Bölüm 10: Komut Kümesi

Mikroişlemciler

Komut yapısı · Makine kodu · Adresleme · Veri taşıma · Aritmetik · Mantık · Kaydırma / döndürme · Dizgi · Kontrol aktarma · İşlemci kontrolü · Bayrak etkileri

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Komut sınıfları

8086'nın yaklaşık 100 komutunu yedi 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.

Bayrak etkileri

Hangi komutun hangi bayrağı nasıl değiştirdiğini bilmek; işaretli ve işaretsiz karşılaştırmayı ayırt etmek.

Program izleme

Kaydırma, dizgi, döngü ve çağrı komutları içeren kısa programları yazmaç düzeyinde adım adım izlemek.

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 YAPISI

Komut Satırının Yapısı ve İşlenenler

```
; etiket:  komut   hedef, kaynak   ; yorum
dongu:    add     ax, [bx+2]      ; AX ← AX + bellek
```

İşlenen türleri

- **Yazmaç**: AX, BL, SI, DS …
- **Bellek**: `[bx]` , `sayi` , `[bp+si+4]`
- **Anlık** (immediate): `5` , `0Fh` , `'A'` , `offset msg`
- Sonuç her zaman **hedefe** (ilk işlenen) yazılır.

Biçim	Örnek	Geçerli?
yazmaç ← yazmaç	`mov ax, bx`	✓
yazmaç ↔ bellek	`mov al, [si]`	✓
yazmaç/bellek ← anlık	`mov byte ptr [di], 5`	✓
bellek ← bellek	`mov [di], [si]`	✗
kesim ← anlık	`mov ds, 2000h`	✗
CS hedef	`mov cs, ax`	✗
farklı boyut	`mov ax, bl`	✗

Bellekten belleğe aktarım bir ara yazmaçla (`mov al, [si]` / `mov [di], al`) ya da dizgi komutu `movsb` ile yapılır. Kesim yazmacına değer: `mov ax, 2000h` / `mov ds, ax` .

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

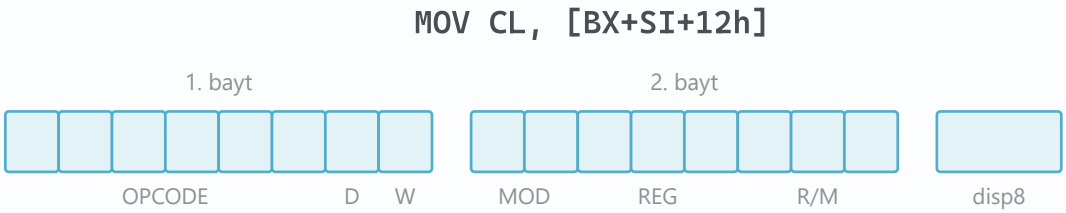
Örnek: `mov ax, bx` → **8B C3**: 100010 **1 1** | **11** 000 011 (D = 1, hedef REG = AX, R/M = BX). Aynı komut **89 D8** olarak da kodlanabilir (D = 0).

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

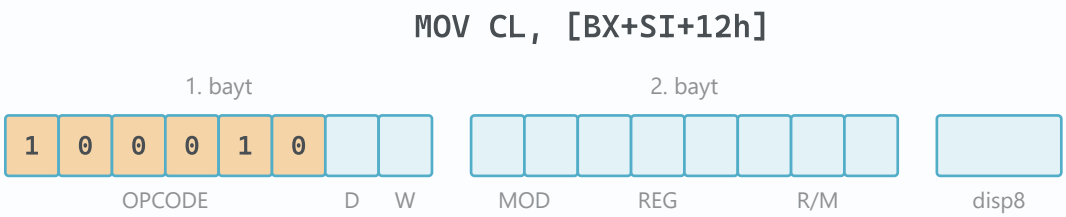
- Yalnızca **BX, BP, SI, DI** adres yazmacı olabilir; `[ax]`, `[cx]`, `[si+di]` geçersizdir.
- `[bp]` yazılınca assembler onu `[bp+0]` (MOD = 01, d8 = 0) olarak kodlar; çünkü MOD = 00, R/M = 110 doğrudan adrese ayrılmıştır.
- Kesim öneki varsayılanı değiştirir: `mov al, es:[bx]` → önek **26h**.

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



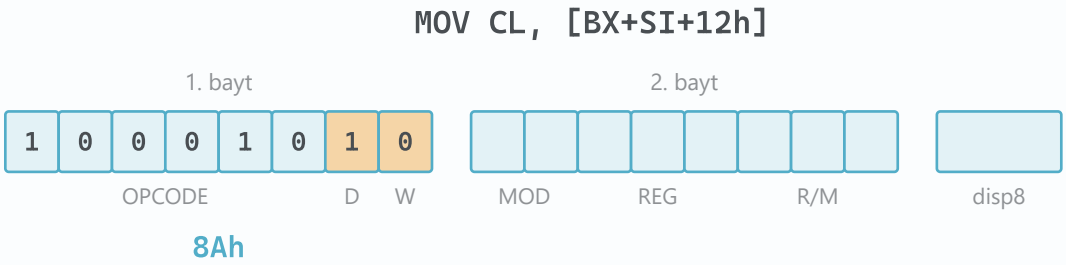
Komut: bellekteki bir baytı CL'ye taşı. Biçim: **MOV reg, r/m** (işlem kodu + ModR/M + yer değiştirme).

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



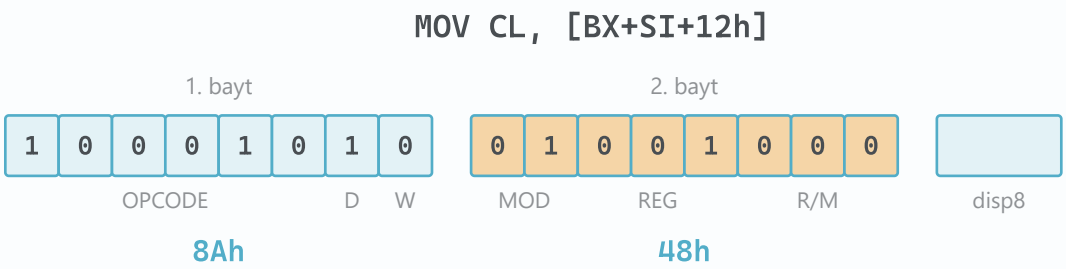
MOV r/m ↔ reg ailesinin işlem kodu **100010**.

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



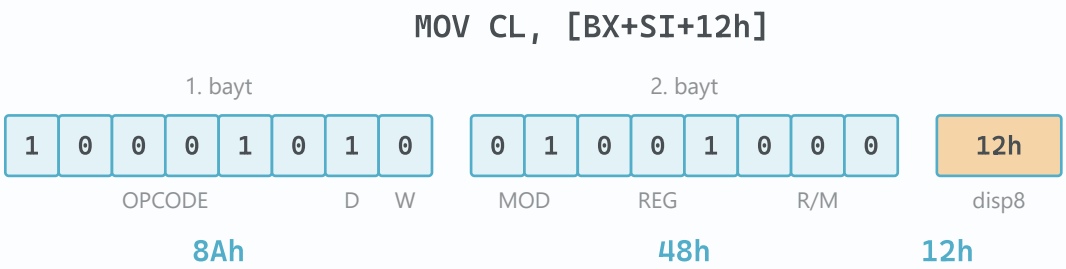
D = 1: REG alanı **hedef**tir. **W = 0**: bayt işlem. 1. bayt = 1000 1010b = **8Ah**.

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.

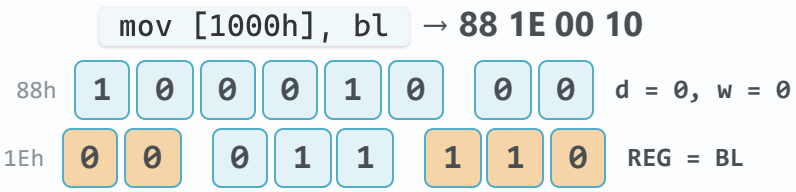
`mov ds, ax → 8E D8`



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

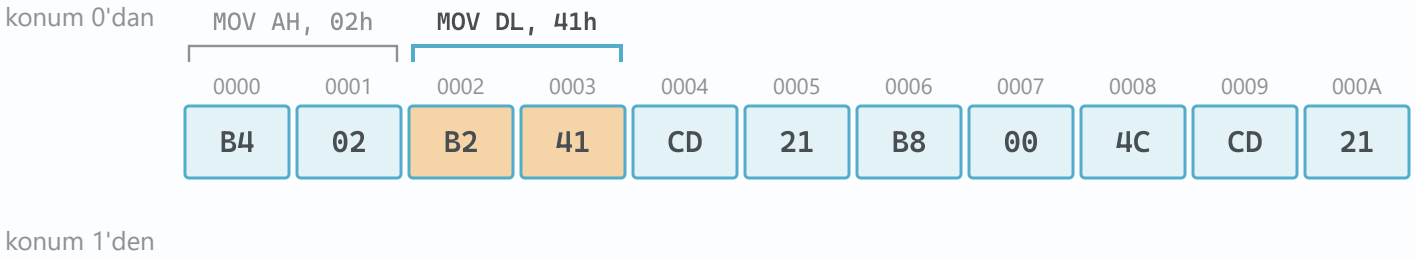


Bağıl konum 0: B4h = MOV AH, imm8 (B0h + 4) → 1 anlık bayt alır: `mov ah, 02h`.

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

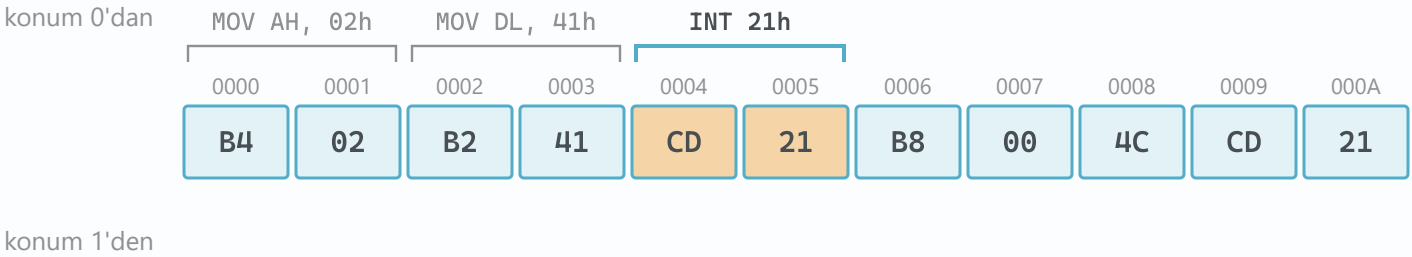


B2h = MOV DL, imm8 → `mov dl, 41h` (41h = 'A').

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.

KOMUT KODLAMA

Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme



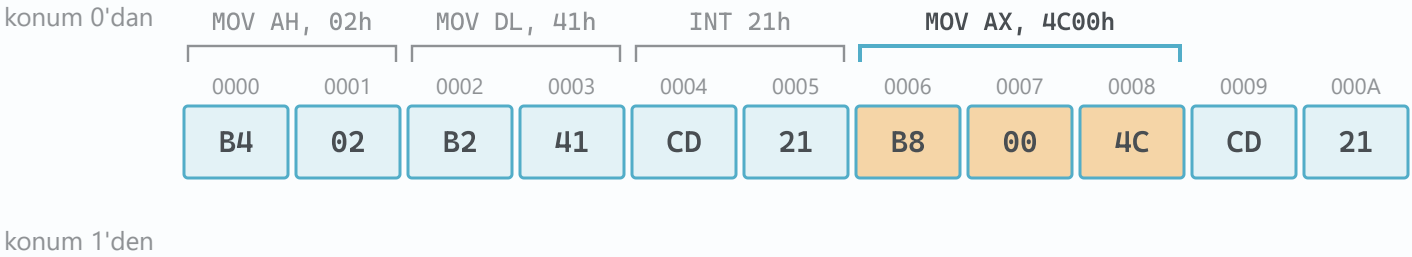
CDh = INT imm8 → `int 21h` : AH = 02h, DOS DL'deki 'A'yı ekrana yazar.

Adım 4 / 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.

KOMUT KODLAMA

Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme

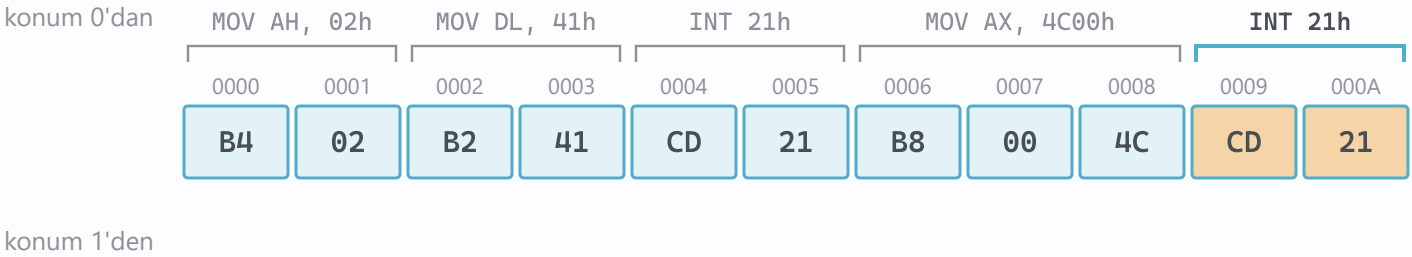


B8h = MOV AX, imm16 → 2 bayt, little-endian: `mov ax, 4C00h`.

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.

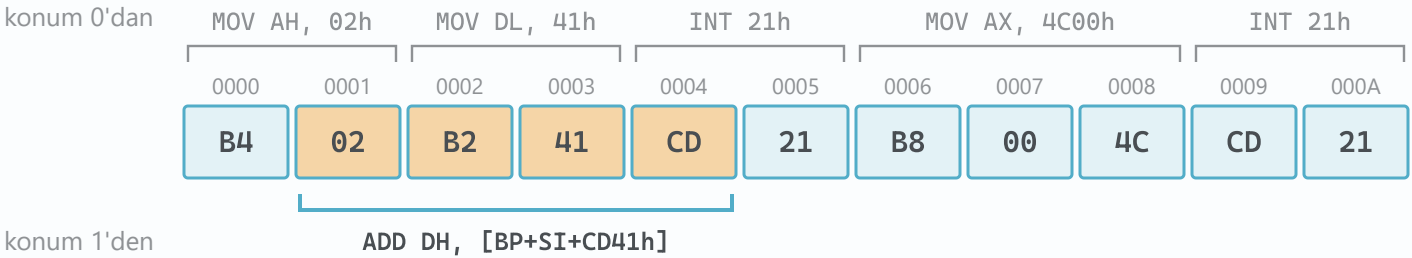
Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme



`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

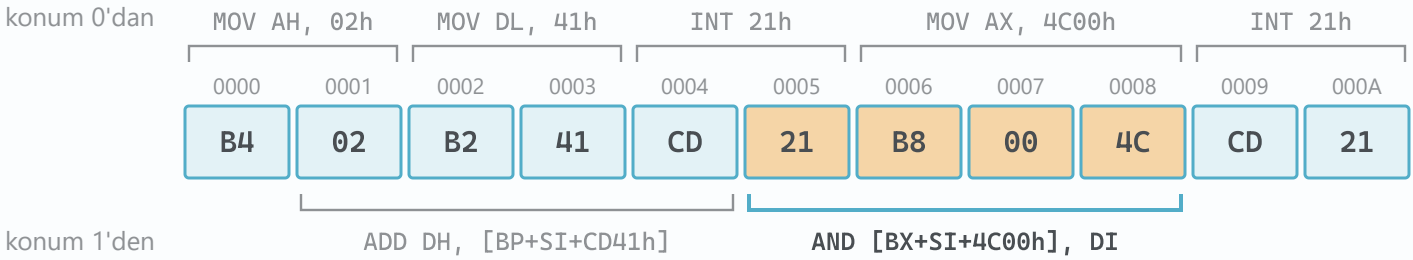


IP bir bayt kaysın (bağıl konum 1). 02h = ADD r8, r/m8; ModR/M B2h = **10 110 010** → MOD = 10 (d16), REG = DH, R/M = [BP+SI]; d16 = CD41h → 4 bayt: 'A' ve INT'in işlem kodu yutuldu.

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

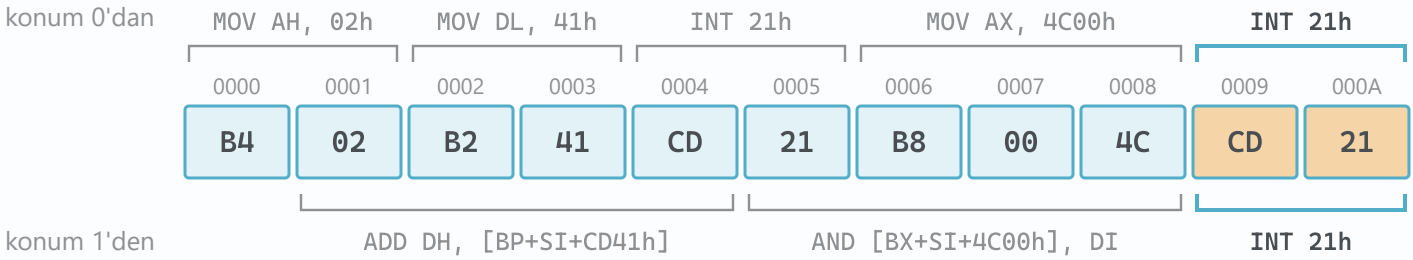
Aynı Baytlar, Farklı Komutlar: Yanlış Bağıl Konumdan Çözme



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.

Adresleme Modları (Özet)

Mod	Örnek	İşlenen nerede?
Anlık	`mov ax, 1234h`	komutun içinde
Yazmaç	`mov ax, bx`	yazmaçta; bellek erişimi yok (en hızlı)
Doğrudan	`mov al, [0120h]`, `mov al, sayi`	EA = komuttaki adres (DS)
Yazmaç dolaylı	`mov al, [bx]`, `[si]`, `[di]`	EA = yazmaç (DS)
Tabanlı	`mov al, [bx+4]`, `[bp+4]`	EA = BX/BP + yer değiştirme (BP → SS)
İndisli	`mov al, dizi[si]`	EA = SI/DI + yer değiştirme
Tabanlı-indisli	`mov al, [bx+si+4]`	EA = taban + indis (+ yer değiştirme)
Örtük	`cbw`, `movsb`, `xlat`	yazmaçlar komutun tanımında sabit
G/Ç portu	`in al, 60h` / `out dx, al`	sabit 8 bit port ya da DX

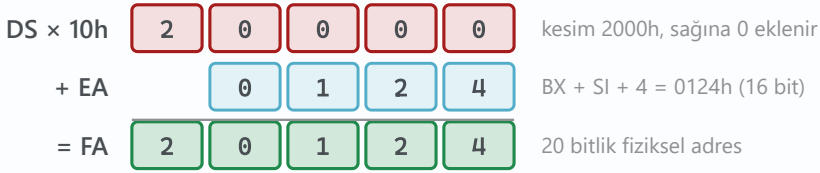
Ayrıntılar Bölüm 4: Bellek Erişimi'nde. Fiziksel adres = kesim × 16 + EA (20 bit).

