Bölüm 11: Komut Kodlama ve Dizgi Komutları

Mikroişlemciler

Komut sınıfları · Makine kodu · ModR/M · MOV biçimleri · XLAT · LEA · LDS · LES · Dizgi komutları · REP önekleri · JMP kodlaması · İşlemci kontrolü · Bayrak etkileri

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Komut sınıfları
8086'nın yaklaşık 100 komutunu altı işlevsel sınıfa ayırmak ve her sınıfın tipik kullanımını açıklamak.

Makine kodu
İşlem kodu, D/W bitleri ve ModR/M baytından bir komutun makine kodunu çıkarmak.

Dizgi komutları
MOVS, CMPS, SCAS, LODS ve STOS komutlarını REP önekleriyle kullanmak; DF, SI, DI ve CX değişimini izlemek.

Başvuru
XLAT, LEA, LDS, LES ve işlemci kontrolü komutlarını tanımak; bayrak etkileri tablosundan bir komutun hangi bayrağı değiştirdiğini okumak.

GİRİŞ

8086 Komut Kümesine Genel Bakış

1. Veri taşıma

Yazmaç, bellek, yığın ve G/Ç arasında veri kopyalama.

`MOV PUSH POP XCHG IN OUT XLAT LEA LDS LES LAHF SAHF PUSHF POPF`

2. Aritmetik

Dört işlem, BCD düzeltme, işaret genişletme.

`ADD ADC INC AAA DAA SUB SBB DEC NEG CMP AAS DAS MUL IMUL AAM DIV IDIV AAD CBW CWD`

3. Mantık, kaydırma, döndürme

Bit düzeyinde işlemler.

`NOT AND OR XOR TEST SHL SAL SHR SAR ROL ROR RCL RCR`

4. Dizgi işleme

Bellek bloklarını SI/DI ile tarama ve kopyalama.

`MOVSB/W CMPSB/W SCASB/W LODSB/W STOSB/W REP REPE REPZ REPNE REPNZ`

5. Kontrol aktarma

Dallanma, döngü, prosedür ve kesme.

`JMP Jcc (JE JNE JA JB JG JL …) LOOP LOOPE LOOPNE JCXZ CALL RET RETF INT INTO IRET`

6. İşlemci kontrolü

Bayrakları kurma/temizleme ve işlemci durumu.

`CLC STC CMC CLD STD CLI STI HLT WAIT ESC LOCK NOP`

KOMUT KODLAMA

Makine Kodu Biçimi



Komut uzunluğu 1–6 bayt; önekler (segment, REP, LOCK) en başa eklenebilir.

D (direction)

1: REG alanı **hedef**; 0: REG alanı **kaynak**.

W (word)

0: bayt (8 bit) işlem; 1: sözcük (16 bit) işlem.

ModR/M

MOD işlenenin yazmaç mı bellek mi olduğunu, R/M adres biçimini, REG diğer yazmacı seçer.

Sık kullanılan komutların daha kısa özel biçimleri vardır: `inc ax` = **40h** (1 bayt), `mov ax, imm16` = **B8h** + 2 bayt, `add al, imm8` = **04h** + 1 bayt.

REG ve MOD Alanları

REG	W = 0	W = 1	SREG
`000`	AL	AX	ES (00)
`001`	CL	CX	CS (01)
`010`	DL	DX	SS (10)
`011`	BL	BX	DS (11)
`100`	AH	SP	
`101`	CH	BP	
`110`	DH	SI	
`111`	BH	DI	

MOD	Anlamı
`00`	Bellek, yer değiştirme yok (R/M = 110 ise **doğrudan** 16 bit adres)
`01`	Bellek + 8 bit yer değiştirme (işaret genişletilir)
`10`	Bellek + 16 bit yer değiştirme
`11`	Yazmaç: R/M alanı da bir REG kodudur

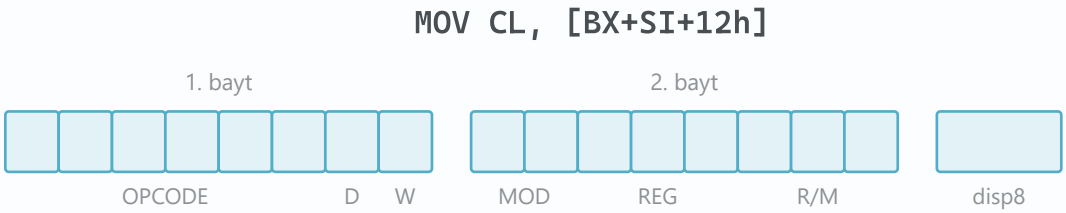
Örnek: `mov ax, bx` → **8B C3**: 100010 **1 1** | **11** 000 011 (D = 1, hedef REG = AX, R/M = BX). Aynı komut **89 D8** olarak da kodlanabilir (D = 0).

R/M Alanı: Etkin Adres (EA)

R/M	MOD = 00	MOD = 01	MOD = 10	Varsayılan kesim
`000`	[BX+SI]	[BX+SI+d8]	[BX+SI+d16]	DS
`001`	[BX+DI]	[BX+DI+d8]	[BX+DI+d16]	DS
`010`	[BP+SI]	[BP+SI+d8]	[BP+SI+d16]	**SS**
`011`	[BP+DI]	[BP+DI+d8]	[BP+DI+d16]	**SS**
`100`	[SI]	[SI+d8]	[SI+d16]	DS
`101`	[DI]	[DI+d8]	[DI+d16]	DS
`110`	**[d16] doğrudan**	[BP+d8]	[BP+d16]	DS / **SS**
`111`	[BX]	[BX+d8]	[BX+d16]	DS

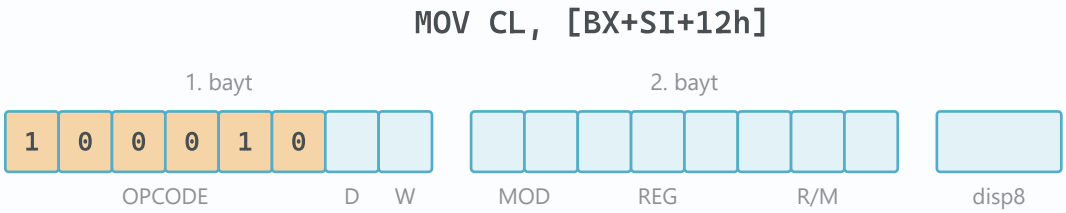
- Yalnızca **BX, BP, SI, DI** adres yazmacı olabilir; `[ax]`, `[cx]`, `[si+di]` geçersizdir.
- `[bp]` yazılınca assembler onu `[bp+0]` (MOD = 01, d8 = 0) olarak kodlar; çünkü MOD = 00, R/M = 110 doğrudan adrese ayrılmıştır.
- Kesim öneki varsayılanı değiştirir: `mov al, es:[bx]` → önek **26h**.

Kodlama Adım Adım: `MOV CL, [BX+SI+12h]`



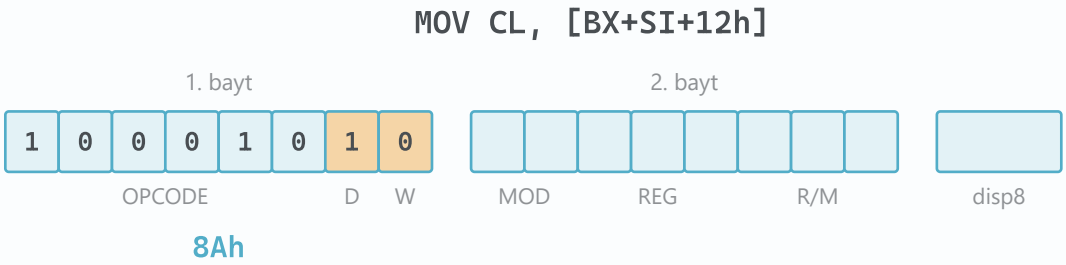
Komut: bellekteki bir baytı CL'ye taşı. Biçim: **MOV reg, r/m** (işlem kodu + ModR/M + yer değiştirme).

Kodlama Adım Adım: `MOV CL, [BX+SI+12h]`



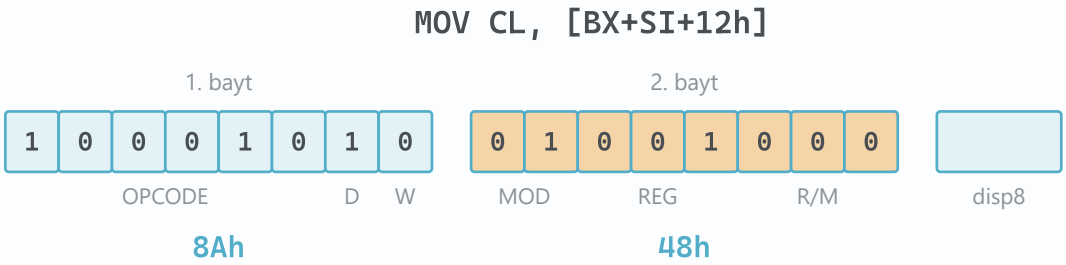
MOV r/m ↔ reg ailesinin işlem kodu **100010**.

