlaması.
- Barry B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, bölüm 3 (Addressing Modes).
- IBM PC metin modu video belleği (B800h) ve öznitelik baytı belgeleri.

Son

Bölüm 4.1: Bellek Erişimi

Sonraki bölüm: Bölüm 4.2: Değişkenler · Terim sözlüğü