Fiziksel Adres Hesabı Adım Adım

DS = **2000h**, BX = **0100h**, SI = **0020h**, SS = **3000h**, BP = **FFF0h**

> `mov al, [bx+si+4]` : tabanlı-indisli + d8 → BP yok, varsayılan kesim **DS**.

Adım 1 / 6



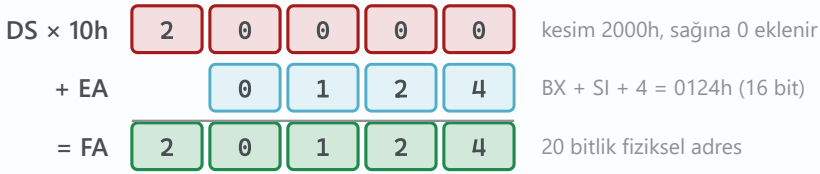
Fiziksel Adres Hesabı Adım Adım

DS = **2000h**, BX = **0100h**, SI = **0020h**, SS = **3000h**, BP = **FFF0h**

> `mov al, [bx+si+4]` : tabanlı-indisli + d8 → BP yok, varsayılan kesim **DS**.

EA = BX + SI + 4 = 0100h + 0020h + 0004h = **0124h** (16 bit).

Adım 2 / 6



ADRESLEME MODLARI

Fiziksel Adres Hesabı Adım Adım

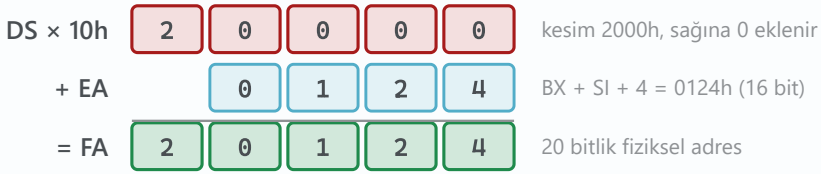
DS = **2000h**, BX = **0100h**, SI = **0020h**, SS = **3000h**, BP = **FFF0h**

> `mov al, [bx+si+4]` : tabanlı-indisli + d8 → BP yok, varsayılan kesim **DS**.

> EA = BX + SI + 4 = 0100h + 0020h + 0004h = **0124h** (16 bit).

> Fiziksel adres = DS × 10h + EA = 20000h + 0124h = **20124h**.

Adım 3 / 6



Fiziksel Adres Hesabı Adım Adım

DS = **2000h**, BX = **0100h**, SI = **0020h**, SS = **3000h**, BP = **FFF0h**

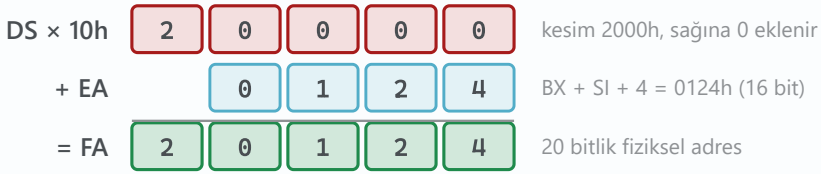
> `mov al, [bx+si+4]` : tabanlı-indisli + d8 → BP yok, varsayılan kesim **DS**.

> EA = BX + SI + 4 = 0100h + 0020h + 0004h = **0124h** (16 bit).

> Fiziksel adres = DS × 10h + EA = 20000h + 0124h = **20124h**.

> `mov ax, [bp+2]` : BP kullanıldığı için kesim **SS**. EA = FFF0h + 2 = **FFF2h**.

Adım 4 / 6



Fiziksel Adres Hesabı Adım Adım

DS = **2000h**, BX = **0100h**, SI = **0020h**, SS = **3000h**, BP = **FFF0h**

> `mov al, [bx+si+4]` : tabanlı-indisli + d8 → BP yok, varsayılan kesim **DS**.

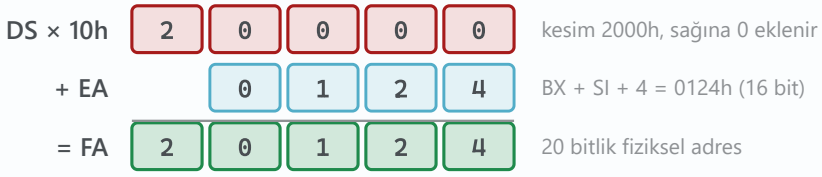
> EA = BX + SI + 4 = 0100h + 0020h + 0004h = **0124h** (16 bit).

> Fiziksel adres = DS × 10h + EA = 20000h + 0124h = **20124h**.

> `mov ax, [bp+2]` : BP kullanıldığı için kesim **SS**. EA = FFF0h + 2 = **FFF2h**.

> Fiziksel adres = 30000h + FFF2h = **3FFF2h**; AL ← [3FFF2h], AH ← [3FFF3h] (little-endian).

Adım 5 / 6



Fiziksel Adres Hesabı Adım Adım

DS = **2000h**, BX = **0100h**, SI = **0020h**, SS = **3000h**, BP = **FFF0h**

> `mov al, [bx+si+4]` : tabanlı-indisli + d8 → BP yok, varsayılan kesim **DS**.

> EA = BX + SI + 4 = 0100h + 0020h + 0004h = **0124h** (16 bit).

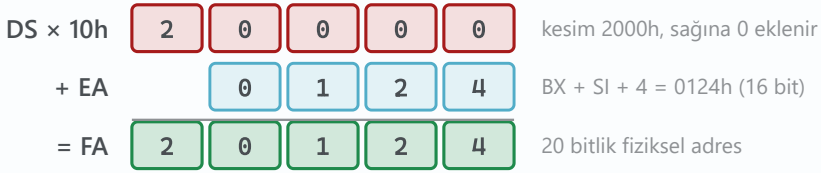
> Fiziksel adres = DS × 10h + EA = 20000h + 0124h = **20124h**.

> `mov ax, [bp+2]` : BP kullanıldığı için kesim **SS**. EA = FFF0h + 2 = **FFF2h**.

> Fiziksel adres = 30000h + FFF2h = **3FFF2h**; AL ← [3FFF2h], AH ← [3FFF3h] (little-endian).

> Dikkat: EA 16 bitte **sarar**. BX = FFFFh, SI = 0002h ise EA = 1 0001h değil **0001h** olur.

Adım 6 / 6



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: PUSH / POP ile Takas ve XCHG

```
1  org 100h
▶  mov ax, 1234h
3  mov bx, 5678h
4  push ax          ; SP -= 2, [SP] = AX
5  push bx
6  pop ax           ; AX = son giren
7  pop bx           ; BX = 1234h
8  xchg ah, al      ; baytları takas et
9  xchg ax, bx
10 ret
```

AX'e anlık değer yüklenir: **AX = 1234h**. MOV hiçbir bayrağı etkilemez.

Adım 1 / 9

DEĞİŞKENLER

AX = **1234h** BX = **0000h** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: PUSH / POP ile Takas ve XCHG

```
1  org 100h
2  mov ax, 1234h
▶  mov bx, 5678h
4  push ax          ; SP -= 2, [SP] = AX
5  push bx
6  pop ax           ; AX = son giren
7  pop bx           ; BX = 1234h
8  xchg ah, al      ; baytları takas et
9  xchg ax, bx
10 ret
```

BX = 5678h. Yığın henüz boş: COM programında **SP = FFFEh**.

Adım 2 / 9

DEĞİŞKENLER

AX = **1234h** | BX = **5678h** | SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: PUSH / POP ile Takas ve XCHG

```
1  org 100h
2  mov ax, 1234h
3  mov bx, 5678h
▶  push ax          ; SP -= 2, [SP] = AX
5  push bx
6  pop ax           ; AX = son giren
7  pop bx           ; BX = 1234h
8  xchg ah, al      ; baytları takas et
9  xchg ax, bx
10 ret
```

`push ax` : önce **SP ← SP − 2 = FFFCh**, sonra [SS:FFFCh] ← 1234h (düşük bayt 34h küçük adreste).

Adım 3 / 9

DEĞİŞKENLER

AX = **1234h** BX = **5678h** SP = **FFFCh**

ÇAĞRI YIĞINI

```
FFFCh: 1234h
```

Kod İzleme: PUSH / POP ile Takas ve XCHG

```
1  org 100h
2  mov ax, 1234h
3  mov bx, 5678h
4  push ax          ; SP -= 2, [SP] = AX
▶  push bx
6  pop ax           ; AX = son giren
7  pop bx           ; BX = 1234h
8  xchg ah, al      ; baytları takas et
9  xchg ax, bx
10 ret
```

`push bx` : SP = FFFAh, yığının üstünde artık **5678h** var.

Adım 4 / 9

DEĞİŞKENLER

AX = **1234h** BX = **5678h** SP = **FFFAh**

ÇAĞRI YIĞINI

FFFAh: 5678h

FFFCh: 1234h

Kod İzleme: PUSH / POP ile Takas ve XCHG

```
1  org 100h
2  mov ax, 1234h
3  mov bx, 5678h
4  push ax          ; SP -= 2, [SP] = AX
5  push bx
▶  pop ax           ; AX = son giren
7  pop bx           ; BX = 1234h
8  xchg ah, al      ; baytları takas et
9  xchg ax, bx
10 ret
```

`pop ax` : üstteki sözcük okunur → **AX = 5678h**, sonra SP ← SP + 2 = FFFCh. Son giren ilk çıkar (LIFO).

Adım 5 / 9

DEĞİŞKENLER

AX = **5678h** BX = **5678h** SP = **FFFCh**

ÇAĞRI YIĞINI

FFFCh: 1234h

Kod İzleme: PUSH / POP ile Takas ve XCHG

```
1  org 100h
2  mov ax, 1234h
3  mov bx, 5678h
4  push ax          ; SP -= 2, [SP] = AX
5  push bx
6  pop ax           ; AX = son giren
▶  pop bx           ; BX = 1234h
8  xchg ah, al      ; baytları takas et
9  xchg ax, bx
10 ret
```

`pop bx` : **BX = 1234h**, SP = FFFEh. İki PUSH + ters sırada iki POP = takas.

Adım 6 / 9

DEĞİŞKENLER

AX = **5678h** BX = **1234h** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: PUSH / POP ile Takas ve XCHG

```
1  org 100h
2  mov ax, 1234h
3  mov bx, 5678h
4  push ax          ; SP -= 2, [SP] = AX
5  push bx
6  pop ax           ; AX = son giren
7  pop bx           ; BX = 1234h
▶ xchg ah, al       ; baytları takas et
9  xchg ax, bx
10 ret
```

`xchg ah, al` : AX'in iki baytı yer değiştirir: 5678h → **7856h**. Ek yazmaç gerekmez.

Adım 7 / 9

DEĞİŞKENLER

AX = **7856h** BX = **1234h** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: PUSH / POP ile Takas ve XCHG

```
1  org 100h
2  mov ax, 1234h
3  mov bx, 5678h
4  push ax          ; SP -= 2, [SP] = AX
5  push bx
6  pop ax           ; AX = son giren
7  pop bx           ; BX = 1234h
8  xchg ah, al      ; baytları takas et
▶  xchg ax, bx
10 ret
```

`xchg ax, bx` : AX ↔ BX. **AX = 1234h, BX = 7856h.** XCHG AX, r16 tek baytlık bir komuttur (90h+r).

Adım 8 / 9

DEĞİŞKENLER

AX = **1234h** BX = **7856h** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: PUSH / POP ile Takas ve XCHG

```
1 org 100h
2 mov ax, 1234h
3 mov bx, 5678h
4 push ax          ; SP -= 2, [SP] = AX
5 push bx
6 pop ax           ; AX = son giren
7 pop bx           ; BX = 1234h
8 xchg ah, al      ; baytları takas et
9 xchg ax, bx
▶ ret
```

Yığın dengeli (SP = FFFEh) olduğu için `ret` programı sonlandırır.

Adım 9 / 9

DEĞİŞKENLER

AX = **1234h** BX = **7856h** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

G/Ç Komutları: IN ve OUT

- 8086'nın bellekten ayrı **64 K'lık G/Ç adres alanı** vardır (M/$\overline{IO}$ ucu).
- Veri yalnızca **AL** (8 bit) ya da **AX** (16 bit) üzerinden aktarılır.
- **Sabit port**: 00h–FFh, komutun içinde.
- **Değişken port**: 0000h–FFFFh, **DX** ile.
- Bayrakları etkilemez.

```
in  al, 60h      ; port 60h'tan bayt oku
out 43h, al      ; port 43h'e yaz

mov dx, 3F8h     ; 255'ten büyük port
in  al, dx       ; DX ile adreslenir
out dx, ax       ; 16 bit yaz

in  bl, 60h      ; HATA: yalnız AL/AX
out 3F8h, al     ; HATA: DX gerekli
```

emu8086'daki sanal aygıtlar (trafik lambası, step motor, LED ekran) bu komutlarla sürülür; ayrıntılar Bölüm 9: Aygıt Kontrolü'nde.

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
▶  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 = 0115h. Bellek okunmaz.

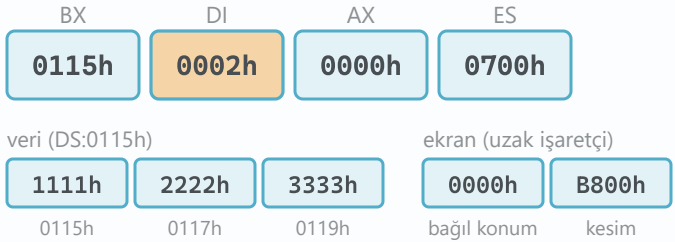
Adım 1 / 8

DEĞİŞKENLER

`BX = 0115h` `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 = 0115h | 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 = 0119h (adres)

LEA AX, [BX+DI+2]: yalnız etkin adres hesaplanır → AX = 0115h + 2 + 2 = **0119h**. Belleğe dokunulmaz, bayrak değişmez.

Adım 3 / 8

DEĞİŞKENLER

`BX = 0115h` `DI = 0002h` `AX = 0119h` `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 = 0115h | 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 = **0115h** | 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 = **0115h** | 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 = **0115h** | 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
```



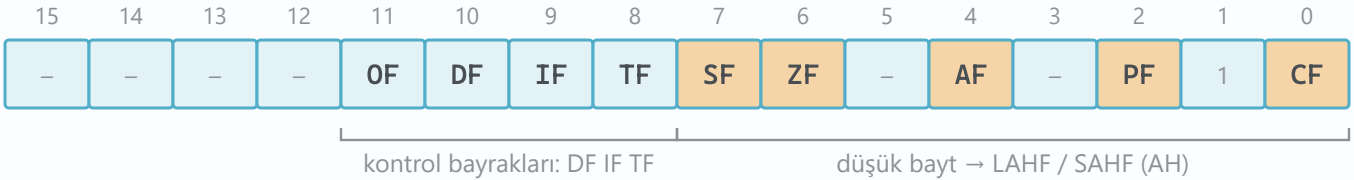
RET: program biter.

Adım 8 / 8

DEĞİŞKENLER

BX = **0115h** · DI = **0001h** · AX = **3341h** · ES = **B800h**

Bayrak Yazmacına Erişim



LAHF / SAHF

AH ↔ FLAGS'ın düşük baytı: **SF ZF – AF – PF – CF**. 8080 uyumluluğu için vardır; OF, DF, IF, TF'ye dokunmaz.

PUSHF / POPF

16 bitlik FLAGS'ın tamamı yığına atılır / geri alınır. Bir prosedürün bayrakları bozmaması ya da TF'yi değiştirmek için kullanılır.

```
pushf               ; bayrakları sakla
pop ax              ; AX = FLAGS
or  ax, 0100h        ; TF (bit 8) = 1 → tek adım kipi
push ax
popf                ; TF'yi doğrudan kuran bir komut yoktur
```

ARİTMETİK

Aritmetik Komutlar

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

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.

ARİTMETİK

Durum Bayrakları Ne Anlatır?

Bayrak	1 olur, eğer...	8 bit örnek
CF	en anlamlı bitten elde çıktı (toplama) ya da ödünç alındı (çıkarma) → işaretsiz taşma	`FFh + 01h = 100h`
OF	işaretli sonuç aralık dışı: iki pozitifin toplamı negatif ya da iki negatifin toplamı pozitif	`7Fh + 01h = 80h`
ZF	sonuç sıfır	`05h − 05h`
SF	sonucun en anlamlı biti 1 (işaretli yorumda negatif)	`05h − 07h = FEh`
PF	sonucun **düşük baytındaki** 1'lerin sayısı çift	`03h` (2 adet 1)
AF	bit 3'ten bit 4'e elde / ödünç (BCD düzeltmesi için)	`08h + 08h`

Kural: OF = (bit 6'dan bit 7'ye giren elde) XOR (bit 7'den çıkan elde). Toplamada işaretleri **farklı** iki sayı asla taşma üretmez.

ARİTMETİK

Bayrak Hesaplayıcı (8 bit)

A = 7F ADD B = 01 ☐ giriş CF = 1

A	0	1	1	1	1	1	1	1	7Fh = 127 / 127
+ B	0	0	0	0	0	0	0	1	01h = 1 / 1
sonuç	1	0	0	0	0	0	0	0	80h = 128 / -128

CF = 0 ZF = 0 **SF = 1** **OF = 1** PF = 0 **AF = 1**

İşaretsiz yorum: 127 + 1 → 128 · **OF = 1**: işaretli sonuç −128…127 dışında

İki onaltılık bayt girilir. Örnekler: 7F+01, FF+01, 80+80, 05−07, 38+45 (AF), 0F AND F0.

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1  org 100h
▶  mov al, 7Fh
3  add al, 1        ; +127 + 1
4  mov bl, 0FFh
5  add bl, 1        ; 255 + 1
6  mov cl, 5
7  sub cl, 7        ; 5 - 7
8  neg cl
9  cmp cl, 2
10 ret
```

AL = 7Fh = 0111 1111b: işaretli olarak en büyük pozitif bayt (+127).