Kodlama Adım Adım: `MOV CL, [BX+SI+12h]`



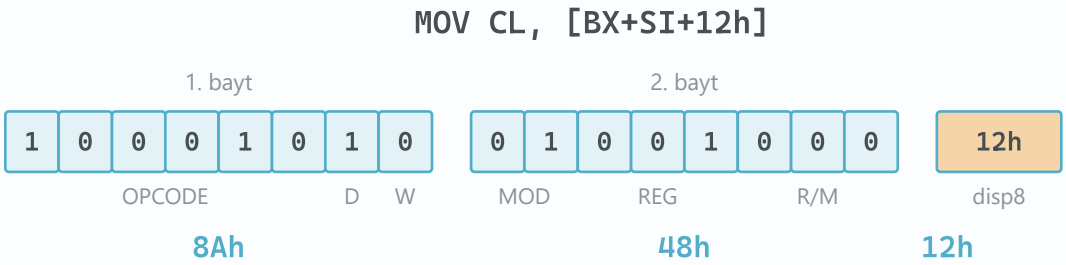
D = 1: REG alanı **hedef**tir. **W = 0**: bayt işlem. 1. bayt = 1000 1010b = **8Ah**.

Kodlama Adım Adım: `MOV CL, [BX+SI+12h]`



MOD = 01: bellek + 8 bit yer değiştirme. **REG = 001** (CL). **R/M = 000** ([BX+SI]). 2. bayt = 01 001 000b = **48h**.

Kodlama Adım Adım: `MOV CL, [BX+SI+12h]`



Makine kodu: 8A 48 12 (3 bayt)

12h, 8 bitlik yer değiştirme olarak eklenir. Sonuç: **8A 48 12**. Yer değiştirme 16 bit olsaydı MOD = 10 ve 4 bayt olurdu.

KOMUT KODLAMA

Örnek Komutlar ve Uzunlukları

Komut	Makine kodu	Bayt	Not
`nop`	`90`	1	aslında `xchg ax, ax`
`inc ax` / `push bx`	`40` / `53`	1	yazmaç kodu işlem kodunun içinde (40h+r, 50h+r)
`mov al, 5`	`B0 05`	2	B0h+r, anlık bayt
`int 21h`	`CD 21`	2	kesme numarası komutta
`mov ax, 1234h`	`B8 34 12`	3	düşük bayt önce (little-endian)
`mov cl, [bx+si+12h]`	`8A 48 12`	3	ModR/M + d8
`mov es:[bx], al`	`26 88 07`	3	26h = ES öneki
`add [bx+di+1000h], dx`	`01 91 00 10`	4	MOD = 10, d16
`mov word ptr [1234h], 5678h`	`C7 06 34 12 78 56`	6	en uzun biçim: d16 + imm16
`rep movsb`	`F3 A4`	2	F3h = REP öneki

Önekler: 26h ES, 2Eh CS, 36h SS, 3Eh DS (segment); F3h REP/REPE, F2h REPNE; F0h LOCK. emu8086'da makine kodunu görmek için derledikten sonra listing (.lst) dosyasına bakılabilir.

KOMUT KODLAMA

MOV'un Kodlama Biçimleri

Biçim	İşlem kodu (8 / 16 bit)	Ardından	Örnek	Kod
yazmaç → r/m	`88` / `89`	ModR/M (+ disp)	`mov [bx], al`	`88 07`
r/m → yazmaç	`8A` / `8B`	ModR/M (+ disp)	`mov cx, [si]`	`8B 0C`
anlık → r/m	`C6` / `C7`	ModR/M (REG = 000) + disp + imm	`mov byte ptr [bx], 5`	`C6 07 05`
anlık → yazmaç	`B0+r` / `B8+r`	imm8 / imm16	`mov si, 1000h`	`BE 00 10`
bellek → AL / AX	`A0` / `A1`	16 bit adres	`mov ax, [2000h]`	`A1 00 20`
AL / AX → bellek	`A2` / `A3`	16 bit adres	`mov [1000h], al`	`A2 00 10`
kesim yazmacı → r/m16	`8C`	ModR/M (REG = sreg)	`mov ax, es`	`8C C0`
r/m16 → kesim yazmacı	`8E`	ModR/M (REG = sreg)	`mov ds, ax`	`8E D8`

Aynı komutun birden çok kodu olabilir: `mov al, [1000h]` = **A0 00 10** (3 bayt) ya da **8A 06 00 10** (4 bayt). Assembler kısa biçimi seçer.

MOV'un Kodlama Biçimleri (devam)

Kesim yazmacı kodları (sreg)

ES = **00**, CS = **01**, SS = **10**, DS = **11**; ModR/M'nin REG alanına 0 + sreg olarak yazılır. `mov ds, ax` : **11 011 000** = D8h.

Olmayan biçimler

Anlık → kesim yazmacı yok (`mov ds, 1234h` geçersiz), bellek → bellek yok. `mov cs, …` kullanılmaz; CS yalnız JMP/CALL/RET/INT ile değişir.

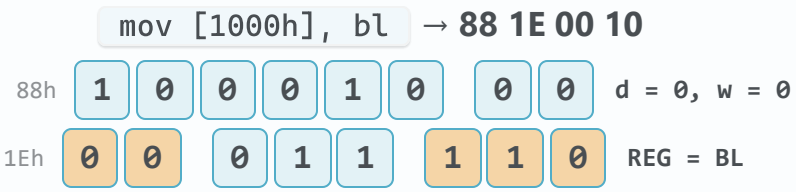
`mov ds, ax → 8E D8`



ES 00 · CS 01 · SS 10 · DS 11 (REG alanına 0 + sreg)

KOMUT KODLAMA

MOD = 00, R/M = 110: [BP] Değil, [d16]



MOD = 00 ve R/M = 110 → ardından **2 bayt adres** gelir: 00 10 = 1000h. Komut **4 bayt**tır. [BP] için assembler MOD = 01, d8 = 0 kullanır.

Komut	Kod	Not
`mov [1000h], al`	`A2 00 10`	AL için kısa biçim, 3 bayt
`mov [1000h], al`	`88 06 00 10`	genel biçim, 4 bayt (aynı iş): ModR/M 06h = 00 000 110
`mov [1000h], bl`	`88 1E 00 10`	BL için kısa biçim yok
`mov [bp], bl`	`88 5E 00`	5Eh = 01 011 110: [BP+d8], d8 = 0; kesim SS

Bir çözücü 1Eh'yi yanlışlıkla [BP] sayarsa `88 1E`'yi 2 baytlık `mov [bp], bl` sanar; adres baytları yeni bir komut olur: `00 10 = add [bx+si], dl`. Sonraki tüm sınırlar kayar ve bellek bozulur.

KOMUT KODLAMA

Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme

konum 0'dan



konum 1'den

Bellekte 11 bayt. Bunlar kod mu, veri mi? İşlemci bunu bilmez: CS:IP nereyi gösteriyorsa oradaki baytı **işlem kodu** sayar.

Adım 1 / 9

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme

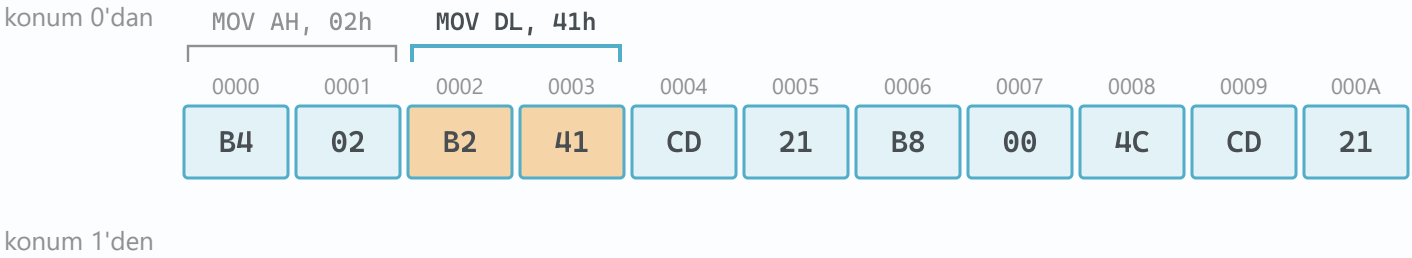


Bağıl konum 0: B4h = MOV AH, imm8 (B0h + 4) → 1 anlık bayt alır: `mov ah, 02h`.

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

KOMUT KODLAMA

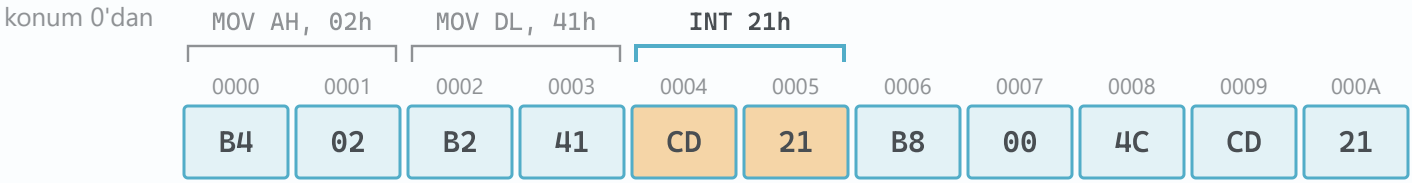
Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme



B2h = MOV DL, imm8 → `mov dl, 41h` (41h = 'A').

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme

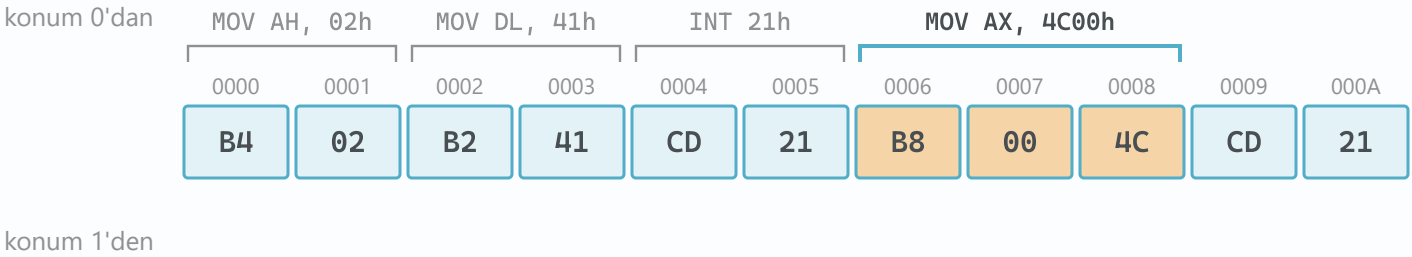


CDh = INT imm8 → `int 21h` : AH = 02h, DOS DL'deki 'A'yı ekrana yazar.

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

KOMUT KODLAMA

Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme

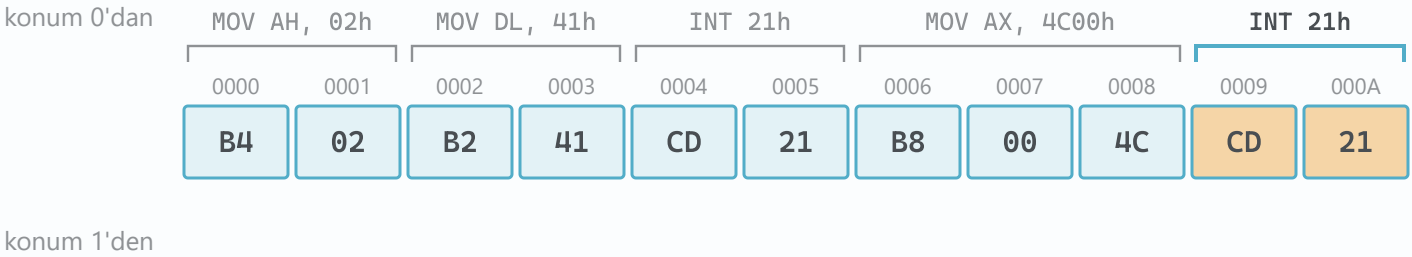


B8h = MOV AX, imm16 → 2 bayt, little-endian: `mov ax, 4C00h`.

Adım 5 / 9

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

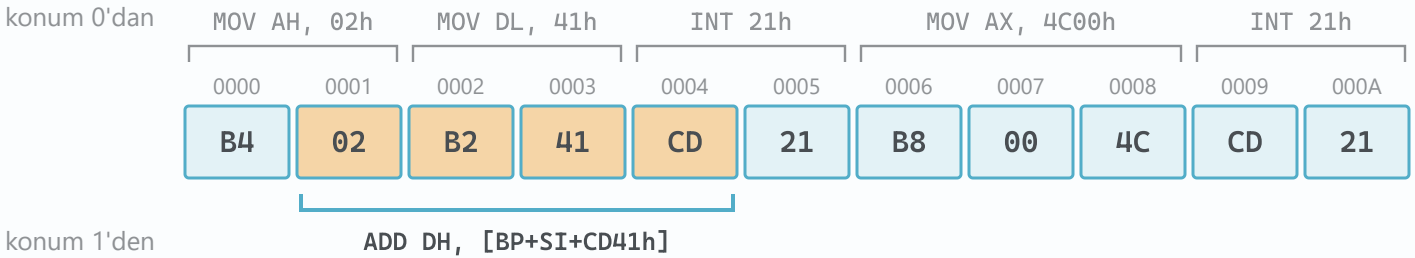
Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme



`int 21h`, AH = 4Ch: program sonlanır. Bağıl konum 0'dan çözüm, programcının yazdığı koddur.

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme

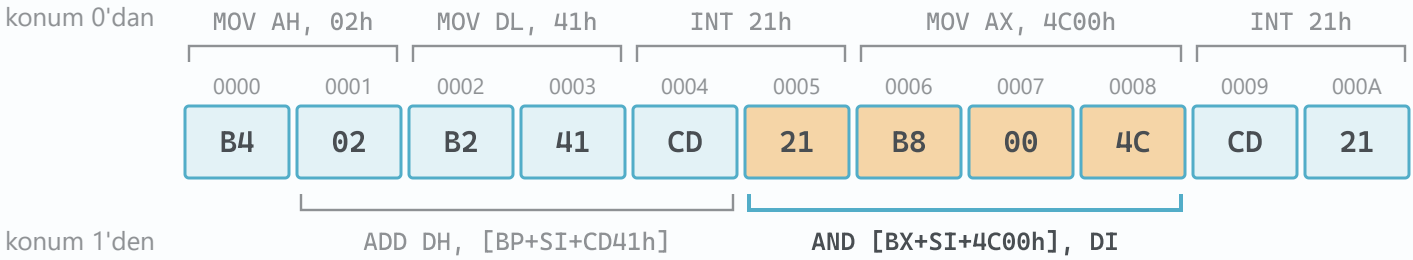


IP bir bayt kaysın (bağıl konum 1). 02h = ADD r8, r/m8; ModR/M B2h = **10 110 010** → MOD = 10 (d16), REG = DH, R/M = [BP+SI]; d16 = CD41h → 4 bayt: 'A' ve INT'in işlem kodu yutuldu.

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

KOMUT KODLAMA

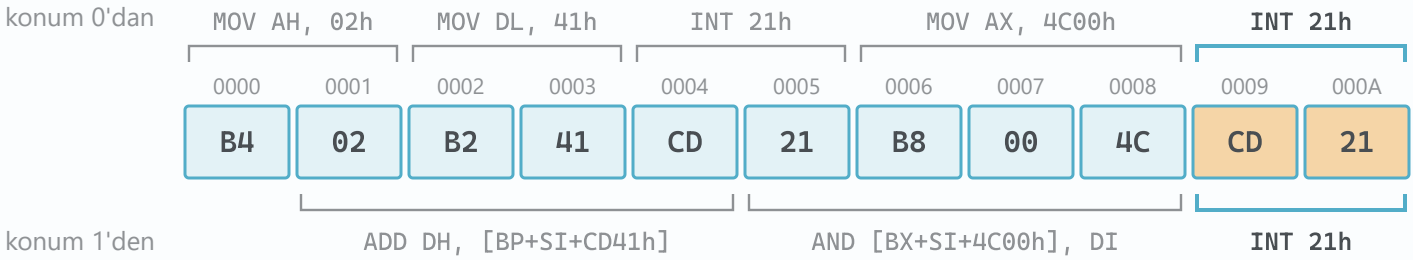
Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme



21h = AND r/m16, r16; ModR/M B8h = **10 111 000** → [BX+SI+d16], REG = DI; d16 = 4C00h. Bu komut **belleğe yazar**.

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme



Son CD 21 iki çözümde de aynı yere düştü: çözme **yeniden eşlendi**. Ama AH = 02h hiç yüklenmedi, INT 21h, AH'de o an ne varsa onunla çağrılır. Anlamı baytlar değil, IP'nin başladığı yer belirler.

8086'da kod ile veri arasında donanım ayrımı yoktur: veriye atlayan bir JMP ya da yanlış hesaplanmış bir IP, baytları komut olarak yürütür. Disassembler da başlangıç noktasını bilmek zorundadır.