Adım 1 / 9

DEĞİŞKENLER

AL = **7Fh** | BL = **00h** | CL = **00h** | CF = **0** | ZF = **0**

SF = **0** | OF = **0** | AF = **0** | PF = **0**

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1  org 100h
2  mov al, 7Fh
▶  add al, 1          ; +127 + 1
4  mov bl, 0FFh
5  add bl, 1          ; 255 + 1
6  mov cl, 5
7  sub cl, 7          ; 5 - 7
8  neg cl
9  cmp cl, 2
10 ret
```

7Fh + 1 = **80h**. İşaretsiz: 128 (doğru, **CF = 0**). İşaretli: +127 + 1 = −128 → **OF = 1** (taşma). SF = 1, bit 3'ten elde → **AF = 1**.

Adım 2 / 9

DEĞİŞKENLER

AL = 80h | BL = 00h | CL = 00h | CF = 0 | ZF = 0 | SF = 1 | OF = 1 | AF = 1 | PF = 0

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1  org 100h
2  mov al, 7Fh
3  add al, 1        ; +127 + 1
▶  mov bl, 0FFh
5  add bl, 1        ; 255 + 1
6  mov cl, 5
7  sub cl, 7        ; 5 - 7
8  neg cl
9  cmp cl, 2
10 ret
```

BL = FFh (işaretsiz 255, işaretli −1).

Adım 3 / 9

DEĞİŞKENLER

AL = **80h** | BL = **FFh** | CL = **00h** | CF = **0** | ZF = **0**

SF = **1** | OF = **1** | AF = **1** | PF = **0**

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1  org 100h
2  mov al, 7Fh
3  add al, 1          ; +127 + 1
4  mov bl, 0FFh
▶  add bl, 1          ; 255 + 1
6  mov cl, 5
7  sub cl, 7          ; 5 - 7
8  neg cl
9  cmp cl, 2
10 ret
```

FFh + 1 = 1 00h → **BL = 00h, CF = 1, ZF = 1**. İşaretli: −1 + 1 = 0, doğru → **OF = 0**.

Adım 4 / 9

DEĞİŞKENLER

AL = **80h** | BL = **00h** | CL = **00h** | CF = **1** | ZF = **1** | SF = **0** | OF = **0** | AF = **1** | PF = **1**

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1  org 100h
2  mov al, 7Fh
3  add al, 1        ; +127 + 1
4  mov bl, 0FFh
5  add bl, 1        ; 255 + 1
▶  mov cl, 5
7  sub cl, 7        ; 5 - 7
8  neg cl
9  cmp cl, 2
10 ret
```

CL = 05h.

Adım 5 / 9

DEĞİŞKENLER

AL = **80h** | BL = **00h** | CL = **05h** | CF = **1** | ZF = **1**

SF = **0** | OF = **0** | AF = **1** | PF = **1**

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1  org 100h
2  mov al, 7Fh
3  add al, 1          ; +127 + 1
4  mov bl, 0FFh
5  add bl, 1          ; 255 + 1
6  mov cl, 5
▶  sub cl, 7          ; 5 - 7
8  neg cl
9  cmp cl, 2
10 ret
```

5 − 7 = −2 = **FEh**. Ödünç alındı → **CF = 1**; SF = 1. İşaretli sonuç doğru → OF = 0.

Adım 6 / 9

DEĞİŞKENLER

AL = **80h** BL = **00h** CL = **FEh** CF = **1** ZF = **0**

SF = **1** OF = **0** AF = **1** PF = **0**

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1  org 100h
2  mov al, 7Fh
3  add al, 1          ; +127 + 1
4  mov bl, 0FFh
5  add bl, 1          ; 255 + 1
6  mov cl, 5
7  sub cl, 7          ; 5 - 7
▶  neg cl
9  cmp cl, 2
10 ret
```

`neg cl` : 0 − FEh = **02h**. İşlenen sıfır olmadığından **CF = 1**.

Adım 7 / 9

DEĞİŞKENLER

AL = **80h** | BL = **00h** | CL = **02h** | CF = **1** | ZF = **0**

SF = **0** | OF = **0** | AF = **1** | PF = **0**

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1  org 100h
2  mov al, 7Fh
3  add al, 1          ; +127 + 1
4  mov bl, 0FFh
5  add bl, 1          ; 255 + 1
6  mov cl, 5
7  sub cl, 7          ; 5 - 7
8  neg cl
▶ cmp cl, 2
10 ret
```

`cmp cl, 2` : 2 − 2 hesaplanır ama sonuç yazılmaz. **ZF = 1** → `je` atlar.

Adım 8 / 9

DEĞİŞKENLER

AL = **80h** BL = **00h** CL = **02h** CF = **0** ZF = **1**

SF = **0** OF = **0** AF = **0** PF = **1**

Kod İzleme: ADD, SUB, NEG, CMP ve Bayraklar

```
1 org 100h
2 mov al, 7Fh
3 add al, 1         ; +127 + 1
4 mov bl, 0FFh
5 add bl, 1         ; 255 + 1
6 mov cl, 5
7 sub cl, 7         ; 5 - 7
8 neg cl
9 cmp cl, 2
▶ ret
```

CMP yalnızca bayrakları değiştirir; CL hâlâ 02h.

Adım 9 / 9

DEĞİŞKENLER

AL = **80h** BL = **00h** CL = **02h** CF = **0** ZF = **1**

SF = **0** OF = **0** AF = **0** PF = **1**

Kod İzleme: ADC ile 32 Bitlik Toplama

```
1 org 100h
▶ mov ax, 0F00Fh      ; DX:AX = 1234F00Fh
3 mov dx, 1234h
4 mov bx, 2001h       ; CX:BX = 00012001h
5 mov cx, 0001h
6 add ax, bx          ; düşük sözcükler
7 adc dx, cx          ; yüksek + CF
8 inc ax              ; CF etkilenmez
9 ret
```

Birinci 32 bitlik sayının düşük sözcüğü: **AX = F00Fh**.

Adım 1 / 8

DEĞİŞKENLER

AX = **F00Fh** DX = **0000h** BX = **0000h**

CX = **0000h** CF = **0** ZF = **0** SF = **0** OF = **0**

Kod İzleme: ADC ile 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0F00Fh      ; DX:AX = 1234F00Fh
▶ mov dx, 1234h
4 mov bx, 2001h       ; CX:BX = 00012001h
5 mov cx, 0001h
6 add ax, bx          ; düşük sözcükler
7 adc dx, cx          ; yüksek + CF
8 inc ax              ; CF etkilenmez
9 ret
```

Yüksek sözcük: **DX = 1234h**. Birlikte DX:AX = 1234F00Fh.

Adım 2 / 8

DEĞİŞKENLER

AX = **F00Fh** | DX = **1234h** | BX = **0000h**

CX = **0000h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0**

Kod İzleme: ADC ile 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0F00Fh      ; DX:AX = 1234F00Fh
3 mov dx, 1234h
▶ mov bx, 2001h       ; CX:BX = 00012001h
5 mov cx, 0001h
6 add ax, bx          ; düşük sözcükler
7 adc dx, cx          ; yüksek + CF
8 inc ax              ; CF etkilenmez
9 ret
```

İkinci sayının düşük sözcüğü **BX = 2001h**.

Adım 3 / 8

DEĞİŞKENLER

AX = **F00Fh** | DX = **1234h** | BX = **2001h**

CX = **0000h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0**

Kod İzleme: ADC ile 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0F00Fh      ; DX:AX = 1234F00Fh
3 mov dx, 1234h
4 mov bx, 2001h       ; CX:BX = 00012001h
▶ mov cx, 0001h
6 add ax, bx          ; düşük sözcükler
7 adc dx, cx          ; yüksek + CF
8 inc ax              ; CF etkilenmez
9 ret
```

Yüksek sözcük **CX = 0001h** → CX:BX = 00012001h.

Adım 4 / 8

DEĞİŞKENLER

AX = **F00Fh** | DX = **1234h** | BX = **2001h**

CX = **0001h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0**

Kod İzleme: ADC ile 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0F00Fh      ; DX:AX = 1234F00Fh
3 mov dx, 1234h
4 mov bx, 2001h       ; CX:BX = 00012001h
5 mov cx, 0001h
▶ add ax, bx          ; düşük sözcükler
7 adc dx, cx          ; yüksek + CF
8 inc ax              ; CF etkilenmez
9 ret
```

`add ax, bx` : F00Fh + 2001h = 1 1010h. 16 bite sığmaz → **AX = 1010h, CF = 1** (elde).

Adım 5 / 8

DEĞİŞKENLER

AX = **1010h** | DX = **1234h** | BX = **2001h** | CX = **0001h** | CF = **1** | ZF = **0** | SF = **0** | OF = **0**

Kod İzleme: ADC ile 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0F00Fh      ; DX:AX = 1234F00Fh
3 mov dx, 1234h
4 mov bx, 2001h       ; CX:BX = 00012001h
5 mov cx, 0001h
6 add ax, bx          ; düşük sözcükler
▶ adc dx, cx          ; yüksek + CF
8 inc ax              ; CF etkilenmez
9 ret
```

`adc dx, cx` : 1234h + 0001h + **CF** = **1236h**. Sonuç DX:AX = **12361010h**.

Adım 6 / 8

DEĞİŞKENLER

AX = **1010h** | DX = **1236h** | BX = **2001h** | CX = **0001h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0**

Kod İzleme: ADC ile 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0F00Fh      ; DX:AX = 1234F00Fh
3 mov dx, 1234h
4 mov bx, 2001h       ; CX:BX = 00012001h
5 mov cx, 0001h
6 add ax, bx          ; düşük sözcükler
7 adc dx, cx          ; yüksek + CF
▶ inc ax              ; CF etkilenmez
9 ret
```

`inc ax` : AX = 1011h. INC/DEC **CF'yi değiştirmez**; döngü sayaçlarında elde bilgisini korur.

Adım 7 / 8

DEĞİŞKENLER

AX = **1011h** | DX = **1236h** | BX = **2001h** | CX = **0001h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0**

Kod İzleme: ADC ile 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0F00Fh      ; DX:AX = 1234F00Fh
3 mov dx, 1234h
4 mov bx, 2001h       ; CX:BX = 00012001h
5 mov cx, 0001h
6 add ax, bx          ; düşük sözcükler
7 adc dx, cx          ; yüksek + CF
8 inc ax              ; CF etkilenmez
▶ ret
```

Bitti: 1234F00Fh + 00012001h = 12361010h, sonra +1.

Adım 8 / 8

DEĞİŞKENLER

AX = **1011h** DX = **1236h** BX = **2001h**

CX = **0001h** CF = **0** ZF = **0** SF = **0** OF = **0**

ARİTMETİK

Çıkarma Ailesi

Komut	İşlem	Not
`sub h, k`	h ← h − k	CF = ödünç
`sbb h, k`	h ← h − k − CF	çok sözcüklü çıkarma
`dec h`	h ← h − 1	CF değişmez
`neg h`	h ← 0 − h	ikiye tümleyen; h ≠ 0 ise CF = 1
`cmp h, k`	h − k (yazılmaz)	yalnızca bayraklar

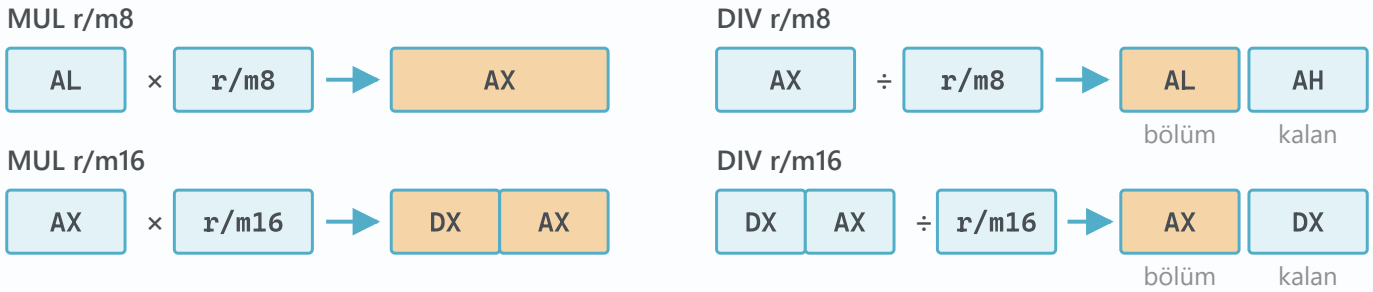
```
; 32 bit: DX:AX ← DX:AX − CX:BX
sub ax, bx     ; düşük sözcük
sbb dx, cx     ; yüksek − ödünç

; mutlak değer
cmp ax, 0
jge pozitif
neg ax         ; −5 → 5
pozitif:
```

`cmp a, b` sonrası: a = b ⇒ ZF = 1; işaretsiz a < b ⇒ CF = 1; işaretli a < b ⇒ SF ≠ OF. Koşullu atlamalar bu bayrakları okur.

NEG −128'de (80h) sonuç yine 80h olur ve OF = 1: bu değerin 8 bitte pozitif karşılığı yoktur.

Çarpma ve Bölme: Örtük Yazmaçlar



MUL / IMUL kaynak

Tek işlenen alır (yazmaç ya da bellek, **anlık olamaz**). Sonuç her zaman çift genişlikte. Üst yarı sıfır değilse (IMUL'da işaret uzantısı değilse) **CF = OF = 1**.

DIV / IDIV kaynak

Bölüm hedefe sığmazsa ya da bölen 0 ise işlemci **INT 0** (bölme hatası) üretir. IDIV'de kalanın işareti bölünenin işaretidir.

```
mov al, 80h   mov bl, 4   mul bl     ; AX = 0200h, CF = OF = 1 (AH ≠ 0)
mov al, -2    mov bl, 3   imul bl    ; AX = FFFAh (−6), CF = OF = 0
mov ax, 1000h mov bl, 2   div bl     ; bölüm 800h > FFh → bölme hatası!
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
 1 org 100h
 2 mov ax, 305
 3 mov bx, 10
 4 xor cx, cx        ; basamak sayacı
 5 bol:
 6 xor dx, dx        ; DX:AX bölünen
 7 div bx            ; AX=bölüm DX=kalan
 8 push dx
 9 inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

Yazdırılacak sayı: 305 = **0131h**.

Adım 1 / 37

DEĞİŞKENLER

AX = **0131h** BX = **0000h** CX = **0000h**
DX = **0000h** SP = **FFFEh** ZF = **0**

ÇAĞRI YIĞINI

boş

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
▶  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

Bölen BX = 10 (000Ah).

Adım 2 / 37

DEĞİŞKENLER

AX = **0131h** | BX = **000Ah** | CX = **0000h**

DX = **0000h** | SP = **FFFEh** | ZF = **0**

ÇAĞRI YIĞINI

boş

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

`xor cx, cx` : CX = 0. Kendisiyle XOR en kısa sıfırlama yoludur.

Adım 3 / 37

DEĞİŞKENLER

AX = **0131h** BX = **000Ah** CX = **0000h**
DX = **0000h** SP = **FFFEh** ZF = **1**

ÇAĞRI YIĞINI

boş

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
▶  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

16 bitlik DIV, bölünen olarak **DX:AX**'i kullanır; DX mutlaka sıfırlanmalı.

Adım 4 / 37

DEĞİŞKENLER

AX = **0131h** BX = **000Ah** CX = **0000h**
DX = **0000h** SP = **FFFEh** ZF = **1**

ÇAĞRI YIĞINI

boş

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
▶  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

305 ÷ 10 → **AX = 30 (001Eh)**, kalan **DX = 5** (birler basamağı).

Adım 5 / 37

DEĞİŞKENLER

AX = 001Eh | BX = 000Ah | CX = 0000h | DX = 0005h | SP = FFFEh | ZF = 1

ÇAĞRI YIĞINI

boş

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
▶  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

Kalan yığına atılır: basamaklar **ters sırada** üretilir, yığın sırayı düzeltir.

Adım 6 / 37

DEĞİŞKENLER

AX = **001Eh** BX = **000Ah** CX = **0000h**
DX = **0005h** SP = **FFFCh** ZF = **1**

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
▶  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

Basamak sayacı CX = 0001h.

Adım 7 / 37

DEĞİŞKENLER

AX = 001Eh | BX = 000Ah | CX = 0001h | DX = 0005h | SP = FFFCh | ZF = 0

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
▶ test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

`test ax, ax` : AX = 001Eh → ZF = 0.

Adım 8 / 37

DEĞİŞKENLER

AX = **001Eh** BX = **000Ah** CX = **0001h**
DX = **0005h** SP = **FFFCh** ZF = **0**

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
▶ 11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

AX ≠ 0 → `bol` 'e dön.

Adım 9 / 37

DEĞİŞKENLER

AX = **001Eh** | BX = **000Ah** | CX = **0001h**

DX = **0005h** | SP = **FFFCh** | ZF = **0**

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
▶ 6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

16 bitlik DIV, bölünen olarak **DX:AX**'i kullanır; DX mutlaka sıfırlanmalı.

Adım 10 / 37

DEĞİŞKENLER

AX = 001Eh | BX = 000Ah | CX = 0001h | DX = 0000h | SP = FFFCh | ZF = 1

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
▶  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

30 ÷ 10 → **AX = 3**, kalan **DX = 0** (onlar basamağı).

Adım 11 / 37

DEĞİŞKENLER

AX = 0003h | BX = 000Ah | CX = 0001h | DX = 0000h | SP = FFFCh | ZF = 1

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
▶  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

0 yığına atılır.

Adım 12 / 37

DEĞİŞKENLER

AX = 0003h | BX = 000Ah | CX = 0001h

DX = 0000h | SP = FFFAh | ZF = 1

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
▶  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

Basamak sayacı CX = 0002h.

Adım 13 / 37

DEĞİŞKENLER

AX = 0003h | BX = 000Ah | CX = 0002h | DX = 0000h | SP = FFFAh | ZF = 0

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
▶  test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

`test ax, ax` : AX = 0003h → ZF = 0.

Adım 14 / 37

DEĞİŞKENLER

AX = **0003h** | BX = **000Ah** | CX = **0002h**

DX = **0000h** | SP = **FFFAh** | ZF = **0**

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
 1 org 100h
 2 mov ax, 305
 3 mov bx, 10
 4 xor cx, cx        ; basamak sayacı
 5 bol:
 6 xor dx, dx        ; DX:AX bölünen
 7 div bx            ; AX=bölüm DX=kalan
 8 push dx
 9 inc cx
10 test ax, ax
▶ jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

AX ≠ 0 → bir tur daha.

Adım 15 / 37

DEĞİŞKENLER

AX = 0003h | BX = 000Ah | CX = 0002h | DX = 0000h | SP = FFFAh | ZF = 0

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

16 bitlik DIV, bölünen olarak **DX:AX**'i kullanır; DX mutlaka sıfırlanmalı.

Adım 16 / 37

DEĞİŞKENLER

AX = 0003h | BX = 000Ah | CX = 0002h | DX = 0000h | SP = FFFAh | ZF = 1

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
▶  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

3 ÷ 10 → **AX = 0**, kalan **DX = 3** (yüzler basamağı).

Adım 17 / 37

DEĞİŞKENLER

AX = **0000h** BX = **000Ah** CX = **0002h**
DX = **0003h** SP = **FFFAh** ZF = **1**

ÇAĞRI YIĞINI

FFFAh: 0000h
FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
▶  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

3 yığına atılır; üstte artık en anlamlı basamak var.

Adım 18 / 37

DEĞİŞKENLER

AX = 0000h | BX = 000Ah | CX = 0002h | DX = 0003h | SP = FFF8h | ZF = 1

ÇAĞRI YIĞINI

FFF8h: 0003h
FFFAh: 0000h
FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
▶  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

Basamak sayacı CX = 0003h.

Adım 19 / 37

DEĞİŞKENLER

AX = 0000h | BX = 000Ah | CX = 0003h | DX = 0003h | SP = FFF8h | ZF = 0

ÇAĞRI YIĞINI

FFF8h: 0003h
FFFAh: 0000h
FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
▶ test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

`test ax, ax` : AX = 0000h → ZF = 1.

Adım 20 / 37

DEĞİŞKENLER

AX = **0000h** | BX = **000Ah** | CX = **0003h**

DX = **0003h** | SP = **FFF8h** | ZF = **1**

ÇAĞRI YIĞINI

FFF8h: 0003h

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
▶ 11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

AX = 0 → döngüden çık. CX = 3 basamak.

Adım 21 / 37

DEĞİŞKENLER

AX = **0000h** | BX = **000Ah** | CX = **0003h**

DX = **0003h** | SP = **FFF8h** | ZF = **1**

ÇAĞRI YIĞINI

FFF8h: 0003h

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
▶ pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

`pop dx` : üstteki basamak DX = 0003h.

Adım 22 / 37

DEĞİŞKENLER

AX = **0000h** | BX = **000Ah** | CX = **0003h**

DX = **0003h** | SP = **FFFAh** | ZF = **1**

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
▶ add dl, '0'        ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

'0' (30h) eklenerek rakam ASCII'ye çevrilir: DL = 33h.

Adım 23 / 37

DEĞİŞKENLER

AX = 0000h | BX = 000Ah | CX = 0003h | DX = 0033h | SP = FFFAh | ZF = 0

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
 1 org 100h
 2 mov ax, 305
 3 mov bx, 10
 4 xor cx, cx        ; basamak sayacı
 5 bol:
 6 xor dx, dx        ; DX:AX bölünen
 7 div bx            ; AX=bölüm DX=kalan
 8 push dx
 9 inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
▶ mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

AH = 02h (karakter yaz).

Adım 24 / 37

DEĞİŞKENLER

AX = 0200h | BX = 000Ah | CX = 0003h | DX = 0033h | SP = FFFAh | ZF = 0

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
▶ int 21h
17 loop yaz
18 ret
```