VERİ TAŞIMA

Veri Taşıma Komutları

Komut	İşlem	Örnek
`MOV h, k`	h ← k (kopyalama)	`mov bx, [si]`
`PUSH k` / `POP h`	SP ← SP−2, [SS:SP] ← k / h ← [SS:SP], SP ← SP+2	`push ax`
`XCHG a, b`	a ↔ b (biri yazmaç olmalı)	`xchg al, [bx]`
`IN` / `OUT`	AL/AX ↔ G/Ç portu	`in al, 60h`
`XLAT`	AL ← [DS:BX + AL]	tablo çevirisi
`LEA r, m`	r ← m'nin **adresi** (EA)	`lea si, metin`
`LDS` / `LES r, m`	r ← [m], DS/ES ← [m+2] (uzak işaretçi)	`les di, uzak`
`LAHF` / `SAHF`	AH ← FLAGS düşük bayt / tersi	
`PUSHF` / `POPF`	FLAGS'ı yığına at / yığından al	

Veri taşıma komutları bayrakları **etkilemez**; istisnalar SAHF ve POPF'tur (bayrakları doğrudan yazarlar).

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
▶ mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

BX, çeviri tablosunun başlangıç bağıl konumunu (offset) alır: **BX = 011Dh**.

Adım 1 / 14

DEĞİŞKENLER

AL = **00h** BX = **011Dh** CL = **00h** DL = **00h** AH = **00h**

ÇIKTI

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
▶  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat               ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h            ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

Yazdırılacak bayt: **AL = 3Ch** (0011 1100b).

Adım 2 / 14

DEĞİŞKENLER

AL = **3Ch** | BX = **011Dh** | CL = **00h** | DL = **00h** | AH = **00h**

ÇIKTI

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
▶ 4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

8086'da 1'den büyük kaydırma sayısı **CL** ile verilir.

Adım 3 / 14

DEĞİŞKENLER

AL = **3Ch** | BX = **011Dh** | CL = **04h** | DL = **00h** | AH = **00h**

ÇIKTI

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
▶  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

`shr al, cl` : 0011 1100 → 0000 0011. **AL = 03h** (yüksek nibble).

Adım 4 / 14

DEĞİŞKENLER

AL = **03h** | BX = **011Dh** | CL = **04h** | DL = **00h** | AH = **00h**

ÇIKTI

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

`xlat` : AL ← [DS:BX + AL] = tablo[3] = **'3' (33h)**.

Adım 5 / 14

DEĞİŞKENLER

AL = **33h** | BX = **011Dh** | CL = **04h** | DL = **00h** | AH = **00h**

ÇIKTI

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
▶  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

Karakter DL'ye kopyalanır (INT 21h/02h parametresi).

Adım 6 / 14

DEĞİŞKENLER

AL = **33h** BX = **011Dh** CL = **04h** DL = **33h** AH = **00h**

ÇIKTI

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
▶  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

AH = 02h: DOS **karakter yaz** hizmeti.

Adım 7 / 14

DEĞİŞKENLER

AL = **33h** | BX = **011Dh** | CL = **04h** | DL = **33h** | AH = **02h**

ÇIKTI



Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat               ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
▶ int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

DOS ekrana **'3'** yazar (AL de DL'nin değerini alır).

Adım 8 / 14

DEĞİŞKENLER

AL = **33h** BX = **011Dh** CL = **04h** DL = **33h** AH = **02h**

ÇIKTI

```
3
```

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
▶  mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

Aynı bayt yeniden alınır: AL = 3Ch.

Adım 9 / 14

DEĞİŞKENLER

AL = **3Ch** | BX = **011Dh** | CL = **04h** | DL = **33h** | AH = **02h**

ÇIKTI

```
3
```

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat               ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h            ; DL'yi yaz
10 mov al, sayi
▶ and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

`and al, 0Fh` : yüksek nibble maskelenir → **AL = 0Ch**.

Adım 10 / 14

DEĞİŞKENLER

AL = **0Ch** | BX = **011Dh** | CL = **04h** | DL = **33h** | AH = **02h**

ÇIKTI

```
3
```

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
▶  xlat
13 mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

tablo[12] = **'C' (43h)**.

Adım 11 / 14

DEĞİŞKENLER

AL = **43h** | BX = **011Dh** | CL = **04h** | DL = **33h** | AH = **02h**

ÇIKTI

```
3
```

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
▶ mov dl, al
14 int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

DL = 43h. AH hâlâ 02h olduğundan yeniden yüklemeye gerek yok.

Adım 12 / 14

DEĞİŞKENLER

AL = **43h** BX = **011Dh** CL = **04h** DL = **43h** AH = **02h**

ÇIKTI

```
3
```

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat               ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
▶ int 21h
15 ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

Ekrana **'C'** yazılır → çıktı **3C**.

Adım 13 / 14

DEĞİŞKENLER

AL = **43h** BX = **011Dh** CL = **04h** DL = **43h** AH = **02h**

ÇIKTI

```
3C
```

Kod İzleme: XLAT ile Baytı Onaltılık Yazdırma

```
1  org 100h
2  mov bx, offset tablo
3  mov al, sayi        ; AL = 3Ch
4  mov cl, 4
5  shr al, cl          ; üst dörtlü
6  xlat                ; AL = [BX+AL]
7  mov dl, al
8  mov ah, 2
9  int 21h             ; DL'yi yaz
10 mov al, sayi
11 and al, 0Fh         ; alt dörtlü
12 xlat
13 mov dl, al
14 int 21h
▶ ret
16 sayi  db 3Ch
17 tablo db '0123456789ABCDEF'
```

Program biter. XLAT, en fazla 256 baytlık tablolarla tek komutta çeviri yapar.

Adım 14 / 14

DEĞİŞKENLER

AL = **43h** BX = **011Dh** CL = **04h** DL = **43h** AH = **02h**

ÇIKTI

```
3C
```

Adres Yükleme: LEA, LDS, LES

- `lea si, metin` ≡ `mov si, offset metin` ; fark: LEA **çalışma anında** hesaplanan adresleri de verir: `lea si, [bx+di+4]` .
- LEA belleğe erişmez ve bayrakları etkilemez; bu yüzden toplama için de kullanılabilir: `lea ax, [bx+si]` → AX = BX + SI.
- `lds si, ptr` : SI ← [ptr], DS ← [ptr+2]. `les di, ptr` aynı işi ES ile yapar.
- Uzak işaretçi (far pointer) bellekte **bağıl konum önce, kesim sonra** saklanır.

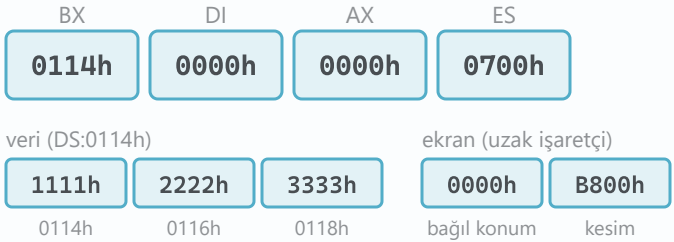
```
mov bx, 0010h
mov di, 0004h
lea ax, [bx+di+2]  ; AX = 0016h
mov ax, [bx+di+2]  ; AX = [0016h]

; ekran belleği: B800:0000
ekran dw 0000h, 0B800h
les di, ekran      ; DI=0000h ES=B800h
mov al, 'A'
stosb              ; köşeye 'A'
```

LEA **adresi**, MOV **içeriği** verir: `lea ax, [bx+di+2]` ile `mov ax, [bx+di+2]` arasındaki fark sınavlarda sık sorulur.

Adres mi, İçerik mi? LEA, MOV ve LES

```
1  org 100h
▶  lea bx, veri          ; BX ← adres
3  mov di, 2
4  lea ax, [bx+di+2]     ; AX ← EA
5  mov ax, [bx+di+2]     ; AX ← [EA]
6  les di, ekran         ; uzak işaretçi
7  mov al, 'A'
8  stosb                 ; ES:[DI] ← AL
9  ret
10 veri  dw 1111h, 2222h, 3333h
11 ekran dw 0000h, 0B800h
```



LEA, **veri** değişkeninin adresini verir: BX = 0114h. Bellek okunmaz.

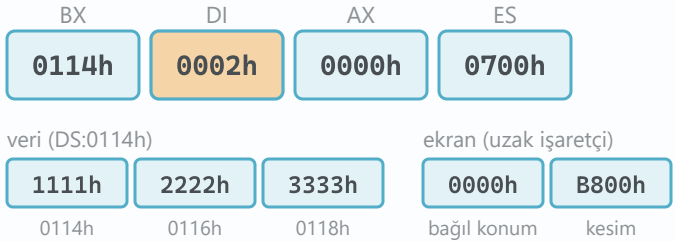
Adım 1 / 8

DEĞİŞKENLER

`BX = 0114h` `DI = 0000h` `AX = 0000h` `ES = 0700h`

Adres mi, İçerik mi? LEA, MOV ve LES

```
1  org 100h
2  lea bx, veri          ; BX ← adres
▶  mov di, 2
4  lea ax, [bx+di+2]     ; AX ← EA
5  mov ax, [bx+di+2]     ; AX ← [EA]
6  les di, ekran         ; uzak işaretçi
7  mov al, 'A'
8  stosb                 ; ES:[DI] ← AL
9  ret
10 veri  dw 1111h, 2222h, 3333h
11 ekran dw 0000h, 0B800h
```



DI = 2.

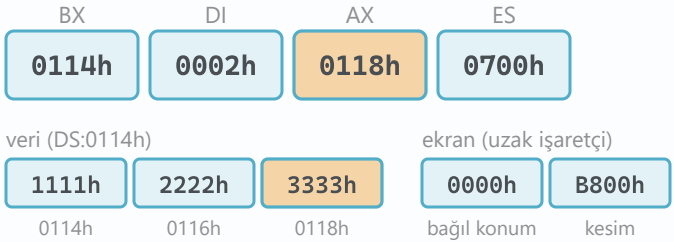
Adım 2 / 8

DEĞİŞKENLER

BX = 0114h | DI = 0002h | AX = 0000h | ES = 0700h

Adres mi, İçerik mi? LEA, MOV ve LES

```
1  org 100h
2  lea bx, veri          ; BX ← adres
3  mov di, 2
▶  lea ax, [bx+di+2]     ; AX ← EA
5  mov ax, [bx+di+2]     ; AX ← [EA]
6  les di, ekran         ; uzak işaretçi
7  mov al, 'A'
8  stosb                 ; ES:[DI] ← AL
9  ret
10 veri  dw 1111h, 2222h, 3333h
11 ekran dw 0000h, 0B800h
```



EA = BX + DI + 2 = 0118h (adres)

LEA AX, [BX+DI+2]: yalnız etkin adres hesaplanır → AX = 0114h + 2 + 2 = **0118h**. Belleğe dokunulmaz, bayrak değişmez.

Adım 3 / 8

DEĞİŞKENLER

BX = 0114h | DI = 0002h | AX = 0118h | ES = 0700h

Adres mi, İçerik mi? LEA, MOV ve LES

```
1  org 100h
2  lea bx, veri          ; BX ← adres
3  mov di, 2
4  lea ax, [bx+di+2]     ; AX ← EA
▶  mov ax, [bx+di+2]     ; AX ← [EA]
6  les di, ekran         ; uzak işaretçi
7  mov al, 'A'
8  stosb                 ; ES:[DI] ← AL
9  ret
10 veri  dw 1111h, 2222h, 3333h
11 ekran dw 0000h, 0B800h
```



[EA] = 3333h (içerik)

MOV AX, [BX+DI+2]: aynı EA'daki **içerik** okunur → AX = **3333h** (veri'nin 3. sözcüğü).

Adım 4 / 8

DEĞİŞKENLER

BX = 0114h | DI = 0002h | AX = 3333h | ES = 0700h

Adres mi, İçerik mi? LEA, MOV ve LES

```
1  org 100h
2  lea bx, veri          ; BX ← adres
3  mov di, 2
4  lea ax, [bx+di+2]     ; AX ← EA
5  mov ax, [bx+di+2]     ; AX ← [EA]
▶  les di, ekran         ; uzak işaretçi
7  mov al, 'A'
8  stosb                 ; ES:[DI] ← AL
9  ret
10 veri  dw 1111h, 2222h, 3333h
11 ekran dw 0000h, 0B800h
```



LES DI, ekran: bellekteki uzak işaretçi (bağıl konum önce, kesim sonra) → DI = **0000h**, ES = **B800h**.

Adım 5 / 8

DEĞİŞKENLER

BX = **0114h** | DI = **0000h** | AX = **3333h** | ES = **B800h**

Adres mi, İçerik mi? LEA, MOV ve LES

```
1  org 100h
2  lea bx, veri          ; BX ← adres
3  mov di, 2
4  lea ax, [bx+di+2]     ; AX ← EA
5  mov ax, [bx+di+2]     ; AX ← [EA]
6  les di, ekran         ; uzak işaretçi
▶  mov al, 'A'
8  stosb                 ; ES:[DI] ← AL
9  ret
10 veri  dw 1111h, 2222h, 3333h
11 ekran dw 0000h, 0B800h
```



AL = 41h ('A').

Adım 6 / 8

DEĞİŞKENLER

BX = 0114h | DI = 0000h | AX = 3341h | ES = B800h

Adres mi, İçerik mi? LEA, MOV ve LES

```
1  org 100h
2  lea bx, veri          ; BX ← adres
3  mov di, 2
4  lea ax, [bx+di+2]     ; AX ← EA
5  mov ax, [bx+di+2]     ; AX ← [EA]
6  les di, ekran         ; uzak işaretçi
7  mov al, 'A'
▶  stosb                 ; ES:[DI] ← AL
9  ret
10 veri  dw 1111h, 2222h, 3333h
11 ekran dw 0000h, 0B800h
```



STOSB: ES:[DI] = B800h × 10h + 0000h = **B8000h** ← 41h; DF = 0 olduğundan DI = 1. Ekranın sol üst köşesinde 'A'.

Adım 7 / 8

DEĞİŞKENLER

BX = 0114h | DI = 0001h | AX = 3341h | ES = B800h

Adres mi, İçerik mi? LEA, MOV ve LES

```
1  org 100h
2  lea bx, veri          ; BX ← adres
3  mov di, 2
4  lea ax, [bx+di+2]     ; AX ← EA
5  mov ax, [bx+di+2]     ; AX ← [EA]
6  les di, ekran         ; uzak işaretçi
7  mov al, 'A'
8  stosb                 ; ES:[DI] ← AL
▶  ret
10 veri  dw 1111h, 2222h, 3333h
11 ekran dw 0000h, 0B800h
```



RET: program biter.

Adım 8 / 8

DEĞİŞKENLER

BX = 0114h | DI = 0001h | AX = 3341h | ES = B800h

Aritmetik, Mantık ve Kaydırma Komutları: Özet

Grup	Komutlar	Açıklama
Toplama	`ADD ADC INC`	ADC eldeyi (CF) de ekler; INC CF'yi değiştirmez
Çıkarma	`SUB SBB DEC NEG CMP`	SBB ödüncü (CF) de çıkarır; NEG = 0 − h; CMP sonucu yazmaz
Çarpma	`MUL IMUL`	işaretsiz / işaretli; örtük AL veya AX
Bölme	`DIV IDIV`	işaretsiz / işaretli; bölüm + kalan
BCD düzeltme	`DAA DAS AAA AAS AAM AAD`	ikili sonucu onluğa düzeltir
İşaret genişletme	`CBW CWD`	AL → AX, AX → DX:AX
Mantık	`AND OR XOR NOT TEST`	bit temizleme, kurma, tersleme; TEST sonucu yazmaz
Kaydırma / döndürme	`SHL SAL SHR SAR ROL ROR RCL RCR`	kaydırma sayısı 1 ya da CL; çıkan bit CF'ye

İşlemci toplamayı ve çıkarmayı **tek bir devreyle** yapar; sayının işaretli mi işaretsiz mi olduğunu bilmez. Bu yüzden her iki yorum için bayrak üretir: **CF** işaretsiz taşmayı, **OF** işaretli taşmayı gösterir.

DİZGİ İŞLEME

Dizgi (String) Komutları

Komut	İşlem	Güncellenen	Bayrak
`MOVSB / MOVSW`	[ES:DI] ← [DS:SI]	SI, DI	–
`CMPSB / CMPSW`	[DS:SI] − [ES:DI]	SI, DI	CMP gibi
`SCASB / SCASW`	AL/AX − [ES:DI]	DI	CMP gibi
`LODSB / LODSW`	AL/AX ← [DS:SI]	SI	–
`STOSB / STOSW`	[ES:DI] ← AL/AX	DI	–

Kaynak / hedef
Kaynak **DS:SI** (segment öneki ile değişebilir), hedef her zaman **ES:DI**.

Yön bayrağı
`cld` → DF = 0, adresler **artar**; `std` → DF = 1, **azalır**.

Adım
B (bayt) sürümünde ±1, W (sözcük) sürümünde ±2.

EXE programlarında ES ayrıca ayarlanmalıdır (`mov ax, @data` / `mov es, ax`); COM programlarında DS = ES = CS'dir.