Rakam ekrana yazılır.

Adım 25 / 37

DEĞİŞKENLER

AX = 0233h | BX = 000Ah | CX = 0003h | DX = 0033h | SP = FFFAh | ZF = 0

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

```
3
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
▶  loop yaz
18 ret
```

`loop yaz` : CX = 0002h, sıfır değil → tekrar.

Adım 26 / 37

DEĞİŞKENLER

AX = **0233h** BX = **000Ah** CX = **0002h**

DX = **0033h** SP = **FFFAh** ZF = **0**

ÇAĞRI YIĞINI

FFFAh: 0000h

FFFCh: 0005h

ÇIKTI

```
3
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
▶ pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

`pop dx` : üstteki basamak DX = 0000h.

Adım 27 / 37

DEĞİŞKENLER

AX = 0233h | BX = 000Ah | CX = 0002h | DX = 0000h | SP = FFFCh | ZF = 0

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

```
3
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
▶ add dl, '0'        ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

'0' (30h) eklenerek rakam ASCII'ye çevrilir: DL = 30h.

Adım 28 / 37

DEĞİŞKENLER

AX = 0233h | BX = 000Ah | CX = 0002h | DX = 0030h | SP = FFFCh | ZF = 0

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

```
3
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
▶  mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

AH = 02h (karakter yaz).

Adım 29 / 37

DEĞİŞKENLER

AX = **0233h** | BX = **000Ah** | CX = **0002h**

DX = **0030h** | SP = **FFFCh** | ZF = **0**

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

```
3
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
▶ int 21h
17 loop yaz
18 ret
```

Rakam ekrana yazılır.

Adım 30 / 37

DEĞİŞKENLER

AX = 0230h | BX = 000Ah | CX = 0002h | DX = 0030h | SP = FFFCh | ZF = 0

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

```
30
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
▶  loop yaz
18 ret
```

CX = 0001h → tekrar.

Adım 31 / 37

DEĞİŞKENLER

AX = **0230h** | BX = **000Ah** | CX = **0001h** | DX = **0030h** | SP = **FFFCh** | ZF = **0**

ÇAĞRI YIĞINI

FFFCh: 0005h

ÇIKTI

```
30
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
▶ pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

`pop dx` : üstteki basamak DX = 0005h.

Adım 32 / 37

DEĞİŞKENLER

AX = **0230h** BX = **000Ah** CX = **0001h**
DX = **0005h** SP = **FFFEh** ZF = **0**

ÇAĞRI YIĞINI

boş

ÇIKTI

```
30
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
▶ add dl, '0'        ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

'0' (30h) eklenerek rakam ASCII'ye çevrilir: DL = 35h.

Adım 33 / 37

DEĞİŞKENLER

AX = **0230h** | BX = **000Ah** | CX = **0001h**

DX = **0035h** | SP = **FFFEh** | ZF = **0**

ÇAĞRI YIĞINI

boş

ÇIKTI

```
30
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
▶  mov ah, 2
16 int 21h
17 loop yaz
18 ret
```

AH = 02h (karakter yaz).

Adım 34 / 37

DEĞİŞKENLER

AX = **0230h** BX = **000Ah** CX = **0001h**

DX = **0035h** SP = **FFFEh** ZF = **0**

ÇAĞRI YIĞINI

boş

ÇIKTI

```
30
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
▶ int 21h
17 loop yaz
18 ret
```

Rakam ekrana yazılır.

Adım 35 / 37

DEĞİŞKENLER

AX = 0235h | BX = 000Ah | CX = 0001h | DX = 0035h | SP = FFFEh | ZF = 0

ÇAĞRI YIĞINI

boş

ÇIKTI

```
305
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
▶ loop yaz
18 ret
```

CX = 0 → döngü biter. Ekranda **305**.

Adım 36 / 37

DEĞİŞKENLER

AX = **0235h** BX = **000Ah** CX = **0000h**

DX = **0035h** SP = **FFFEh** ZF = **0**

ÇAĞRI YIĞINI

boş

ÇIKTI

```
305
```

Kod İzleme: DIV ile Sayıyı Onluk Yazdırma

```
1  org 100h
2  mov ax, 305
3  mov bx, 10
4  xor cx, cx        ; basamak sayacı
5  bol:
6  xor dx, dx        ; DX:AX bölünen
7  div bx            ; AX=bölüm DX=kalan
8  push dx
9  inc cx
10 test ax, ax
11 jnz bol
12 yaz:
13 pop dx
14 add dl, '0'       ; rakam → ASCII
15 mov ah, 2
16 int 21h
17 loop yaz
▶  ret
```

Bölme–kalan yöntemi her tabana uygulanabilir (bölen 2 → ikili, 16 → onaltılık).

Adım 37 / 37

DEĞİŞKENLER

AX = **0235h** BX = **000Ah** CX = **0000h**

DX = **0035h** SP = **FFFEh** ZF = **0**

ÇAĞRI YIĞINI

boş

ÇIKTI

```
305
```

ARİTMETİK

İşaret Genişletme: CBW ve CWD

- `cbw` : AL'nin işaret bitini AH'ye kopyalar (AL → AX).
- `cwd` : AX'in işaret bitini DX'e kopyalar (AX → DX:AX).
- İşaretli bölmeden (IDIV) önce bölünen bu komutlarla genişletilir; işaretsiz bölmede ise üst yarı **sıfırlanır** (`xor dx, dx` / `mov ah, 0`).
- Bayrakları etkilemez.

Önce	Komut	Sonra
AL = 05h	`cbw`	AX = 0005h
AL = F6h (−10)	`cbw`	AX = FFF6h
AX = FFF6h	`cwd`	DX = FFFFh

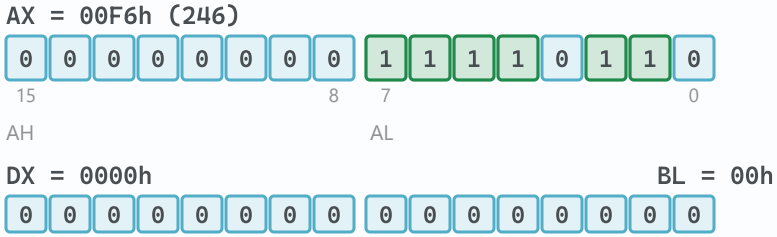
```
; -10 ÷ 3 (işaretli)
mov al, -10     ; AL = F6h
cbw             ; AX = FFF6h
mov bl, 3
idiv bl         ; AL = FDh (-3)
                ; AH = FFh (-1)

; yanlış: mov ah, 0
; AX = 00F6h = +246
; idiv bl → AL = 52h (82)
```

IDIV bölümü sıfıra doğru keser: −10 / 3 = −3, kalan −1 (−3 × 3 − 1 = −10).

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
▶  mov al, -10     ; AL = F6h
3  cbw             ; AX = FFF6h
4  mov bl, 3
5  idiv bl         ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0       ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd             ; DX:AX
11 mov cx, 7
12 idiv cx         ; AX:bölüm DX:kalan
13 ret
```



AL = −10 = **F6h** = 1111 0110b. İşaret biti (bit 7) 1.

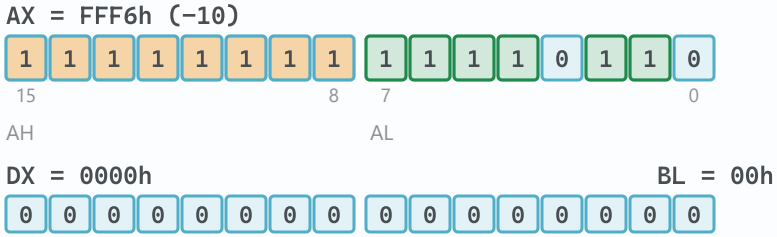
Adım 1 / 12

DEĞİŞKENLER

AX = **00F6h** DX = **0000h** BL = **00h** CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10      ; AL = F6h
▶  cbw              ; AX = FFF6h
4  mov bl, 3
5  idiv bl          ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0        ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd              ; DX:AX
11 mov cx, 7
12 idiv cx          ; AX:bölüm DX:kalan
13 ret
```



CBW: AL'nin bit 7'si AH'nin 8 bitine kopyalanır → AX = **FFF6h** = −10 (16 bit).

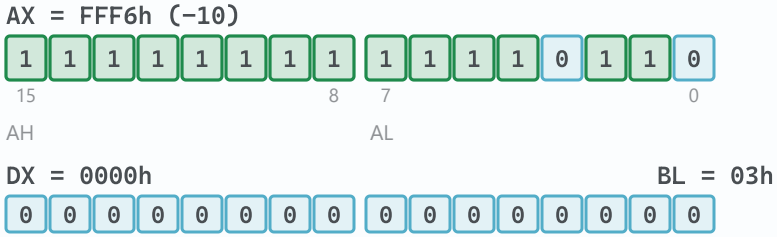
Adım 2 / 12

DEĞİŞKENLER

AX = **FFF6h** | DX = **0000h** | BL = **00h** | CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10      ; AL = F6h
3  cbw              ; AX = FFF6h
▶  mov bl, 3
5  idiv bl          ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0        ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd              ; DX:AX
11 mov cx, 7
12 idiv cx          ; AX:bölüm DX:kalan
13 ret
```



BL = 3: bölen.

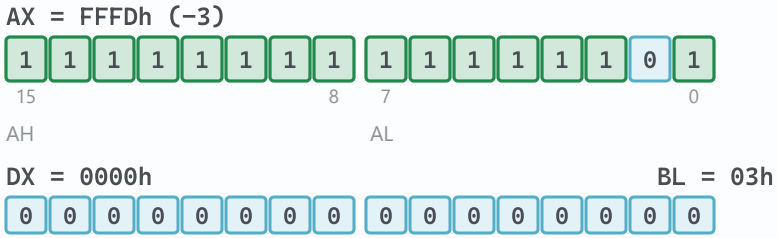
Adım 3 / 12

DEĞİŞKENLER

AX = **FFF6h** | DX = **0000h** | BL = **03h** | CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10      ; AL = F6h
3  cbw              ; AX = FFF6h
4  mov bl, 3
▶  idiv bl          ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0        ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd              ; DX:AX
11 mov cx, 7
12 idiv cx          ; AX:bölüm DX:kalan
13 ret
```



IDIV BL: −10 ÷ 3 → bölüm AL = **FDh** (−3), kalan AH = **FFh** (−1). Sıfıra doğru keser; kalanın işareti bölünenin işareti.

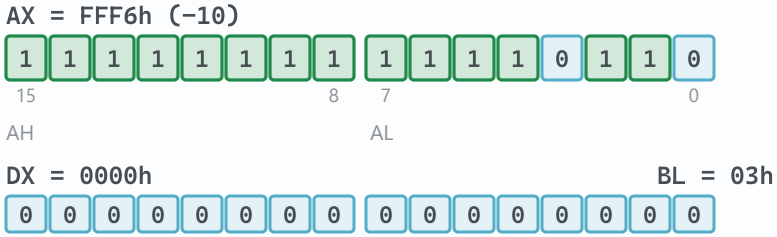
Adım 4 / 12

DEĞİŞKENLER

AX = **FFFDh** | DX = **0000h** | BL = **03h** | CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10     ; AL = F6h
3  cbw             ; AX = FFF6h
4  mov bl, 3
5  idiv bl         ; AL:bölüm AH:kalan
▶  mov al, -10
7  mov ah, 0       ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd             ; DX:AX
11 mov cx, 7
12 idiv cx         ; AX:bölüm DX:kalan
13 ret
```



Yeniden AL = F6h.

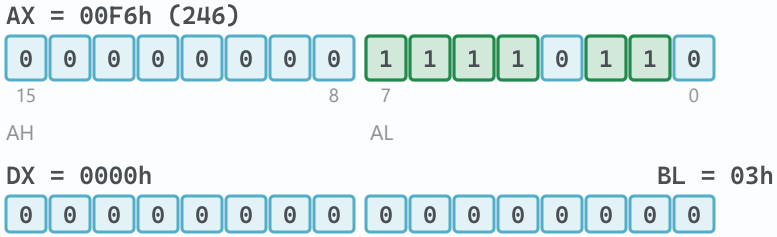
Adım 5 / 12

DEĞİŞKENLER

AX = **FFF6h** | DX = **0000h** | BL = **03h** | CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10     ; AL = F6h
3  cbw             ; AX = FFF6h
4  mov bl, 3
5  idiv bl         ; AL:bölüm AH:kalan
6  mov al, -10
▶  mov ah, 0       ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd             ; DX:AX
11 mov cx, 7
12 idiv cx         ; AX:bölüm DX:kalan
13 ret
```



Yanlış: AH = 0 → AX = **00F6h** = +246. İşaret kayboldu.

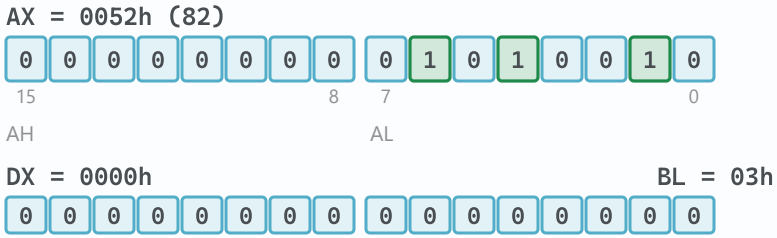
Adım 6 / 12

DEĞİŞKENLER

AX = **00F6h** | DX = **0000h** | BL = **03h** | CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10      ; AL = F6h
3  cbw              ; AX = FFF6h
4  mov bl, 3
5  idiv bl          ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0        ; yanlış genişletme
▶  idiv bl
9  mov ax, -1000
10 cwd              ; DX:AX
11 mov cx, 7
12 idiv cx          ; AX:bölüm DX:kalan
13 ret
```



IDIV BL: 246 ÷ 3 = **82** (52h) kalan 0 → AL = 52h. Yanlış sonuç.

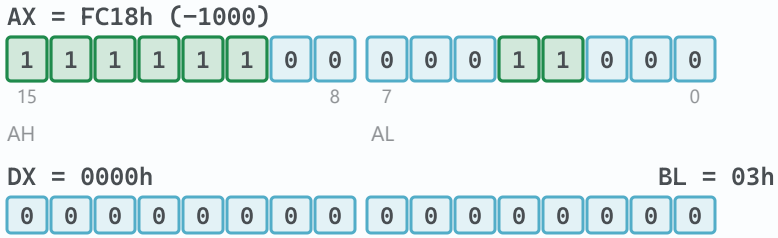
Adım 7 / 12

DEĞİŞKENLER

AX = **0052h** | DX = **0000h** | BL = **03h** | CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10     ; AL = F6h
3  cbw             ; AX = FFF6h
4  mov bl, 3
5  idiv bl         ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0       ; yanlış genişletme
8  idiv bl
▶  mov ax, -1000
10 cwd             ; DX:AX
11 mov cx, 7
12 idiv cx         ; AX:bölüm DX:kalan
13 ret
```



AX = −1000 = **FC18h**.

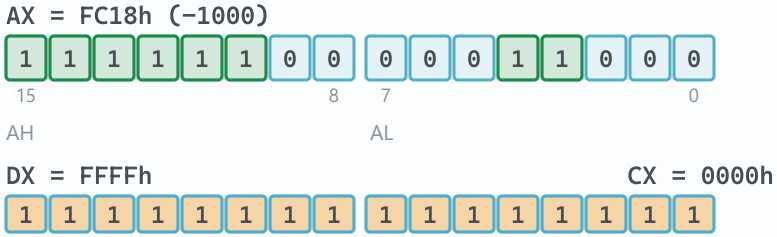
Adım 8 / 12

DEĞİŞKENLER

AX = **FC18h** | DX = **0000h** | BL = **03h** | CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10     ; AL = F6h
3  cbw             ; AX = FFF6h
4  mov bl, 3
5  idiv bl         ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0       ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
▶  cwd             ; DX:AX
11 mov cx, 7
12 idiv cx         ; AX:bölüm DX:kalan
13 ret
```



CWD: AX'in bit 15'i DX'e kopyalanır → DX:AX = **FFFF FC18h**.