Yineleme Önekleri: REP, REPE, REPNE

Önek	Sürdürme koşulu	Kullanım
`REP`	CX ≠ 0	MOVS, STOS
`REPE / REPZ`	CX ≠ 0 ve ZF = 1	CMPS, SCAS: "eşitken"
`REPNE / REPNZ`	CX ≠ 0 ve ZF = 0	CMPS, SCAS: "eşit değilken"

1. CX = 0 ise dur (komut hiç çalışmayabilir).


```
; 100 baytlık blok kopyalama
cld
lea si, kaynak
lea di, hedef
mov cx, 100
rep movsb

; ekranı boşlukla doldur
mov ax, 0B800h
mov es, ax
xor di, di
mov ax, 0720h    ; ' ' + öznitelik
mov cx, 2000     ; 80×25
rep stosw
```

Yineleme Önekleri: REP, REPE, REPNE

Önek	Sürdürme koşulu	Kullanım
`REP`	CX ≠ 0	MOVS, STOS
`REPE / REPZ`	CX ≠ 0 ve ZF = 1	CMPS, SCAS: "eşitken"
`REPNE / REPNZ`	CX ≠ 0 ve ZF = 0	CMPS, SCAS: "eşit değilken"

1. CX = 0 ise dur (komut hiç çalışmayabilir).

2. Dizgi komutunu bir kez yürüt, SI/DI'yi güncelle.


```
; 100 baytlık blok kopyalama
cld
lea si, kaynak
lea di, hedef
mov cx, 100
rep movsb

; ekranı boşlukla doldur
mov ax, 0B800h
mov es, ax
xor di, di
mov ax, 0720h    ; ' ' + öznitelik
mov cx, 2000     ; 80×25
rep stosw
```

Yineleme Önekleri: REP, REPE, REPNE

Önek	Sürdürme koşulu	Kullanım
`REP`	CX ≠ 0	MOVS, STOS
`REPE` / `REPZ`	CX ≠ 0 ve ZF = 1	CMPS, SCAS: "eşitken"
`REPNE` / `REPNZ`	CX ≠ 0 ve ZF = 0	CMPS, SCAS: "eşit değilken"

1. CX = 0 ise dur (komut hiç çalışmayabilir).
2. Dizgi komutunu bir kez yürüt, SI/DI'yi güncelle.
3. CX ← CX − 1 (bayraklar değişmez).

```
; 100 baytlık blok kopyalama
cld
lea si, kaynak
lea di, hedef
mov cx, 100
rep movsb

; ekranı boşlukla doldur
mov ax, 0B800h
mov es, ax
xor di, di
mov ax, 0720h    ; ' ' + öznitelik
mov cx, 2000     ; 80×25
rep stosw
```

Yineleme Önekleri: REP, REPE, REPNE

Önek	Sürdürme koşulu	Kullanım
`REP`	CX ≠ 0	MOVS, STOS
`REPE` / `REPZ`	CX ≠ 0 ve ZF = 1	CMPS, SCAS: "eşitken"
`REPNE` / `REPNZ`	CX ≠ 0 ve ZF = 0	CMPS, SCAS: "eşit değilken"

1. CX = 0 ise dur (komut hiç çalışmayabilir).

2. Dizgi komutunu bir kez yürüt, SI/DI'yi güncelle.

3. CX ← CX − 1 (bayraklar değişmez).

4. REPE/REPNE ise ZF koşulunu sına; sağlanmıyorsa dur. Değilse 1'e dön.


```
; 100 baytlık blok kopyalama
cld
lea si, kaynak
lea di, hedef
mov cx, 100
rep movsb

; ekranı boşlukla doldur
mov ax, 0B800h
mov es, ax
xor di, di
mov ax, 0720h   ; ' ' + öznitelik
mov cx, 2000    ; 80×25
rep stosw
```

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
▶  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

DF = 0: ileri yönde kopyalama.

Adım 1 / 13

DEĞİŞKENLER

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

hedef = **'....'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
▶  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

SI = 010Eh: kaynak dizgi.

Adım 2 / 13

DEĞİŞKENLER

`SI = 010Eh` `DI = 0000h` `CX = 0000h`

`hedef = '....'`

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
▶  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

DI = 0112h: hedef alan ('....').

Adım 3 / 13

DEĞİŞKENLER

SI = **010Eh** | DI = **0112h** | CX = **0000h**

hedef = **'....'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
▶  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

Kopyalanacak bayt sayısı CX = 4.

Adım 4 / 13

DEĞİŞKENLER

SI = **010Eh** DI = **0112h** CX = **0004h**

hedef = **'....'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
▶  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

`movsb` : bir bayt kopyalanır; SI = 010Fh, DI = 0113h. Bayraklar değişmez.

Adım 5 / 13

DEĞİŞKENLER

SI = **010Fh** DI = **0113h** CX = **0004h**

hedef = **'A...'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
▶  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

CX = 0003h.

Adım 6 / 13

DEĞİŞKENLER

SI = **010Fh** DI = **0113h** CX = **0003h**

hedef = **'A...'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
▶  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

`movsb` : bir bayt kopyalanır; SI = 0110h, DI = 0114h. Bayraklar değişmez.

Adım 7 / 13

DEĞİŞKENLER

SI = **0110h** DI = **0114h** CX = **0003h**

hedef = **'AB..'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
▶  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

CX = 0002h.

Adım 8 / 13

DEĞİŞKENLER

SI = **0110h** DI = **0114h** CX = **0002h**

hedef = **'AB..'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
▶  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

`movsb` : bir bayt kopyalanır; SI = 0111h, DI = 0115h. Bayraklar değişmez.

Adım 9 / 13

DEĞİŞKENLER

SI = **0111h** DI = **0115h** CX = **0002h**

hedef = **'ABC.'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
▶  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

CX = 0001h.

Adım 10 / 13

DEĞİŞKENLER

SI = **0111h** DI = **0115h** CX = **0001h**

hedef = **'ABC.'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
▶  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

`movsb` : bir bayt kopyalanır; SI = 0112h, DI = 0116h. Bayraklar değişmez.

Adım 11 / 13

DEĞİŞKENLER

SI = **0112h** DI = **0116h** CX = **0001h**

hedef = **'ABCD'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
▶  loop kopya          ; = rep movsb
9  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

CX = 0000h.

Adım 12 / 13

DEĞİŞKENLER

SI = **0112h** DI = **0116h** CX = **0000h**

hedef = **'ABCD'**

Kod İzleme: MOVSB ile Kopyalama

```
1  org 100h
2  cld
3  lea si, kaynak      ; DS:SI kaynak
4  lea di, hedef       ; ES:DI hedef
5  mov cx, 4
6  kopya:
7  movsb               ; [DI]←[SI], SI++,
   DI++
8  loop kopya          ; = rep movsb
▶  ret
10 kaynak db 'ABCD'
11 hedef  db 4 dup('.')
```

hedef = **'ABCD'**. `mov cx, 4` + `rep movsb` aynı işi tek komutla yapar.

Adım 13 / 13

DEĞİŞKENLER

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

`hedef = 'ABCD'`

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
▶  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`cld` : DF = 0 → dizgi komutları SI/DI'yi **artırır**.

Adım 1 / 25

DEĞİŞKENLER

AL = **00h** | SI = **0000h** | DI = **0000h** | CX = **0000h** | metin = **'mikro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
▶  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

SI kaynağı gösterir: SI = 0110h (metin).

Adım 2 / 25

DEĞİŞKENLER

AL = 00h | SI = 0110h | DI = 0000h | CX = 0000h | metin = 'mikro'

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
▶  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

Hedef aynı dizgi: DI = 0110h. COM programında ES = DS olduğundan ES:DI aynı yeri gösterir.

Adım 3 / 25

DEĞİŞKENLER

AL = **00h** | SI = **0110h** | DI = **0110h** | CX = **0000h** | metin = **'mikro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
▶  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

5 karakter işlenecek.

Adım 4 / 25

DEĞİŞKENLER

`AL = 00h` `SI = 0110h` `DI = 0110h` `CX = 0005h`

`metin = 'mikro'`

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
▶  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`lodsb` : AL = [DS:SI] = 6Dh, ardından SI = 0111h.

Adım 5 / 25

DEĞİŞKENLER

AL = **6Dh** SI = **0111h** DI = **0110h** CX = **0005h**

metin = **'mikro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
▶  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

Bit 5 temizlenir: AL = 4Dh.

Adım 6 / 25

DEĞİŞKENLER

AL = **4Dh** | SI = **0111h** | DI = **0110h** | CX = **0005h**

metin = **'mikro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`stosb` : [ES:DI] ← AL, ardından DI = 0111h.

Adım 7 / 25

DEĞİŞKENLER

AL = **4Dh** | SI = **0111h** | DI = **0111h** | CX = **0005h**

metin = **'Mikro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
▶  loop don
11 ret
12 metin db 'mikro'
```

CX = 0004h.

Adım 8 / 25

DEĞİŞKENLER

AL = **4Dh** | SI = **0111h** | DI = **0111h** | CX = **0004h**

metin = **'Mikro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
▶  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`lodsb` : AL = [DS:SI] = 69h, ardından SI = 0112h.

Adım 9 / 25

DEĞİŞKENLER

AL = **69h** SI = **0112h** DI = **0111h** CX = **0004h**

metin = **'Mikro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
▶  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

Bit 5 temizlenir: AL = 49h.

Adım 10 / 25

DEĞİŞKENLER

AL = **49h** | SI = **0112h** | DI = **0111h** | CX = **0004h**

metin = **'Mikro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`stosb` : [ES:DI] ← AL, ardından DI = 0112h.

Adım 11 / 25

DEĞİŞKENLER

AL = **49h** | SI = **0112h** | DI = **0112h** | CX = **0004h**

metin = **'MIkro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
▶  loop don
11 ret
12 metin db 'mikro'
```

CX = 0003h.

Adım 12 / 25

DEĞİŞKENLER

AL = **49h** | SI = **0112h** | DI = **0112h** | CX = **0003h** | metin = **'MIkro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
▶  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`lodsb` : AL = [DS:SI] = 6Bh, ardından SI = 0113h.

Adım 13 / 25

DEĞİŞKENLER

AL = **6Bh** SI = **0113h** DI = **0112h** CX = **0003h**

metin = **'MIkro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
▶  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

Bit 5 temizlenir: AL = 4Bh.

Adım 14 / 25

DEĞİŞKENLER

AL = **4Bh** | SI = **0113h** | DI = **0112h** | CX = **0003h**

metin = **'MIkro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`stosb` : [ES:DI] ← AL, ardından DI = 0113h.

Adım 15 / 25

DEĞİŞKENLER

AL = **4Bh** | SI = **0113h** | DI = **0113h** | CX = **0003h**

metin = **'MIKro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
▶  loop don
11 ret
12 metin db 'mikro'
```

CX = 0002h.

Adım 16 / 25

DEĞİŞKENLER

AL = **4Bh** | SI = **0113h** | DI = **0113h** | CX = **0002h** | metin = **'MIKro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
▶  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`lodsb` : AL = [DS:SI] = 72h, ardından SI = 0114h.

Adım 17 / 25

DEĞİŞKENLER

AL = **72h** | SI = **0114h** | DI = **0113h** | CX = **0002h** | metin = **'MIKro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
▶  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

Bit 5 temizlenir: AL = 52h.

Adım 18 / 25

DEĞİŞKENLER

AL = **52h** | SI = **0114h** | DI = **0113h** | CX = **0002h**

metin = **'MIKro'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`stosb` : [ES:DI] ← AL, ardından DI = 0114h.

Adım 19 / 25

DEĞİŞKENLER

AL = **52h** | SI = **0114h** | DI = **0114h** | CX = **0002h**

metin = **'MIKRo'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
▶  loop don
11 ret
12 metin db 'mikro'
```

CX = 0001h.

Adım 20 / 25

DEĞİŞKENLER

AL = **52h** | SI = **0114h** | DI = **0114h** | CX = **0001h** | metin = **'MIKRo'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
▶  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`lodsb` : AL = [DS:SI] = 6Fh, ardından SI = 0115h.

Adım 21 / 25

DEĞİŞKENLER

AL = **6Fh** SI = **0115h** DI = **0114h** CX = **0001h**

metin = **'MIKRo'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
▶  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

Bit 5 temizlenir: AL = 4Fh.

Adım 22 / 25

DEĞİŞKENLER

AL = **4Fh** | SI = **0115h** | DI = **0114h** | CX = **0001h**

metin = **'MIKRo'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`stosb` : [ES:DI] ← AL, ardından DI = 0115h.

Adım 23 / 25

DEĞİŞKENLER

AL = **4Fh** | SI = **0115h** | DI = **0115h** | CX = **0001h**

metin = **'MIKRO'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
▶ loop don
11 ret
12 metin db 'mikro'
```

CX = 0000h.

Adım 24 / 25

DEĞİŞKENLER

AL = **4Fh** | SI = **0115h** | DI = **0115h** | CX = **0000h** | metin = **'MIKRO'**

Kod İzleme: LODSB + STOSB ile Büyük Harfe Çevirme

```
1  org 100h
2  cld                ; DF = 0: ileri
3  lea si, metin
4  mov di, si         ; aynı yere yaz
5  mov cx, 5
6  don:
7  lodsb              ; AL = [SI], SI++
8  and al, 0DFh       ; büyük harf
9  stosb              ; [DI] = AL, DI++
10 loop don
▶ ret
12 metin db 'mikro'
```

metin = **'MIKRO'**. LODS/STOS çifti diziyi yerinde dönüştürmenin tipik kalıbıdır.

Adım 25 / 25

DEĞİŞKENLER

AL = **4Fh** | SI = **0115h** | DI = **0115h** | CX = **0000h** | metin = **'MIKRO'**

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
▶  cld
3  lea di, metin
4  mov cx, 7          ; uzunluk
5  mov al, 'h'        ; aranan
6  repne scasb        ; ara
7  jne yok
8  dec di             ; bulunan konum
9  sub di, offset metin
10 yok:
11 ret
12 metin db 'merhaba'
```

İleri yönde tarama.

Adım 1 / 9

DEĞİŞKENLER

AL = **00h** DI = **0000h** CX = **0000h** ZF = **0**

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
▶  lea di, metin
4  mov cx, 7          ; uzunluk
5  mov al, 'h'        ; aranan
6  repne scasb        ; ara
7  jne yok
8  dec di             ; bulunan konum
9  sub di, offset metin
10 yok:
11 ret
12 metin db 'merhaba'
```

DI = 0113h: taranacak dizgi ES:DI'de olmalı.

Adım 2 / 9

DEĞİŞKENLER

AL = 00h | DI = 0113h | CX = 0000h | ZF = 0

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
3  lea di, metin
▶ 4  mov cx, 7          ; uzunluk
5  mov al, 'h'          ; aranan
6  repne scasb          ; ara
7  jne yok
8  dec di               ; bulunan konum
9  sub di, offset metin
10 yok:
11 ret
12 metin db 'merhaba'
```

En fazla 7 karşılaştırma.

Adım 3 / 9

DEĞİŞKENLER

AL = **00h** | DI = **0113h** | CX = **0007h** | ZF = **0**

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
3  lea di, metin
4  mov cx, 7          ; uzunluk
▶  mov al, 'h'        ; aranan
6  repne scasb        ; ara
7  jne yok
8  dec di             ; bulunan konum
9  sub di, offset metin
10 yok:
11 ret
12 metin db 'merhaba'
```

Aranan karakter AL = 'h' (68h).

Adım 4 / 9

DEĞİŞKENLER

AL = **68h** | DI = **0113h** | CX = **0007h** | ZF = **0**

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
3  lea di, metin
4  mov cx, 7          ; uzunluk
5  mov al, 'h'        ; aranan
6  repne scasb        ; ara
7  jne yok
8  dec di             ; bulunan konum
9  sub di, offset metin
10 yok:
11 ret
12 metin db 'merhaba'
```

`repne scasb`: 'm','e','r','h' ile karşılaştırılır; 4. karşılaştırmada **ZF = 1** olur ve durur. Her turda DI++ ve CX−−: **DI = 0117h, CX = 0003h**.

Adım 5 / 9

DEĞİŞKENLER

AL = **68h** | DI = **0117h** | CX = **0003h** | ZF = **1**

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
3  lea di, metin
4  mov cx, 7          ; uzunluk
5  mov al, 'h'        ; aranan
6  repne scasb        ; ara
▶  jne yok
8  dec di             ; bulunan konum
9  sub di, offset metin
10 yok:
11 ret
12 metin db 'merhaba'
```

ZF = 1 (bulundu) → atlama yok.

Adım 6 / 9

DEĞİŞKENLER

AL = **68h** | DI = **0117h** | CX = **0003h** | ZF = **1**

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
3  lea di, metin
4  mov cx, 7          ; uzunluk
5  mov al, 'h'        ; aranan
6  repne scasb        ; ara
7  jne yok
▶  dec di             ; bulunan konum
9  sub di, offset metin
10 yok:
11 ret
12 metin db 'merhaba'
```

DI eşleşmeden **bir sonrayı** gösterir; geri al: DI = 0116h.

Adım 7 / 9

DEĞİŞKENLER

AL = **68h** | DI = **0116h** | CX = **0003h** | ZF = **0**

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
3  lea di, metin
4  mov cx, 7          ; uzunluk
5  mov al, 'h'        ; aranan
6  repne scasb        ; ara
7  jne yok
8  dec di             ; bulunan konum
▶  sub di, offset metin
10 yok:
11 ret
12 metin db 'merhaba'
```

Başlangıç çıkarılır: **DI = 3** → 'h' 3. indiste (0'dan sayarak).

Adım 8 / 9

DEĞİŞKENLER

AL = **68h** | DI = **0003h** | CX = **0003h** | ZF = **0**

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
3  lea di, metin
4  mov cx, 7          ; uzunluk
5  mov al, 'h'        ; aranan
6  repne scasb        ; ara
7  jne yok
8  dec di             ; bulunan konum
9  sub di, offset metin
10 yok:
▶ ret
12 metin db 'merhaba'
```

Bulunamasaydı CX = 0 ve ZF = 0 ile çıkılır, `jne yok` atlardı.