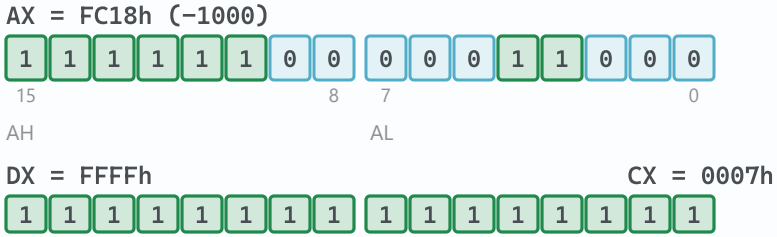
Adım 9 / 12

DEĞİŞKENLER

AX = **FC18h** | DX = **FFFFh** | BL = **03h** | CX = **0000h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10     ; AL = F6h
3  cbw             ; AX = FFF6h
4  mov bl, 3
5  idiv bl         ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0       ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd             ; DX:AX
▶  mov cx, 7
12 idiv cx         ; AX:bölüm DX:kalan
13 ret
```



CX = 7.

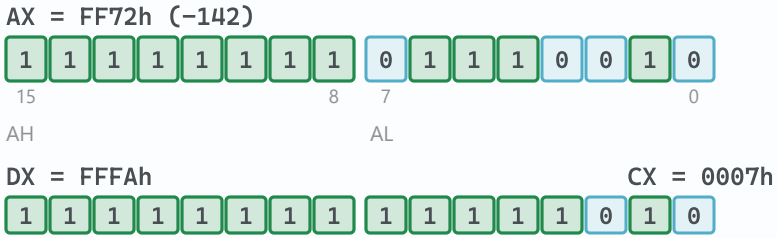
Adım 10 / 12

DEĞİŞKENLER

AX = **FC18h** | DX = **FFFFh** | BL = **03h** | CX = **0007h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10      ; AL = F6h
3  cbw              ; AX = FFF6h
4  mov bl, 3
5  idiv bl          ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0        ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd              ; DX:AX
11 mov cx, 7
▶ idiv cx           ; AX:bölüm DX:kalan
13 ret
```



IDIV CX: −1000 ÷ 7 → AX = **FF72h** (−142), DX = **FFFAh** (−6). Doğrulama: −142 × 7 − 6 = −1000.

Adım 11 / 12

DEĞİŞKENLER

AX = **FF72h** | DX = **FFFAh** | BL = **03h** | CX = **0007h**

İşaret Genişletme ile Doğru Bölme

```
1  org 100h
2  mov al, -10     ; AL = F6h
3  cbw             ; AX = FFF6h
4  mov bl, 3
5  idiv bl         ; AL:bölüm AH:kalan
6  mov al, -10
7  mov ah, 0       ; yanlış genişletme
8  idiv bl
9  mov ax, -1000
10 cwd             ; DX:AX
11 mov cx, 7
12 idiv cx         ; AX:bölüm DX:kalan
▶  ret
```

AX = FF72h (−142)



DX = FFFAh

CX = 0007h



RET: program biter.

Adım 12 / 12

DEĞİŞKENLER

AX = **FF72h** DX = **FFFAh** BL = **03h** CX = **0007h**

ARİTMETİK

BCD Düzeltme Komutları

Komut	Ne zaman?	Örnek
`DAA`	paketli BCD toplamadan sonra	38h + 45h = 7Dh → **83h**
`DAS`	paketli BCD çıkarmadan sonra	83h − 38h = 4Bh → **45h**
`AAA`	paketsiz/ASCII toplamadan sonra	'8' + '5' = 6Dh → AX = **0103h** (AH+1)
`AAS`	paketsiz/ASCII çıkarmadan sonra	'3' − '5' = FEh → AL = **08h**, AH−1, CF = 1
`AAM`	paketsiz çarpmadan sonra	7 × 8 = 38h → AH = **05**, AL = **06**
`AAD`	paketsiz bölmeden **önce**	AX = 0506h → AL = **38h** (56), AH = 0

Paketli BCD

Bir baytta iki onluk basamak: 45 → `45h` (0100 0101).

Paketsiz BCD

Bir baytta bir basamak: 45 → `04h 05h`; ASCII'de `34h 35h`.

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
▶  mov al, 38h      ; BCD 38
3  add al, 45h      ; + BCD 45
4  daa              ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47       ; ikili 2Fh
9  aam              ; AH=onlar AL=birler
10 add ax, 3030h    ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

Paketli BCD: 38h, onluk **38** anlamına gelir.

Adım 1 / 16

DEĞİŞKENLER

AL = **38h** AH = **00h** BX = **0000h** CF = **0** AF = **0**

ÇIKTI



Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
▶  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

İkili toplama: 38h + 45h = **7Dh**. Düşük nibble Dh > 9 → geçerli BCD değil.

Adım 2 / 16

DEĞİŞKENLER

AL = **7Dh** | AH = **00h** | BX = **0000h** | CF = **0** | AF = **0**

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
▶  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

`daa` : düşük nibble > 9 → +06h: 7Dh + 6 = **83h**. 38 + 45 = **83** ✓.

Adım 3 / 16

DEĞİŞKENLER

AL = **83h** AH = **00h** BX = **0000h** CF = **0** AF = **1**

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
▶  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

Yeni örnek: BCD 79.

Adım 4 / 16

DEĞİŞKENLER

AL = 79h | AH = 00h | BX = 0000h | CF = 0 | AF = 1

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
▶  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

79h + 35h = **AEh**. Bit 3'ten elde yok → AF = 0.

Adım 5 / 16

DEĞİŞKENLER

AL = **AEh** | AH = **00h** | BX = **0000h** | CF = **0** | AF = **0**

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
▶  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

`daa`: Eh > 9 → +06h (B4h); sonra AL > 9Fh → +60h → **14h, CF = 1**. 79 + 35 = **1**14 ✓.

Adım 6 / 16

DEĞİŞKENLER

AL = **14h** | AH = **00h** | BX = **0000h** | CF = **1** | AF = **1**

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h      ; BCD 38
3  add al, 45h      ; + BCD 45
4  daa              ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
▶ mov al, 47        ; ikili 2Fh
9  aam              ; AH=onlar AL=birler
10 add ax, 3030h    ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

AAM ikili sayıyla çalışır: AL = 47 = **2Fh**.

Adım 7 / 16

DEĞİŞKENLER

AL = **2Fh** | AH = **00h** | BX = **0000h** | CF = **1** | AF = **1**

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
▶ aam                ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

`aam` : AH = AL ÷ 10 = **4**, AL = AL mod 10 = **7** → AX = 0407h (paketsiz BCD).

Adım 8 / 16

DEĞİŞKENLER

AL = **07h** | AH = **04h** | BX = **0000h** | CF = **1** | AF = **1**

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
▶ add ax, 3030h      ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

Her iki bayta 30h eklenir: AX = **3437h** = '4','7'.

Adım 9 / 16

DEĞİŞKENLER

AL = **37h** AH = **34h** BX = **0000h** CF = **0** AF = **0**

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
▶ mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

Rakamlar BX'te saklanır, çünkü INT 21h/02h AL'yi değiştirir.

Adım 10 / 16

DEĞİŞKENLER

AL = 37h | AH = 34h | BX = 3437h | CF = 0 | AF = 0

ÇIKTI

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h      ; BCD 38
3  add al, 45h      ; + BCD 45
4  daa              ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47       ; ikili 2Fh
9  aam              ; AH=onlar AL=birler
10 add ax, 3030h    ; → ASCII
11 mov bx, ax
▶ mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

DL = '4'.

Adım 11 / 16

DEĞİŞKENLER

AL = **37h** AH = **34h** BX = **3437h** CF = **0** AF = **0**

ÇIKTI



Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
▶ mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
17 ret
```

AH = 02h.

Adım 12 / 16

DEĞİŞKENLER

AL = **37h** | AH = **02h** | BX = **3437h** | CF = **0** | AF = **0**

ÇIKTI



Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
▶ int 21h
15 mov dl, bl
16 int 21h
17 ret
```

Ekrana **4**.

Adım 13 / 16

DEĞİŞKENLER

AL = **34h** | AH = **02h** | BX = **3437h** | CF = **0** | AF = **0**

ÇIKTI

```
4
```

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h      ; BCD 38
3  add al, 45h      ; + BCD 45
4  daa              ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47       ; ikili 2Fh
9  aam              ; AH=onlar AL=birler
10 add ax, 3030h    ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
▶ mov dl, bl
16 int 21h
17 ret
```

DL = '7'.

Adım 14 / 16

DEĞİŞKENLER

AL = **34h** AH = **02h** BX = **3437h** CF = **0** AF = **0**

ÇIKTI

```
4
```

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h      ; BCD 38
3  add al, 45h      ; + BCD 45
4  daa              ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47       ; ikili 2Fh
9  aam              ; AH=onlar AL=birler
10 add ax, 3030h    ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
▶ int 21h
17 ret
```

Ekrana **7** → çıktı **47**.

Adım 15 / 16

DEĞİŞKENLER

AL = **37h** | AH = **02h** | BX = **3437h** | CF = **0** | AF = **0**

ÇIKTI

```
47
```

Kod İzleme: DAA ve AAM ile Onluk Sonuç

```
1  org 100h
2  mov al, 38h       ; BCD 38
3  add al, 45h       ; + BCD 45
4  daa               ; ondalık düzelt
5  mov al, 79h
6  add al, 35h
7  daa
8  mov al, 47        ; ikili 2Fh
9  aam               ; AH=onlar AL=birler
10 add ax, 3030h     ; → ASCII
11 mov bx, ax
12 mov dl, bh
13 mov ah, 2
14 int 21h
15 mov dl, bl
16 int 21h
▶  ret
```

AAM, 0–99 arası bir değeri iki basamaklı yazdırmanın en kısa yoludur (saat, tarih).

Adım 16 / 16

DEĞİŞKENLER

AL = **37h** | AH = **02h** | BX = **3437h** | CF = **0** | AF = **0**

ÇIKTI

```
47
```

MANTIK İŞLEMLERİ

Mantık Komutları

Komut	İşlem	Tipik kullanım	Örnek
`AND`	h ← h ∧ k	bit **temizleme** (maskeleme)	`and al, 0Fh`
`OR`	h ← h ∨ k	bit **kurma**	`or al, 80h`
`XOR`	h ← h ⊕ k	bit **tersleme**, sıfırlama	`xor ax, ax`
`NOT`	h ← ¬h	birler tümleyeni	`not bl`
`TEST`	h ∧ k (yazılmaz)	bit **sınama**	`test al, 1` → çift/tek

Bayraklar

AND, OR, XOR, TEST: **CF = OF = 0**; SF, ZF, PF sonuca göre; AF tanımsız. NOT hiçbir bayrağı değiştirmez.

Harf dönüşümü

'a' = 61h, 'A' = 41h: yalnız bit 5 farklı. `and al, 0DFh` büyük, `or al, 20h` küçük harf.

Rakam ↔ sayı

`and al, 0Fh`: '7' (37h) → 7. `or al, 30h`: 7 → '7'.

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
▶  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
7  test bl, 00001000b
8  test bl, 10000000b
9  not bl
10 xor ax, ax
11 ret
```

AL = 'g' = 67h = 0110 0111b. Küçük ve büyük harf yalnızca **bit 5**'te farklıdır.

Adım 1 / 10

DEĞİŞKENLER

AL = **67h** | BL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **0**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
▶  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
7  test bl, 00001000b
8  test bl, 10000000b
9  not bl
10 xor ax, ax
11 ret
```

`and al, 0DFh` (1101 1111b): bit 5 temizlenir → **47h = 'G'**. AND sonrası CF = OF = 0.

Adım 2 / 10

DEĞİŞKENLER

AL = **47h** | BL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **1**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
▶  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
7  test bl, 00001000b
8  test bl, 10000000b
9  not bl
10 xor ax, ax
11 ret
```

`or al, 20h` : bit 5 kurulur → **67h = 'g'** (küçük harfe çevirme).

Adım 3 / 10

DEĞİŞKENLER

AL = **67h** | BL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **0**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
▶  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
7  test bl, 00001000b
8  test bl, 10000000b
9  not bl
10 xor ax, ax
11 ret
```

`xor al, 20h` : bit 5 terslenir → **47h = 'G'**. XOR harf büyüklüğünü değiştirir.

Adım 4 / 10

DEĞİŞKENLER

AL = **47h** | BL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **1**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
▶  mov bl, 10110101b
7  test bl, 00001000b
8  test bl, 10000000b
9  not bl
10 xor ax, ax
11 ret
```

BL = 1011 0101b = B5h.

Adım 5 / 10

DEĞİŞKENLER

AL = **47h** | BL = **B5h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **1**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
▶  test bl, 00001000b
8  test bl, 10000000b
9  not bl
10 xor ax, ax
11 ret
```

`test` = sonucu yazmayan AND. Bit 3 = 0 → B5h AND 08h = 0 → **ZF = 1**.

Adım 6 / 10

DEĞİŞKENLER

AL = **47h** | BL = **B5h** | CF = **0** | ZF = **1** | SF = **0** | OF = **0** | PF = **1**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
7  test bl, 00001000b
▶  test bl, 10000000b
9  not bl
10 xor ax, ax
11 ret
```

Bit 7 = 1 → sonuç 80h ≠ 0 → **ZF = 0**, SF = 1. BL değişmedi.

Adım 7 / 10

DEĞİŞKENLER

AL = **47h** | BL = **B5h** | CF = **0** | ZF = **0** | SF = **1** | OF = **0** | PF = **0**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
7  test bl, 00001000b
8  test bl, 10000000b
▶  not bl
10 xor ax, ax
11 ret
```

`not bl` : tüm bitler terslenir → 0100 1010b = **4Ah**. NOT **hiçbir bayrağı** etkilemez.

Adım 8 / 10

DEĞİŞKENLER

AL = **47h** | BL = **4Ah** | CF = **0** | ZF = **0** | SF = **1** | OF = **0** | PF = **0**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
7  test bl, 00001000b
8  test bl, 10000000b
9  not bl
▶ xor ax, ax
11 ret
```

`xor ax, ax` : AX = 0, **ZF = 1**, CF = 0. MOV AX, 0'dan kısa (2 bayt).

Adım 9 / 10

DEĞİŞKENLER

AL = **00h** | BL = **4Ah** | CF = **0** | ZF = **1** | SF = **0** | OF = **0** | PF = **1**

Kod İzleme: AND, OR, XOR, TEST, NOT