Adım 9 / 9

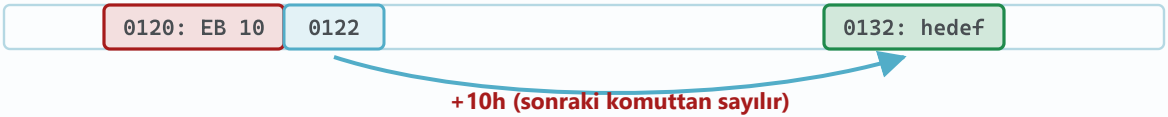
DEĞİŞKENLER

AL = **68h** | DI = **0003h** | CX = **0003h** | ZF = **0**

Koşulsuz Atlama: JMP

Tür	Kod	Erim	Örnek
Kısa (short)	`EB d8`	−128 … +127 bayt	`jmp short dongu`
Yakın (near)	`E9 d16`	aynı kesim, ±32 K	`jmp basla`
Uzak (far)	`EA IP CS`	herhangi bir kesim	`jmp far ptr hedef`
Dolaylı yakın	`FF /4`	IP ← yazmaç/bellek	`jmp bx`, `jmp tablo[bx]`
Dolaylı uzak	`FF /5`	CS:IP ← bellek (4 bayt)	`jmp dword ptr [si]`

- Kısa ve yakın atlamalar **göreli**dir: hedef = sonraki komutun adresi + yer değiştirme. Bu sayede kod bellekte taşınabilir.
- Dolaylı atlama ile **atlama tablosu** (switch-case) kurulur: `shl bx, 1` / `jmp tablo[bx]`.
- `EB FE` kendine atlar (sonsuz döngü): FEh = −2 ve komut 2 bayttır.



İŞLEMCİ KONTROLÜ

İşlemci Kontrolü Komutları

Komut	Etki
`CLC` / `STC`	CF ← 0 / CF ← 1
`CMC`	CF ← ¬CF
`CLD` / `STD`	DF ← 0 / 1 (dizgi yönü)
`CLI` / `STI`	IF ← 0 / 1 (INTR kesmeleri)

Komut	Etki
`HLT`	kesme ya da RESET gelene kadar dur
`WAIT`	TEST̄ ucu etkinleşene kadar bekle (8087 eşlemesi)
`ESC`	komutu yardımcı işlemciye (8087) ilet
`LOCK`	önek: sonraki komut süresince LOCK̄ ucu → yol kilitlenir
`NOP`	hiçbir şey yapma (90h), 3 saat turu

CLC/STC
Prosedürden başarı/hata döndürmenin yaygın yolu: hata varsa `stc`, çağıran `jc hata`.

CLI/STI
Kritik bölgeyi (ör. SS:SP değiştirme) kesmelerden korur. NMI maskelenemez.

LOCK
Çok işlemcili sistemde `lock xchg` ile atomik semafor kurulur.

Bayrak Etkileri Tablosu: Hesaplama Komutları

Komut	OF	SF	ZF	AF	PF	CF
`ADD ADC SUB SBB CMP NEG`	*	*	*	*	*	*
`CMPS SCAS`	*	*	*	*	*	*
`INC DEC`	*	*	*	*	*	–
`MUL IMUL`	*	?	?	?	?	*
`DIV IDIV`	?	?	?	?	?	?
`DAA DAS`	?	*	*	*	*	*
`AAA AAS`	?	?	?	*	?	*
`AAM AAD`	?	*	*	?	*	?
`AND OR XOR TEST`	0	*	*	?	*	0
`SHL SAL SHR SAR`	*[1]	*	*	?	*	*
`ROL ROR RCL RCR`	*[1]	–	–	–	–	*

***** sonuca göre değişir · **0** temizlenir · **–** değişmez · **?** tanımsız (güvenilmez) · [1] OF yalnızca kaydırma sayısı 1 iken tanımlı. DF, IF, TF bu komutlardan etkilenmez.

Bayrak Etkileri Tablosu: Diğer Komutlar

Komutlar	Etkilenen bayraklar
`MOV XCHG LEA LDS LES XLAT PUSH POP IN OUT LAHF PUSHF CBW CWD NOT`	**hiçbiri**
`MOVS LODS STOS`, `REP`	hiçbiri
`JMP Jcc LOOP LOOPE LOOPNE JCXZ CALL RET`	hiçbiri (yalnızca okunur)
`SAHF`	SF ZF AF PF CF ← AH
`POPF`, `IRET`	tümü (yığından)
`INT n`, `INTO`	IF ← 0, TF ← 0 (eski FLAGS yığında)
`CLC STC CMC`	CF
`CLD STD`	DF
`CLI STI`	IF

Sık yapılan hata: `cmp` ile `jcc` arasına bayrak değiştiren bir komut (ör. `inc`, `add`) koymak. Araya yalnızca MOV, PUSH, LEA gibi bayrak etkilemeyen komutlar girebilir.

Laboratuvar: Komut Kümesi

XLAT ile onaltılık yazdırma

KOD · **45 bayt**

```
 1 org 100h
 2 mov bx, offset tablo
 3 mov al, sayi          ; AL = …
 4 mov cl, 4
 5 shr al, cl            ; üst d…
 6 xlat                  ; AL = …
 7 mov dl, al
 8 mov ah, 2
 9 int 21h               ; DL'yi…
10 mov al, sayi
11 and al, 0Fh           ; alt d…
12 xlat
13 mov dl, al
14 int 21h
```

YAZMAÇLAR

AX	0000	AH	00	AL	00
BX	0000	BH	00	BL	00
CX	002D	CH	00	CL	2D
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**

```
0118 D0 CD 21 C3 3C 30 31 32
0120 33 34 35 36 37 38 39 41
0128 42 43 44 45 46 00 00 00
0130 00 00 00 00 00 00 00 00
0138 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: 45 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (002Dh) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

Kodlama

Komut 1–6 bayt: işlem kodu (D, W) + ModR/M + yer değiştirme + anlık veri; önekler başa eklenir.

ModR/M

MOD = 11 yazmaç, 00/01/10 bellek; MOD = 00, R/M = 110 doğrudan adres olduğu için [BP] d8 = 0 ile kodlanır.

Veri taşıma

XLAT: AL ← [DS:BX + AL]. LEA adresi, MOV içeriği verir; LDS/LES uzak işaretçiyi yükler. Bayraklar değişmez.

Dizgi

DS:SI → ES:DI, DF yönü, REP/REPE/REPNE ile CX kadar yineleme; CX = 0 ise komut hiç çalışmaz.

JMP

Kısa (EB d8) ve yakın (E9 d16) atlama göreli, uzak ve dolaylı atlama mutlak adres kullanır.

Bayrak etkileri

Taşıma, dizgi taşıma ve atlama komutları bayrak değiştirmez; CMP ile Jcc arasına bayrak etkileyen komut girmemeli.

ÖZET

Alıştırmalar

1. `mov dx, [bp+di+8]` komutunu kodlayın (opcode 100010, d = 1, w = 1).
2. DS = 1000h, BX = 0200h, AL = 05h iken `xlat` hangi fiziksel adresteki baytı AL'ye yükler?
3. Bellekteki `uzak dd` değişkeninin baytları 34 12 00 B8. `les di, uzak` sonrası ES ve DI nedir?
4. CX = 0 iken `rep movsb` kaç bayt kopyalar?
5. CLD, CX = 5, SI = 0100h, DI = 0200h; `repe cmpsb` 3. baytta fark buluyor. Sonra CX, SI, DI ve ZF nedir?
6. 0150h adresindeki `jmp short` komutu 0130h'ye atlıyor. Makine kodu nedir?
7. Bir dizgideki boşlukları sayan kodu `repne scasb` ile yazın.

▶ **İpuçları / cevaplar**

ÖZET

Alıştırmalar (2)

8. `8A 1E 34 12` baytları hangi komuttur? Aynı baytlar bir bayt kaydırılarak (bağıl konum 1'den) çözülürse ne olur?
9. `mov es, dx` ve `mov al, [0020h]` komutlarını kodlayın.

▶ **İpuçları / cevaplar**

ÖZET

Kaynaklar

- Intel, *The 8086 Family User's Manual*, 1979 (Bölüm 2: komut kümesi, Bölüm 4: komut kodlama).
- Barry B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009.
- Kip R. Irvine, *Assembly Language for x86 Processors*, 7. baskı, Pearson, 2014.
- Ş. Ç. Öztürk, O. Ergin, *Intel 8086 ile Mikroişlemci Programlamaya Giriş*, 2009.
- emu8086 yardım belgeleri: *8086 instructions* ve *8086 assembler tutorial*.

Son

Bölüm 11: Komut Kodlama ve Dizgi Komutları

Terim sözlüğü