```
1  org 100h
2  mov al, 'g'        ; 67h
3  and al, 0DFh       ; bit 5 = 0
4  or  al, 20h        ; bit 5 = 1
5  xor al, 20h        ; bit 5 tersle
6  mov bl, 10110101b
7  test bl, 00001000b
8  test bl, 10000000b
9  not bl
10 xor ax, ax
▶  ret
```

Son.

Adım 10 / 10

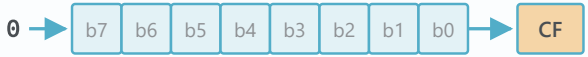
DEĞİŞKENLER

AL = **00h** | BL = **4Ah** | CF = **0** | ZF = **1** | SF = **0** | OF = **0** | PF = **1**

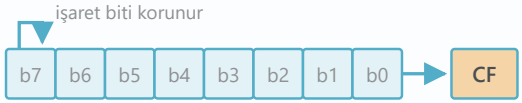
Kaydırma: SHL/SAL, SHR, SAR



`SHL/SAL`: sola kaydır, sağdan 0 girer, çıkan bit CF'ye. Her adım × **2**.



`SHR`: mantıksal sağa kaydır, soldan 0 girer. İşaretsiz ÷ **2**.



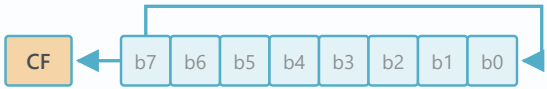
`SAR`: aritmetik sağa kaydır, işaret biti korunur. İşaretli ÷ **2** (−∞'a yuvarlar).

Kaydırma sayısı

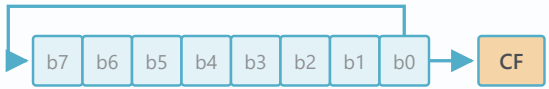
8086'da yalnızca `1` ya da `CL`: `shl ax, 1`, `shr al, cl`. `shl al, 4` gibi anlık sayı 80186 ve sonrasında geçerlidir. OF yalnızca sayı 1 iken tanımlıdır.

SAR ile IDIV farklı yuvarlar: −7 (F9h) `sar` 1 → FCh = **−4**; `idiv` 2 → **−3**.

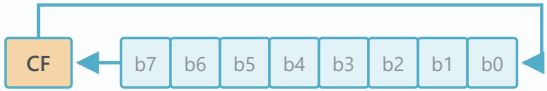
Döndürme: ROL, ROR, RCL, RCR



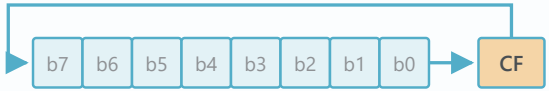
ROL: sola döndür; b7 hem b0'a hem CF'ye gider.



ROR: sağa döndür; b0 hem b7'ye hem CF'ye gider.



RCL: CF üzerinden sola döndür (9 bitlik halka).



RCR: CF üzerinden sağa döndür.

Döndürmede bit kaybolmaz. Yalnızca **CF** ve (sayı 1 iken) **OF** değişir; SF, ZF, PF etkilenmez. RCL/RCR çok sözcüklü kaydırmalarda kullanılır: `shl ax, 1 + rcl dx, 1` → DX:AX × 2.

KAYDIRMA VE DÖNDÜRME

Kaydırma ve Döndürme Gezgini

AL = B4 **SHL** **SHR** **SAR** **ROL** **ROR** **RCL** **RCR** **CF tersle**

AL **1 0 1 1 0 1 0 0** **B4h = 180 / -76**

CF = 0

AL = **B4h**. Bir komut seçin.

Her düğme komutu `op al, 1` olarak uygular. SHL'yi art arda uygulayıp sayının iki katına çıktığını, SAR'ın işareti koruduğunu gözleyin.

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
▶  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

Çarpılan BL = 13 = **0Dh** (0000 1101b).

Adım 1 / 22

DEĞİŞKENLER

BL = **0Dh** | DL = **00h** | CX = **0000h** | CF = **0**

sonuc = **00h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan   ; 13
▶  mov dl, carpan     ; 5 = 0101b
4  mov cx, 4          ; 4 bit
5  kontrol:
6  shr dl, 1          ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl      ; bit 1 ise ekle
9  atla:
10 shl bl, 1          ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

Çarpan DL = 5 = **0101b**: kağıt üzerindeki çarpmada olduğu gibi bitleri tek tek inceleriz.

Adım 2 / 22

DEĞİŞKENLER

BL = **0Dh** | DL = **05h** | CX = **0000h** | CF = **0** | sonuc = **00h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
▶  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

4 bitlik çarpan için 4 tur.

Adım 3 / 22

DEĞİŞKENLER

BL = 0Dh | DL = 05h | CX = 0004h | CF = 0 | sonuc = 00h

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan   ; 13
3  mov dl, carpan     ; 5 = 0101b
4  mov cx, 4          ; 4 bit
5  kontrol:
▶  shr dl, 1          ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl      ; bit 1 ise ekle
9  atla:
10 shl bl, 1          ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

`shr dl, 1` : çıkan bit 1 → **CF = 1**.

Adım 4 / 22

DEĞİŞKENLER

BL = **0Dh** | DL = **02h** | CX = **0004h** | CF = **1** | sonuc = **00h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
▶  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

CF = 1 → atlama yok, toplama yapılacak.

Adım 5 / 22

DEĞİŞKENLER

BL = **0Dh** | DL = **02h** | CX = **0004h** | CF = **1** | sonuc = **00h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
▶  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

sonuc = 0 + 0Dh = **0Dh** (13 × 1).

Adım 6 / 22

DEĞİŞKENLER

BL = **0Dh** DL = **02h** CX = **0004h** CF = **0**

sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
▶  shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

Çarpılan bir sola: BL = **1Ah** (26).

Adım 7 / 22

DEĞİŞKENLER

BL = **1Ah** | DL = **02h** | CX = **0004h** | CF = **0**

sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
▶ loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

`loop` : CX = 0003h.

Adım 8 / 22

DEĞİŞKENLER

BL = **1Ah** | DL = **02h** | CX = **0003h** | CF = **0** | sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
▶  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

Çıkan bit 0 → **CF = 0**.

Adım 9 / 22

DEĞİŞKENLER

BL = **1Ah** | DL = **01h** | CX = **0003h** | CF = **0** | sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
▶  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

CF = 0 → `atla` 'ya atlar.

Adım 10 / 22

DEĞİŞKENLER

BL = **1Ah** | DL = **01h** | CX = **0003h** | CF = **0** | sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
▶  shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

BL = **34h** (52).

Adım 11 / 22

DEĞİŞKENLER

BL = **34h** | DL = **01h** | CX = **0003h** | CF = **0**

sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
▶ loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

`loop` : CX = 0002h.

Adım 12 / 22

DEĞİŞKENLER

BL = **34h** | DL = **01h** | CX = **0002h** | CF = **0** | sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
▶  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

Çıkan bit 1 → **CF = 1**.

Adım 13 / 22

DEĞİŞKENLER

BL = **34h** | DL = **00h** | CX = **0002h** | CF = **1** | sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
▶  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

CF = 1 → toplama yapılacak.

Adım 14 / 22

DEĞİŞKENLER

BL = **34h** | DL = **00h** | CX = **0002h** | CF = **1**

sonuc = **0Dh**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
▶  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

sonuc = 0Dh + 34h = **41h = 65** (13 × 4 eklendi).

Adım 15 / 22

DEĞİŞKENLER

BL = **34h** DL = **00h** CX = **0002h** CF = **0**

sonuc = **41h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
▶  shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

BL = **68h** (104).

Adım 16 / 22

DEĞİŞKENLER

BL = **68h** | DL = **00h** | CX = **0002h** | CF = **0**

sonuc = **41h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
▶ loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

`loop` : CX = 0001h.

Adım 17 / 22

DEĞİŞKENLER

BL = **68h** | DL = **00h** | CX = **0001h** | CF = **0** | sonuc = **41h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
▶  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

Çıkan bit 0 → CF = 0.

Adım 18 / 22

DEĞİŞKENLER

BL = **68h** | DL = **00h** | CX = **0001h** | CF = **0** | sonuc = **41h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
▶  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

CF = 0 → atla.

Adım 19 / 22

DEĞİŞKENLER

BL = **68h** | DL = **00h** | CX = **0001h** | CF = **0** | sonuc = **41h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
 1  org 100h
 2  mov bl, carpilan  ; 13
 3  mov dl, carpan    ; 5 = 0101b
 4  mov cx, 4         ; 4 bit
 5  kontrol:
 6  shr dl, 1         ; en sağ bit → CF
 7  jnc atla
 8  add sonuc, bl     ; bit 1 ise ekle
 9  atla:
▶ 10  shl bl, 1        ; çarpılan × 2
11  loop kontrol
12  ret
13  carpilan db 13
14  carpan   db 5
15  sonuc    db 0
```

BL = **D0h**.

Adım 20 / 22

DEĞİŞKENLER

BL = **D0h** | DL = **00h** | CX = **0001h** | CF = **0**

sonuc = **41h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
▶ loop kontrol
12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

`loop` : CX = 0000h.

Adım 21 / 22

DEĞİŞKENLER

BL = **D0h** | DL = **00h** | CX = **0000h** | CF = **0** | sonuc = **41h**

Kod İzleme: Kaydırarak Çarpma (13 × 5)

```
1  org 100h
2  mov bl, carpilan  ; 13
3  mov dl, carpan    ; 5 = 0101b
4  mov cx, 4         ; 4 bit
5  kontrol:
6  shr dl, 1         ; en sağ bit → CF
7  jnc atla
8  add sonuc, bl     ; bit 1 ise ekle
9  atla:
10 shl bl, 1         ; çarpılan × 2
11 loop kontrol
▶ 12 ret
13 carpilan db 13
14 carpan   db 5
15 sonuc    db 0
```

Sonuç: 13 × 5 = **65 = 41h**. MUL olmayan işlemcilerde çarpma böyle yapılır.

Adım 22 / 22

DEĞİŞKENLER

BL = **D0h** | DL = **00h** | CX = **0000h** | CF = **0** | sonuc = **41h**

Kaydırmayla Hızlı Çarpma ve Bölme

```
; AX ← AX × 10 = AX×8 + AX×2
shl ax, 1       ; AX × 2
mov bx, ax      ; sakla
shl ax, 1       ; AX × 4
shl ax, 1       ; AX × 8
add ax, bx      ; ×8 + ×2

; işaretsiz ÷ 16, kalan
mov bx, ax
and bx, 000Fh   ; kalan = alt 4 bit
mov cl, 4
shr ax, cl      ; bölüm
```

- 8086'da `mul` 70–133, `div` 80–162 saat turu sürer; bir `shl reg, 1` yalnızca **2** tur.
- 2'nin kuvvetleriyle çarpma/bölme kaydırmaya, diğer sabitler kaydırma + toplamaya dönüştürülebilir.
- 2^n'e bölümden kalan: `and` ile alt n bit.
- İşaretli bölmede SAR kullanılır; negatif sayılarda yuvarlama farkına dikkat.

Örnek: AX = 25 → 50 → (BX = 50) → 100 → 200 → 200 + 50 = **250**.

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
▶  mov ax, 37
3  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```

AX = 0025h = 37



BX = 0000h = 0



AX = 37 = **0025h** = 0000 0000 0010 0101b.

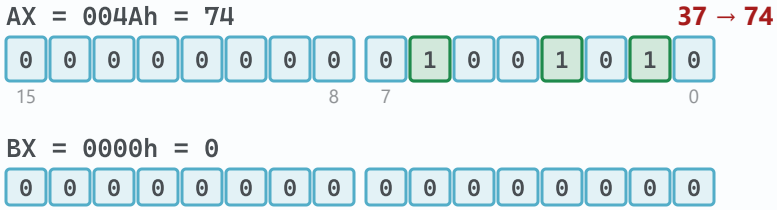
Adım 1 / 11

DEĞİŞKENLER

AX = **0025h (37)** BX = **0000h** CL = **00h**

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
▶  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```



SHL AX, 1: bitler bir sola, sağdan 0 girer → AX = **74** (004Ah). 2 saat turu; MUL 70 turdan fazla sürer.

Adım 2 / 11

DEĞİŞKENLER

AX = **004Ah (74)** | BX = **0000h** | CL = **00h**

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1        ; AX × 2
▶  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```

AX = 004Ah = 74



BX = 004Ah = 74



BX = 74: ×2'yi sakla.

Adım 3 / 11

DEĞİŞKENLER

AX = 004Ah (74) | BX = 004Ah | CL = 00h

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
▶  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```

AX = 0094h = 148 — 74 → 148

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0

BX = 004Ah = 74

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0

SHL → AX = **148** (×4).

Adım 4 / 11

DEĞİŞKENLER

AX = **0094h (148)** | BX = **004Ah** | CL = **00h**

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
▶  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```

AX = 0128h = 296 — 148 → 296

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0

BX = 004Ah = 74

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0

SHL → AX = **296** (×8).

Adım 5 / 11

DEĞİŞKENLER

AX = **0128h (296)** | BX = **004Ah** | CL = **00h**

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
▶  add ax, bx       ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```

AX = 0172h = 370 — 296 → 370



BX = 004Ah = 74



ADD AX, BX: 296 + 74 = **370** = 0172h = 37 × 10.

Adım 6 / 11

DEĞİŞKENLER

AX = **0172h (370)** | BX = **004Ah** | CL = **00h**

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
▶  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```

AX = 0172h = 370



BX = 0172h = 370



BX = 370: bölme için kopya.

Adım 7 / 11

DEĞİŞKENLER

AX = 0172h (370) | BX = 0172h | CL = 00h

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
8  mov bx, ax
▶  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```

AX = 0172h = 370



BX = 0002h = 2



AND BX, 000Fh: alt 4 bit = 370 mod 16 = **2**.

Adım 8 / 11

DEĞİŞKENLER

AX = **0172h (370)** BX = **0002h** CL = **00h**

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
▶  mov cl, 4
11 shr ax, cl       ; bölüm: ÷ 16
12 ret
```

AX = 0172h = 370



BX = 0002h = 2



CL = 4: 8086'da 1'den büyük kaydırma sayısı CL ile verilir.

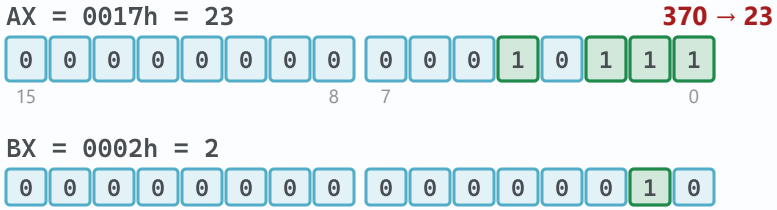
Adım 9 / 11

DEĞİŞKENLER

AX = **0172h (370)** BX = **0002h** CL = **04h**

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1        ; AX × 2
4  mov bx, ax       ; sakla
5  shl ax, 1        ; AX × 4
6  shl ax, 1        ; AX × 8
7  add ax, bx       ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh    ; kalan: alt 4 bit
10 mov cl, 4
▶  shr ax, cl       ; bölüm: ÷ 16
12 ret
```



SHR AX, CL: 4 bit sağa = ÷ 16 → AX = **23** (0017h). 23 × 16 + 2 = 370.

Adım 10 / 11

DEĞİŞKENLER

AX = **0017h (23)** | BX = **0002h** | CL = **04h**

Kaydırmayla 37 × 10 ve 370 ÷ 16

```
1  org 100h
2  mov ax, 37
3  shl ax, 1       ; AX × 2
4  mov bx, ax      ; sakla
5  shl ax, 1       ; AX × 4
6  shl ax, 1       ; AX × 8
7  add ax, bx      ; ×8 + ×2
8  mov bx, ax
9  and bx, 000Fh   ; kalan: alt 4 bit
10 mov cl, 4
11 shr ax, cl      ; bölüm: ÷ 16
▶  ret
```

AX = 0017h = 23



BX = 0002h = 2



RET: program biter.

Adım 11 / 11

DEĞİŞKENLER

AX = **0017h (23)** BX = **0002h** CL = **04h**

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).

Adım 1 / 4

```
; 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 = 0110h: kaynak dizgi.

Adım 2 / 13

DEĞİŞKENLER

`SI = 0110h` `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 = 0114h: hedef alan ('....').

Adım 3 / 13

DEĞİŞKENLER

SI = **0110h** | DI = **0114h** | 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 = **0110h** DI = **0114h** 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 = 0111h, DI = 0115h. Bayraklar değişmez.

Adım 5 / 13

DEĞİŞKENLER

SI = **0111h** DI = **0115h** 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 = **0111h** DI = **0115h** 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 = 0112h, DI = 0116h. Bayraklar değişmez.

Adım 7 / 13

DEĞİŞKENLER

SI = **0112h** DI = **0116h** 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 = **0112h** DI = **0116h** 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 = 0113h, DI = 0117h. Bayraklar değişmez.

Adım 9 / 13

DEĞİŞKENLER

SI = **0113h** DI = **0117h** 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 = **0113h** DI = **0117h** 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 = 0114h, DI = 0118h. Bayraklar değişmez.

Adım 11 / 13

DEĞİŞKENLER

SI = **0114h** DI = **0118h** 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 = **0114h** DI = **0118h** 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 = **0114h** DI = **0118h** 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 = 0111h (metin).

Adım 2 / 25

DEĞİŞKENLER

AL = **00h** | SI = **0111h** | 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 = 0111h. COM programında ES = DS olduğundan ES:DI aynı yeri gösterir.

Adım 3 / 25

DEĞİŞKENLER

AL = **00h** | SI = **0111h** | DI = **0111h** | 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 = 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:
▶  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 = 0112h.

Adım 5 / 25

DEĞİŞKENLER

AL = **6Dh** SI = **0112h** 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++
▶  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 = **0112h** | 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
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

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

Adım 7 / 25

DEĞİŞKENLER

AL = **4Dh** | SI = **0112h** | DI = **0112h** | 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 = **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:
▶  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 = 0113h.

Adım 9 / 25

DEĞİŞKENLER

AL = **69h** | SI = **0113h** | 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++
▶  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 = **0113h** | 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++
10 loop don
11 ret
12 metin db 'mikro'
```

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

Adım 11 / 25

DEĞİŞKENLER

AL = **49h** SI = **0113h** DI = **0113h** 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 = **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:
▶  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 = 0114h.

Adım 13 / 25

DEĞİŞKENLER

AL = **6Bh** SI = **0114h** 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++
▶  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 = **0114h** | 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
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

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

Adım 15 / 25

DEĞİŞKENLER

AL = **4Bh** | SI = **0114h** | DI = **0114h** | 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 = **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:
▶  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 = 0115h.

Adım 17 / 25

DEĞİŞKENLER

AL = **72h** | SI = **0115h** | 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++
▶  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 = **0115h** | 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
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

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

Adım 19 / 25

DEĞİŞKENLER

AL = **52h** | SI = **0115h** | DI = **0115h** | 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 = **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:
▶  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 = 0116h.

Adım 21 / 25

DEĞİŞKENLER

AL = **6Fh** SI = **0116h** 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++
▶  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 = **0116h** | 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
▶  stosb              ; [DI] = AL, DI++
10 loop don
11 ret
12 metin db 'mikro'
```

`stosb` : [ES:DI] ← AL, ardından DI = 0116h.

Adım 23 / 25

DEĞİŞKENLER

AL = **4Fh** | SI = **0116h** | DI = **0116h** | 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 = **0116h** | DI = **0116h** | 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 = **0116h** DI = **0116h** 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 = 0114h: taranacak dizgi ES:DI'de olmalı.

Adım 2 / 9

DEĞİŞKENLER

AL = 00h | DI = 0114h | CX = 0000h | ZF = 0

Kod İzleme: REPNE SCASB ile Karakter Arama

```
1  org 100h
2  cld
3  lea di, metin
▶  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 = 0114h | 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 = **0114h** | 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
▶ 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 = 0118h, CX = 0003h**.

Adım 5 / 9

DEĞİŞKENLER

AL = **68h** | DI = **0118h** | 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 = 0118h` `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 = 0117h.

Adım 7 / 9

DEĞİŞKENLER

AL = **68h** | DI = **0117h** | 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:
▶ 11 ret
12 metin db 'merhaba'
```

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

Adım 9 / 9

DEĞİŞKENLER

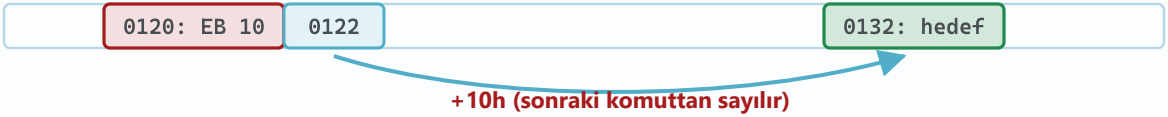
AL = **68h** | DI = **0003h** | CX = **0003h** | ZF = **0**

KONTROL AKTARMA

Koşulsuz Atlama: JMP

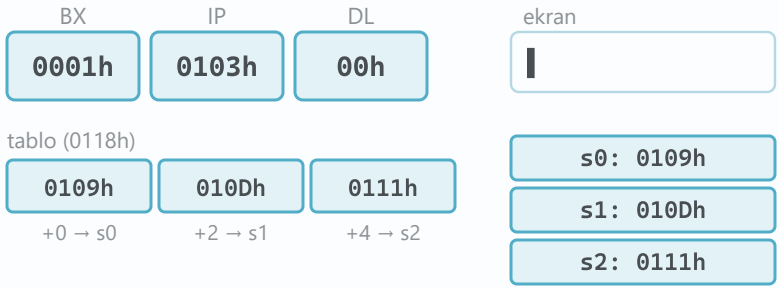
Tür	Kod	Erim	Örnek
Kısa (short)	`EB d8`	−128 … +127 bayt	`jmp short dongu`
Yakın (near)	`E9 d16`	aynı kesim, ±32 K	`jmp basla`
Uzak (far)	`EA IP CS`	herhangi bir kesim	`jmp far ptr hedef`
Dolaylı yakın	`FF /4`	IP ← yazmaç/bellek	`jmp bx`, `jmp tablo[bx]`
Dolaylı uzak	`FF /5`	CS:IP ← bellek (4 bayt)	`jmp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



Dolaylı JMP ile Atlama Tablosu

```
1  org 100h
▶  mov bx, 1       ; seçim (0–2)
3  shl bx, 1       ; sözcük dizini
4  jmp tablo[bx]   ; IP ← [tablo+BX]
5  s0: mov dl, 'A'
6  jmp yaz
7  s1: mov dl, 'B'
8  jmp yaz
9  s2: mov dl, 'C'
10 yaz: mov ah, 2
11 int 21h
12 ret
13 tablo dw s0, s1, s2
```



Seçim BX = 1. Tabloda üç yordam adresi var: s0 = 0109h, s1 = 010Dh, s2 = 0111h (her biri bir sözcük).

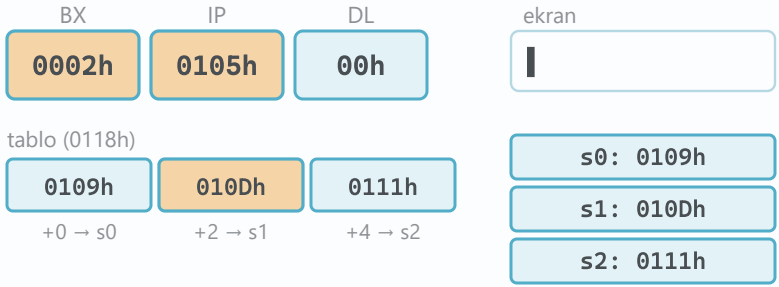
Adım 1 / 8

DEĞİŞKENLER

`BX = 0001h` `IP = 0103h` `DL = 00h`

Dolaylı JMP ile Atlama Tablosu

```
1  org 100h
2  mov bx, 1       ; seçim (0–2)
3  shl bx, 1       ; sözcük dizini
4  jmp tablo[bx]   ; IP ← [tablo+BX]
5  s0: mov dl, 'A'
6  jmp yaz
7  s1: mov dl, 'B'
8  jmp yaz
9  s2: mov dl, 'C'
10 yaz: mov ah, 2
11 int 21h
12 ret
13 tablo dw s0, s1, s2
```



SHL BX, 1: her girdi 2 bayt olduğu için dizin ×2 → BX = **2**.

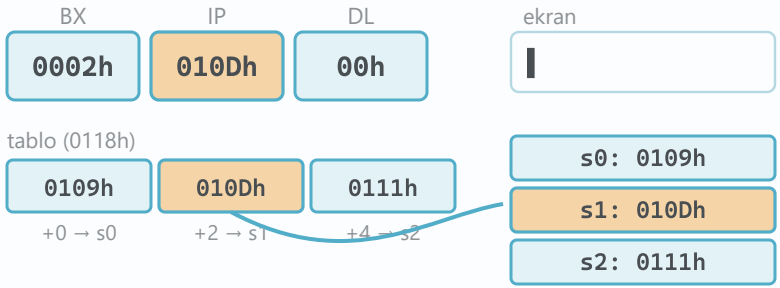
Adım 2 / 8

DEĞİŞKENLER

BX = 0002h IP = 0105h DL = 00h

Dolaylı JMP ile Atlama Tablosu

```
1  org 100h
2  mov bx, 1       ; seçim (0–2)
3  shl bx, 1       ; sözcük dizini
▶  jmp tablo[bx]   ; IP ← [tablo+BX]
5  s0: mov dl, 'A'
6  jmp yaz
7  s1: mov dl, 'B'
8  jmp yaz
9  s2: mov dl, 'C'
10 yaz: mov ah, 2
11 int 21h
12 ret
13 tablo dw s0, s1, s2
```



JMP tablo[BX]: IP ← [0118h + 2] = **010Dh** (s1). Hedef bellekten okunur: dolaylı yakın atlama.

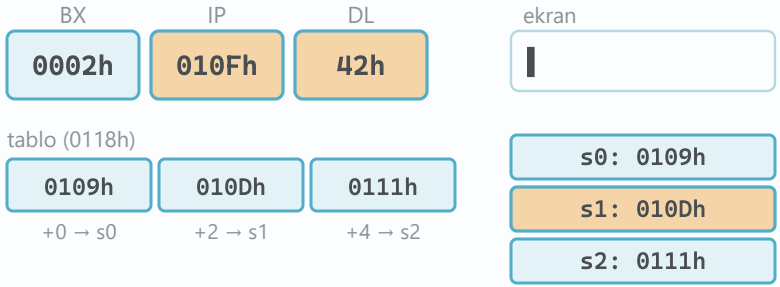
Adım 3 / 8

DEĞİŞKENLER

BX = 0002h | IP = 010Dh | DL = 00h

Dolaylı JMP ile Atlama Tablosu

```
1  org 100h
2  mov bx, 1       ; seçim (0–2)
3  shl bx, 1       ; sözcük dizini
4  jmp tablo[bx]   ; IP ← [tablo+BX]
5  s0: mov dl, 'A'
6  jmp yaz
▶  s1: mov dl, 'B'
8  jmp yaz
9  s2: mov dl, 'C'
10 yaz: mov ah, 2
11 int 21h
12 ret
13 tablo dw s0, s1, s2
```



s1: DL = 'B'.

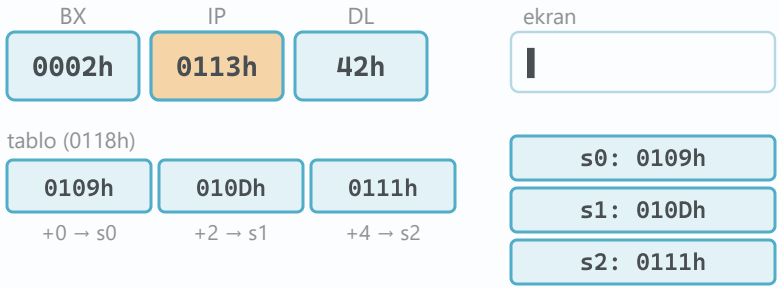
Adım 4 / 8

DEĞİŞKENLER

BX = 0002h IP = 010Fh DL = 42h

Dolaylı JMP ile Atlama Tablosu

```
1  org 100h
2  mov bx, 1       ; seçim (0–2)
3  shl bx, 1       ; sözcük dizini
4  jmp tablo[bx]   ; IP ← [tablo+BX]
5  s0: mov dl, 'A'
6  jmp yaz
7  s1: mov dl, 'B'
▶  jmp yaz
9  s2: mov dl, 'C'
10 yaz: mov ah, 2
11 int 21h
12 ret
13 tablo dw s0, s1, s2
```



JMP yaz.

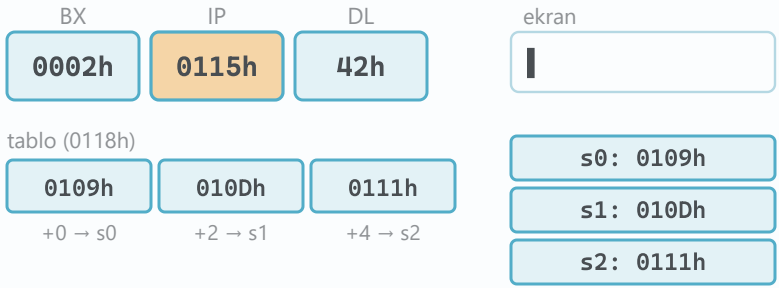
Adım 5 / 8

DEĞİŞKENLER

BX = 0002h | IP = 0113h | DL = 42h

Dolaylı JMP ile Atlama Tablosu

```
1  org 100h
2  mov bx, 1       ; seçim (0–2)
3  shl bx, 1       ; sözcük dizini
4  jmp tablo[bx]   ; IP ← [tablo+BX]
5  s0: mov dl, 'A'
6  jmp yaz
7  s1: mov dl, 'B'
8  jmp yaz
9  s2: mov dl, 'C'
▶  yaz: mov ah, 2
11 int 21h
12 ret
13 tablo dw s0, s1, s2
```



AH = 2.

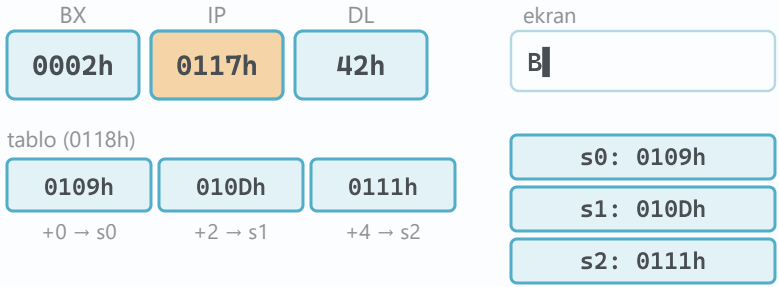
Adım 6 / 8

DEĞİŞKENLER

BX = 0002h | IP = 0115h | DL = 42h

Dolaylı JMP ile Atlama Tablosu

```
1  org 100h
2  mov bx, 1       ; seçim (0–2)
3  shl bx, 1       ; sözcük dizini
4  jmp tablo[bx]   ; IP ← [tablo+BX]
5  s0: mov dl, 'A'
6  jmp yaz
7  s1: mov dl, 'B'
8  jmp yaz
9  s2: mov dl, 'C'
10 yaz: mov ah, 2
▶  int 21h
12 ret
13 tablo dw s0, s1, s2
```



INT 21h: ekrana **B**. CMP/JE zinciri olmadan tek komutla seçim yapıldı (switch-case).

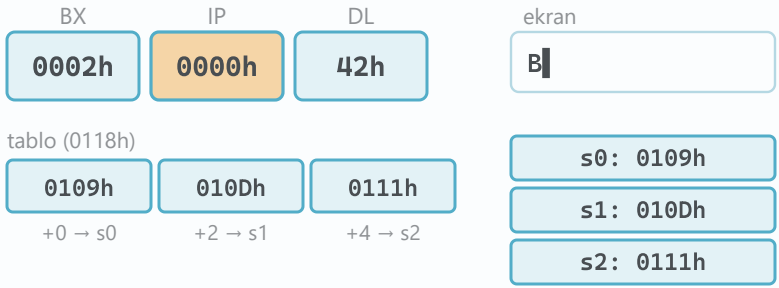
Adım 7 / 8

DEĞİŞKENLER

BX = **0002h** IP = **0117h** DL = **42h**

Dolaylı JMP ile Atlama Tablosu

```
1  org 100h
2  mov bx, 1       ; seçim (0–2)
3  shl bx, 1       ; sözcük dizini
4  jmp tablo[bx]   ; IP ← [tablo+BX]
5  s0: mov dl, 'A'
6  jmp yaz
7  s1: mov dl, 'B'
8  jmp yaz
9  s2: mov dl, 'C'
10 yaz: mov ah, 2
11 int 21h
▶ 12 ret
13 tablo dw s0, s1, s2
```



RET: program biter.

Adım 8 / 8

DEĞİŞKENLER

BX = 0002h IP = 0000h DL = 42h

KONTROL AKTARMA

Koşullu Atlamalar (Jcc)

İşaretsiz karşılaştırma	
`JA / JNBE`	CF = 0 ve ZF = 0 (>)
`JAE / JNB / JNC`	CF = 0 (≥)
`JB / JNAE / JC`	CF = 1 (<)
`JBE / JNA`	CF = 1 veya ZF = 1 (≤)

İşaretli karşılaştırma	
`JG / JNLE`	ZF = 0 ve SF = OF (>)
`JGE / JNL`	SF = OF (≥)
`JL / JNGE`	SF ≠ OF (<)
`JLE / JNG`	ZF = 1 veya SF ≠ OF (≤)

Tek bayrak / eşitlik	
`JE / JZ`	ZF = 1
`JNE / JNZ`	ZF = 0
`JS` / `JNS`	SF = 1 / SF = 0
`JO` / `JNO`	OF = 1 / OF = 0
`JP / JPE`	PF = 1 (çift eşlik)
`JNP / JPO`	PF = 0 (tek eşlik)
`JCXZ`	CX = 0 (bayrak değil)

Akılda tutma: **A**bove/**B**elow → işaretsiz; **G**reater/**L**ess → işaretli. N = not, E = equal.

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
▶ mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

BL = FFh. İşaretsiz yorum **255**, işaretli yorum **−1**.

Adım 1 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0**

ÇIKTI

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
▶  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

FFh − 01h = FEh: **CF = 0** (ödünç yok), ZF = 0, **SF = 1, OF = 0**.

Adım 2 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **00h** | CF = **0** | ZF = **0** | SF = **1** | OF = **0**

ÇIKTI

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
▶ 4  jg isaretli        ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz         ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

`jg` : ZF = 0 **ve** SF = OF ister. SF ≠ OF → −1 > 1 değil, **atlamaz**.

Adım 3 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **00h** | CF = **0** | ZF = **0** | SF = **1** | OF = **0**

ÇIKTI

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
▶  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

DL = 'K' (küçük).

Adım 4 / 12

DEĞİŞKENLER

BL = FFh | DL = 4Bh | CF = 0 | ZF = 0 | SF = 1 | OF = 0

ÇIKTI

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

Ortak yazdırma koduna atla.

Adım 5 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **4Bh** | CF = **0** | ZF = **0** | SF = **1** | OF = **0**

ÇIKTI

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
▶  mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

AH = 02h: karakter yaz.

Adım 6 / 12

DEĞİŞKENLER

BL = **FFh** DL = **4Bh** CF = **0** ZF = **0** SF = **1** OF = **0**

ÇIKTI

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
▶ int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

İşaretli karşılaştırmanın sonucu ekranda: **K**.

Adım 7 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **4Bh** | CF = **0** | ZF = **0** | SF = **1** | OF = **0**

ÇIKTI

```
K
```

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
▶ cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

Aynı karşılaştırma yeniden (INT 21h bayrakları değiştirmiş olabilir).

Adım 8 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **4Bh** | CF = **0** | ZF = **0** | SF = **1** | OF = **0**

ÇIKTI

```
K
```

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
▶ ja isaretsiz       ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

`ja` : CF = 0 **ve** ZF = 0 ister → 255 > 1, **atlar**.

Adım 9 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **4Bh** | CF = **0** | ZF = **0** | SF = **1** | OF = **0**

ÇIKTI

```
K
```

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
▶  mov dl, 'B'
18 yaz2:
19 int 21h
20 ret
```

DL = 'B' (büyük).

Adım 10 / 12

DEĞİŞKENLER

BL = FFh | DL = 42h | CF = 0 | ZF = 0 | SF = 1 | OF = 0

ÇIKTI

```
K
```

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
▶ int 21h
20 ret
```

Çıktı **KB**: aynı bit dizisi, farklı atlama.

Adım 11 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **42h** | CF = **0** | ZF = **0** | SF = **1** | OF = **0**

ÇIKTI

```
KB
```

Kod İzleme: Aynı Bayt, İki Farklı Karşılaştırma

```
1  org 100h
2  mov bl, 0FFh
3  cmp bl, 1
4  jg isaretli       ; -1 > 1 ?
5  mov dl, 'K'
6  jmp yaz1
7  isaretli:
8  mov dl, 'B'
9  yaz1:
10 mov ah, 2
11 int 21h
12 cmp bl, 1
13 ja isaretsiz      ; 255 > 1 ?
14 mov dl, 'K'
15 jmp yaz2
16 isaretsiz:
17 mov dl, 'B'
18 yaz2:
19 int 21h
   ret
```

İşaretli veride JG/JGE/JL/JLE, **işaretsiz** veride JA/JAE/JB/JBE kullanılır.

Adım 12 / 12

DEĞİŞKENLER

BL = **FFh** | DL = **42h** | CF = **0** | ZF = **0** | SF = **1** | OF = **0**

ÇIKTI

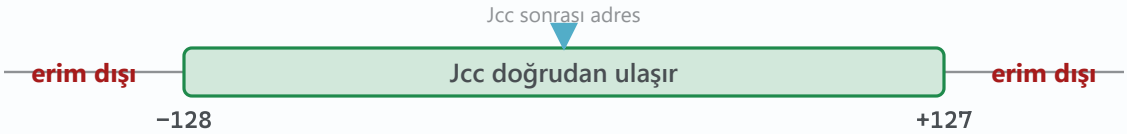
```
KB
```

8086'da Koşullu Atlama Erimi

- 8086'da tüm Jcc komutları **kısa**dır: 2 bayt, hedef −128…+127 bayt içinde olmalı.
- Hedef uzaksa assembler **"relative jump out of range"** hatası verir.
- Çözüm: koşulu **tersine çevir** ve yakın JMP ile atla.
- 80386 ve sonrasında 16/32 bit yer değiştirmeli Jcc (0F 8x) vardır.

```
; hatalı (hedef çok uzak)
cmp al, 0Dh
je  bitis        ; > 127 bayt

; düzeltilmiş
cmp al, 0Dh
jne devam        ; ters koşul, kısa
jmp bitis        ; yakın, ±32 K
devam:
```



Örnek kod: `EB FE` = "kendine atla" (sonsuz döngü), çünkü FEh = −2 ve komut 2 bayttır.

KONTROL AKTARMA

Döngü Komutları

Komut	İşlem	Atlama koşulu
`LOOP h`	CX ← CX − 1	CX ≠ 0
`LOOPE / LOOPZ h`	CX ← CX − 1	CX ≠ 0 **ve** ZF = 1
`LOOPNE / LOOPNZ h`	CX ← CX − 1	CX ≠ 0 **ve** ZF = 0
`JCXZ h`	(CX değişmez)	CX = 0

Tuzak: CX = 0

LOOP önce azaltır: 0 − 1 = FFFFh ≠ 0 → döngü **65 536** kez döner. Döngüden önce `jcxz atla` ile korunun.

Bayraklar

LOOP ailesi bayrakları **değiştirmez**; LOOPE/LOOPNE gövdedeki son CMP'nin ZF'sini okur. Hepsi kısa atlamadır.

```
mov cx, n
jcxz bos        ; n = 0 ise hiç girme
tekrar: ; …gövde…
loop tekrar
bos:
```

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
▶ lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5         ; eleman sayısı
4  mov ax, 0         ; toplam
5  l1:
6  add ax, [bx]      ; elemanı ekle
7  add bx, 2         ; sonraki sözcük
8  loop l1           ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

BX = 0115h: dizinin ilk elemanının adresi.

Adım 1 / 20

DEĞİŞKENLER

AX = **0000h** BX = **0115h** CX = **0000h**

toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
▶  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

CX = 5: LOOP sayacı.

Adım 2 / 20

DEĞİŞKENLER

AX = **0000h** BX = **0115h** CX = **0005h**

toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
▶  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

Toplam AX = 0.

Adım 3 / 20

DEĞİŞKENLER

`AX = 0000h` `BX = 0115h` `CX = 0005h` `toplam = 0000h`

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
▶  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

AX = AX + [BX] = **000Ah**.

Adım 4 / 20

DEĞİŞKENLER

AX = **000Ah** | BX = **0115h** | CX = **0005h** | toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
▶  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

Sözcük (dw) dizisinde sonraki eleman **2 bayt** ileride: BX = 0117h.

Adım 5 / 20

DEĞİŞKENLER

AX = **000Ah** | BX = **0117h** | CX = **0005h** | toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
▶  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

`loop` : CX = 0004h ≠ 0 → l1'e dön.

Adım 6 / 20

DEĞİŞKENLER

AX = **000Ah** BX = **0117h** CX = **0004h**

toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
▶  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

AX = AX + [BX] = **001Eh**.

Adım 7 / 20

DEĞİŞKENLER

AX = **001Eh** | BX = **0117h** | CX = **0004h** | toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
▶  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

Sözcük (dw) dizisinde sonraki eleman **2 bayt** ileride: BX = 0119h.

Adım 8 / 20

DEĞİŞKENLER

AX = **001Eh** | BX = **0119h** | CX = **0004h** | toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
▶ loop l1             ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

CX = 0003h → dön.

Adım 9 / 20

DEĞİŞKENLER

AX = **001Eh** BX = **0119h** CX = **0003h**

toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
▶  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

AX = AX + [BX] = **003Ch**.

Adım 10 / 20

DEĞİŞKENLER

AX = **003Ch** | BX = **0119h** | CX = **0003h** | toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
▶  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

Sözcük (dw) dizisinde sonraki eleman **2 bayt** ileride: BX = 011Bh.

Adım 11 / 20

DEĞİŞKENLER

AX = **003Ch** | BX = **011Bh** | CX = **0003h** | toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
▶  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

CX = 0002h → dön.

Adım 12 / 20

DEĞİŞKENLER

AX = **003Ch** BX = **011Bh** CX = **0002h**

toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
▶  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

AX = AX + [BX] = **0064h**.

Adım 13 / 20

DEĞİŞKENLER

AX = **0064h** | BX = **011Bh** | CX = **0002h** | toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
▶  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

Sözcük (dw) dizisinde sonraki eleman **2 bayt** ileride: BX = 011Dh.

Adım 14 / 20

DEĞİŞKENLER

`AX = 0064h` `BX = 011Dh` `CX = 0002h` `toplam = 0000h`

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
▶  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

CX = 0001h → dön.

Adım 15 / 20

DEĞİŞKENLER

AX = **0064h** BX = **011Dh** CX = **0001h**

toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
▶  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

AX = AX + [BX] = **0096h**.

Adım 16 / 20

DEĞİŞKENLER

AX = **0096h** | BX = **011Dh** | CX = **0001h** | toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
▶  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

Sözcük (dw) dizisinde sonraki eleman **2 bayt** ileride: BX = 011Fh.

Adım 17 / 20

DEĞİŞKENLER

`AX = 0096h` `BX = 011Fh` `CX = 0001h` `toplam = 0000h`

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
▶  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

CX = 0 → döngüden çık.

Adım 18 / 20

DEĞİŞKENLER

AX = **0096h** BX = **011Fh** CX = **0000h**

toplam = **0000h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
▶  mov toplam, ax
10 ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

toplam = AX = **0096h = 150**.

Adım 19 / 20

DEĞİŞKENLER

AX = **0096h** BX = **011Fh** CX = **0000h**

toplam = **0096h**

Kod İzleme: LOOP ile Dizi Toplamı

```
1  org 100h
2  lea bx, dizi       ; BX → ilk eleman
3  mov cx, 5          ; eleman sayısı
4  mov ax, 0          ; toplam
5  l1:
6  add ax, [bx]       ; elemanı ekle
7  add bx, 2          ; sonraki sözcük
8  loop l1            ; CX--, ≠0 ise dön
9  mov toplam, ax
▶  ret
11 dizi   dw 10, 20, 30, 40, 50
12 toplam dw 0
```

LOOP = `dec cx` + `jnz`, ama bayrakları etkilemez ve 2 bayttır.

Adım 20 / 20

DEĞİŞKENLER

AX = **0096h** BX = **011Fh** CX = **0000h**

toplam = **0096h**

Prosedür Çağrısı: CALL ve RET

Komut	Yığına / yığından	Sonra
`CALL yakın`	push IP (dönüş adresi)	IP ← hedef
`CALL uzak`	push CS, push IP	CS:IP ← hedef
`RET`	pop IP	çağıranın bir sonraki komutu
`RETF`	pop IP, pop CS	uzak dönüş
`RET n`	pop IP, SP ← SP + n	yığındaki n baytlık parametreyi atar

- Dönüş adresi, CALL'dan **sonraki** komutun adresidir.
- Prosedür içinde PUSH sayısı POP sayısına eşit olmalı; aksi hâlde RET yanlış adrese döner.
- emu8086'da prosedür `ad PROC` … `ad ENDP` ile tanımlanır (Bölüm 7: Prosedürler).



Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
▶  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
10 mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

AL = 5.

Adım 1 / 11

DEĞİŞKENLER

AX = **0005h** BL = **00h** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
▶  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
10 mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

`call kare` : sonraki komutun adresi (**0105h**) yığına atılır, SP = FFFCh, IP = kare (010Fh).

Adım 2 / 11

DEĞİŞKENLER

AX = **0005h** BL = **00h** SP = **FFFCh**

ÇAĞRI YIĞINI

FFFCh: 0105h

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
▶  mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

`mul al` : AX = 5 × 5 = **0019h** (25).

Adım 3 / 11

DEĞİŞKENLER

AX = **0019h** | BL = **00h** | SP = **FFFCh**

ÇAĞRI YIĞINI

FFFCh: 0105h

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
10 mul al             ; AX = AL × AL
▶  ret                ; dönüş adresini çek
```

`ret` : yığından 0105h çekilir → IP = 0105h, SP = FFFEh.

Adım 4 / 11

DEĞİŞKENLER

AX = **0019h** BL = **00h** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
▶  mov bl, al
5  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
10 mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

BL = 19h (25) saklanır.

Adım 5 / 11

DEĞİŞKENLER

AX = **0019h** | BL = **19h** | SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
▶  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
10 mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

AL = 7.

Adım 6 / 11

DEĞİŞKENLER

AX = **0007h** | BL = **19h** | SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
▶  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
10 mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

İkinci çağrı: dönüş adresi bu kez **010Ch**.

Adım 7 / 11

DEĞİŞKENLER

AX = **0007h** BL = **19h** SP = **FFFCh**

ÇAĞRI YIĞINI

FFFCh: 010Ch

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
▶  mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

AX = 7 × 7 = **0031h** (49).

Adım 8 / 11

DEĞİŞKENLER

AX = **0031h** | BL = **19h** | SP = **FFFCh**

ÇAĞRI YIĞINI

FFFCh: 010Ch

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
8  ret
9  kare:
10 mul al             ; AX = AL × AL
▶  ret                ; dönüş adresini çek
```

`ret` : IP = 010Ch, SP = FFFEh.

Adım 9 / 11

DEĞİŞKENLER

AX = **0031h** BL = **19h** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
6  call kare
▶ add bl, al          ; 25 + 49
8  ret
9  kare:
10 mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

BL = 19h + 31h = **4Ah = 74**.

Adım 10 / 11

DEĞİŞKENLER

AX = **0031h** | BL = **4Ah** | SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Kod İzleme: CALL / RET ve Dönüş Adresi

```
1  org 100h
2  mov al, 5
3  call kare          ; yığına dönüş adresi
4  mov bl, al
5  mov al, 7
6  call kare
7  add bl, al         ; 25 + 49
▶  ret
9  kare:
10 mul al             ; AX = AL × AL
11 ret                ; dönüş adresini çek
```

Ana program biter.

Adım 11 / 11

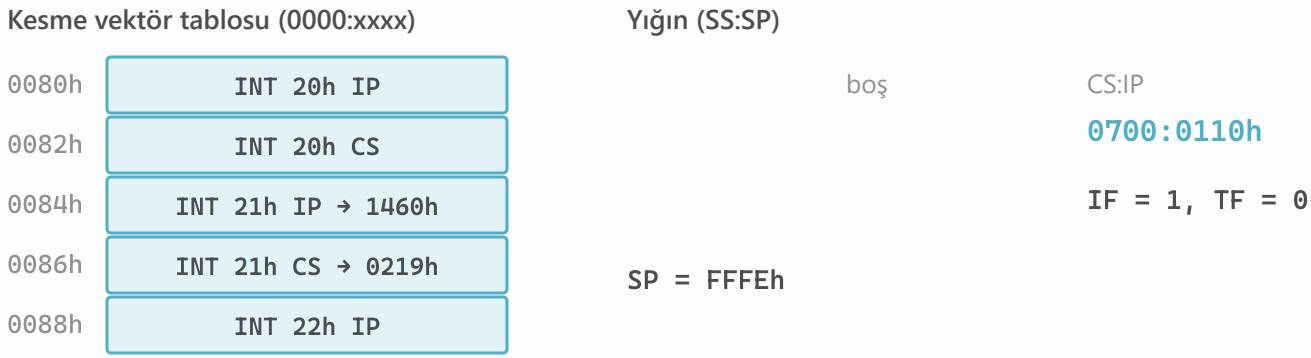
DEĞİŞKENLER

AX = **0031h** BL = **4Ah** SP = **FFFEh**

ÇAĞRI YIĞINI

boş

Yazılım Kesmesi: INT n



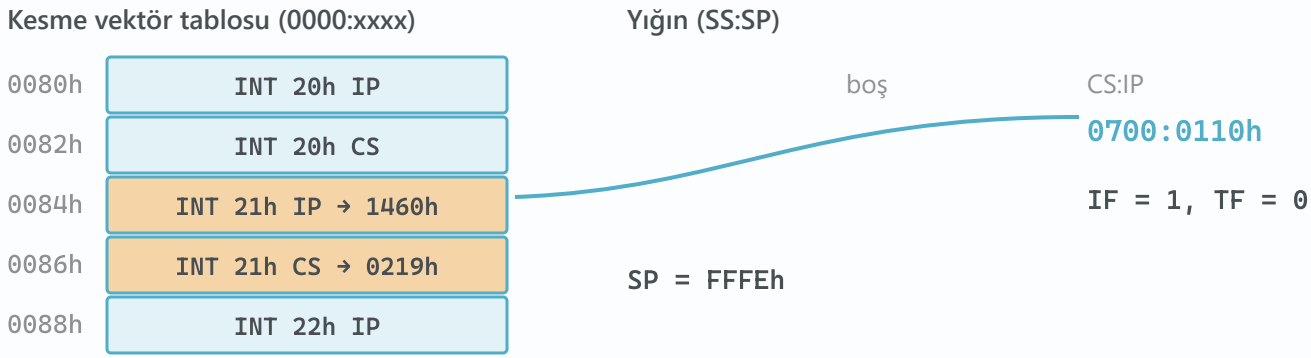
Başlangıç: **CS:IP = 0700:0110h** → `int 21h` (CD 21) yürütülmek üzere. SP = FFFEh.

Adım 1 / 6

`INT 3` tek baytlık (CCh) kesme noktası komutudur; `INTO`, OF = 1 ise INT 4 üretir. Tablodaki ISR adresi örnektir.

KONTROL AKTARMA

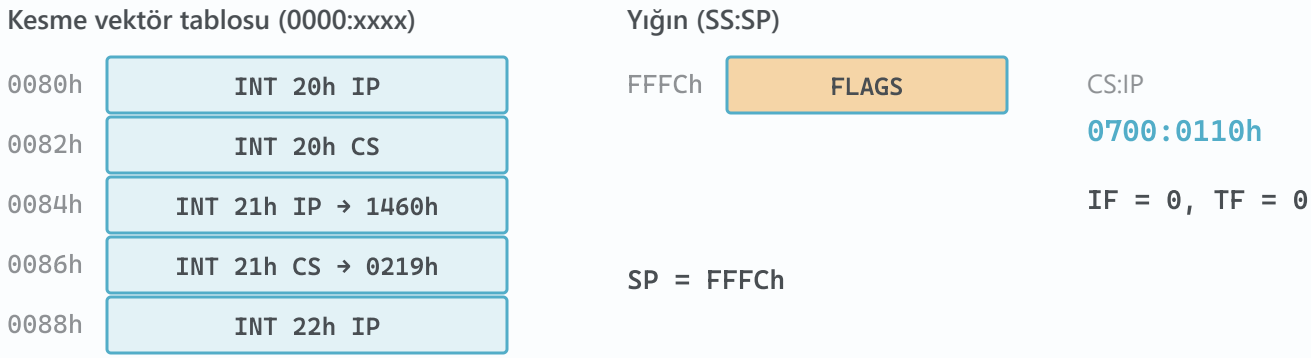
Yazılım Kesmesi: INT n



Vektör adresi = **21h × 4 = 0084h**. Kesme vektör tablosu 0000:0000–0000:03FFh arasındadır (256 × 4 bayt).

`INT 3` tek baytlık (CCh) kesme noktası komutudur; `INTO`, OF = 1 ise INT 4 üretir. Tablodaki ISR adresi örnektir.

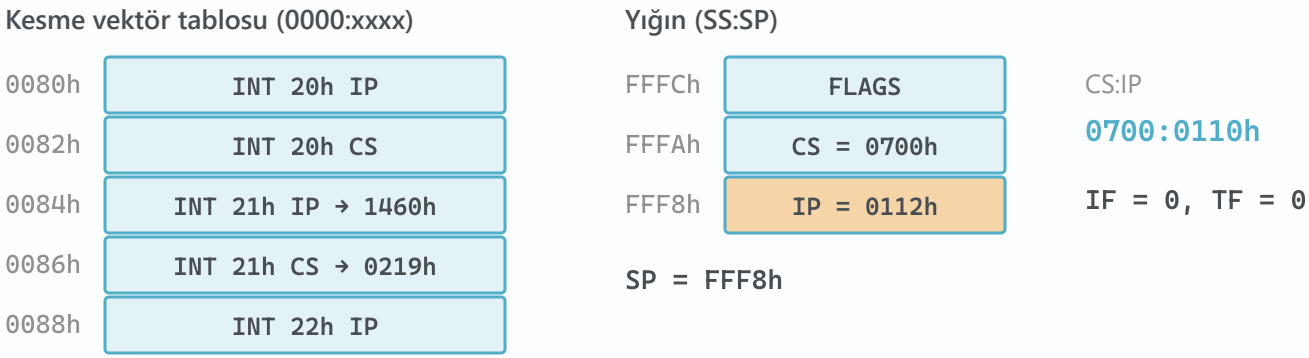
Yazılım Kesmesi: INT n



FLAGS yığına atılır; ardından **IF = 0, TF = 0** yapılır (kesme sırasında yeni maskelenebilir kesme ve tek adım kapalı).

`INT 3` tek baytlık (CCh) kesme noktası komutudur; `INTO`, OF = 1 ise INT 4 üretir. Tablodaki ISR adresi örnektir.

Yazılım Kesmesi: INT n



CS ve dönüş IP'si (**0112h**, INT'ten sonraki komut) yığına atılır. SP = FFF8h.

Adım 4 / 6

`INT 3` tek baytlık (CCh) kesme noktası komutudur; `INTO`, OF = 1 ise INT 4 üretir. Tablodaki ISR adresi örnektir.

KONTROL AKTARMA

Yazılım Kesmesi: INT n

Kesme vektör tablosu (0000:xxxx)

0080h	INT 20h IP
0082h	INT 20h CS
0084h	INT 21h IP → 1460h
0086h	INT 21h CS → 0219h
0088h	INT 22h IP

Yığın (SS:SP)

FFFCh	FLAGS
FFFAh	CS = 0700h
FFF8h	IP = 0112h

SP = FFF8h

CS:IP

0219:1460h

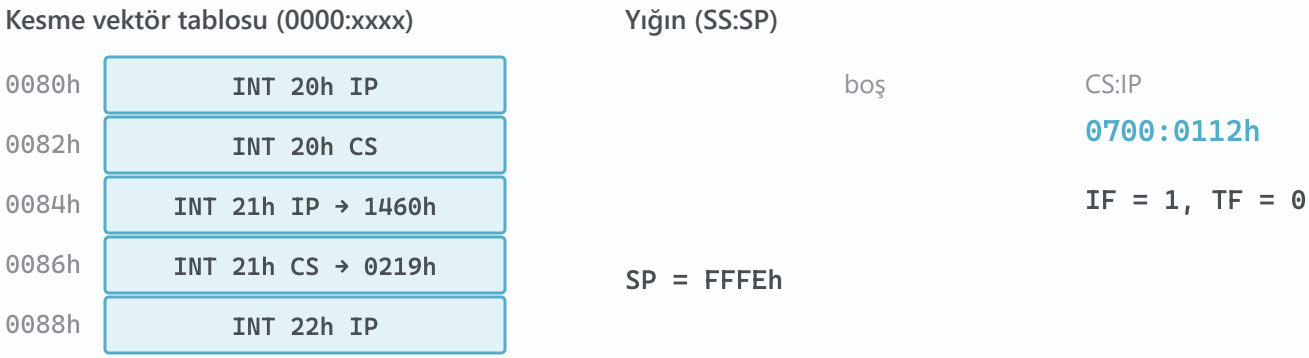
IF = 0, TF = 0

IP ← [0000:0084h], CS ← [0000:0086h]. Yürütme kesme hizmet yordamında (ISR) devam eder.

Adım 5 / 6

`INT 3` tek baytlık (CCh) kesme noktası komutudur; `INTO`, OF = 1 ise INT 4 üretir. Tablodaki ISR adresi örnektir.

Yazılım Kesmesi: INT n



ISR `iret` ile biter: IP, CS ve FLAGS ters sırada geri alınır; program 0700:0112h'den sürer.

`INT 3` tek baytlık (CCh) kesme noktası komutudur; `INTO`, OF = 1 ise INT 4 üretir. Tablodaki ISR adresi örnektir.

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 ETKİLERİ

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.

ÖZET

Özet

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

Veri taşıma
MOV, XCHG, PUSH/POP, XLAT, LEA/LDS/LES, IN/OUT bayrakları etkilemez; bellekten belleğe MOV yoktur.

Aritmetik
CF işaretsiz, OF işaretli taşma. MUL/DIV örtük AL/AX/DX kullanır; BCD için DAA…AAD.

Mantık / kaydırma
AND temizler, OR kurar, XOR tersler, TEST sınar. Kaydırma sayısı 1 ya da CL.

Dizgi
DS:SI → ES:DI, DF yönü, REP/REPE/REPNE ile CX kadar yineleme.

Denetim
JA/JB işaretsiz, JG/JL işaretli; Jcc ve LOOP kısa; CALL/INT dönüş adresini yığına atar.

ÖZET

Alıştırmalar

1. `mov dx, [bp+di+8]` komutunu kodlayın (opcode 100010, d = 1, w = 1).
2. AL = 96h, BL = 70h iken `add al, bl` sonrası AL, CF, OF, SF, ZF ne olur?
3. AX = 0FFFh iken `shl ax, 1`, `rcl dx, 1` (DX = 0, CF = 0) sonrası DX:AX?
4. AL = 'z' iken `xor al, 20h` ve ardından `and al, 0DFh` sonrası AL?
5. CX = 0 iken `loop` içeren bir döngü kaç kez döner? Nasıl önlenir?
6. AX = 0007h, BL = 02h için `div bl` ve AX = FFF9h için `idiv bl` sonuçları?
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**

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 10: Komut Kümesi

Sonraki bölüm: Bölüm 11: Kesme Fonksiyonları · Terim sözlüğü