Bölüm 6: Aritmetik, Mantık ve Kaydırma Komutları

Mikroişlemciler

Bayrak yazmacı · CF · ZF · SF · OF · PF · AF · Komut grupları · ADD · SUB · CMP · AND · OR · XOR · TEST · MUL · IMUL · DIV · IDIV · INC · DEC · NOT · NEG · SHL · SHR · SAR · ROL · RCR · Dizi toplama · BCD: AAA · AAS · Sıcaklık dönüşümü

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Bayraklar

Bir işlemin sonucuna göre CF, ZF, SF, OF, PF ve AF bayraklarının değerini hesaplamak; CF ile OF farkını açıklamak.

Komutlar

Toplama, çıkarma, karşılaştırma, mantıksal, çarpma, bölme, tek işlenenli, kaydırma ve döndürme komutlarını doğru işlenenlerle kullanmak; kaydırmanın CF'ye etkisini izlemek.

Çarpma / bölme

MUL, IMUL, DIV, IDIV'in örtük yazmaçlarını (AL, AX, DX:AX) ve bölme hatasını bilmek.

Uygulama

Dizi toplama, BCD düzeltme ve sıcaklık dönüşümü programlarını adım adım izlemek.

BAYRAK YAZMACI

Aritmetik ve Mantık Komutları

- Aritmetik ve mantık komutlarının sonuçları, işlemcinin **durum bayraklarını** etkiler.
- Bayraklar, 16 bitlik **FLAGS** yazmacında tutulur; her bit 0 ya da 1 olan bir bayraktır.
- 8086'da bu 16 bitin 9'u kullanılır: 6 **durum** bayrağı (sonucu anlatır) ve 3 **kontrol** bayrağı (işlemcinin davranışını yönetir).
- Koşullu atlamalar (`JZ` , `JC` , `JO` ...) bu bayraklara bakarak karar verir.



Kesik çizgili bitler 8086'da kullanılmaz.

Bayraklar Tablosu

Bit	Bayrak	Tür	1 olur, eğer...	Örnek / not
0	**CF** · Carry	durum	işaretsiz taşma: toplamada elde, çıkarmada ödünç	255 + 1 → AL = 00h, CF = 1; 7 − 9 → CF = 1; `STC`, `CLC`, `CMC`
2	**PF** · Parity	durum	sonucun **düşük baytında** çift sayıda 1 biti var	96h = 1001 0110b → PF = 1; sözcük sonuçta da yalnız düşük bayt
4	**AF** · Auxiliary	durum	bit 3'ten bit 4'e elde/ödünç	08h + 08h → AF = 1; yalnız AAA, AAS, DAA, DAS kullanır
6	**ZF** · Zero	durum	sonuç sıfır	`SUB AL, AL` → ZF = 1; `CMP AX, BX` sonrası ZF = 1 ise AX = BX
7	**SF** · Sign	durum	sonucun en anlamlı biti (MSB) 1	MSB'nin kopyası: 96h → SF = 1
8	**TF** · Trap	kontrol	her komuttan sonra INT 1	tek adım hata ayıklama
9	**IF** · Interrupt	kontrol	INTR kesmeleri kabul edilir	`STI` → 1, `CLI` → 0; NMI'yi etkilemez
10	**DF** · Direction	kontrol	dizgi komutları geriye doğru	0: SI/DI artar, 1: azalır; `CLD` / `STD`
11	**OF** · Overflow	durum	işaretli taşma: sonuç işaretli aralığa sığmadı	100 + 50 → AL = 96h (−106), OF = 1

BAYRAK YAZMACI

CF mi, OF mi?

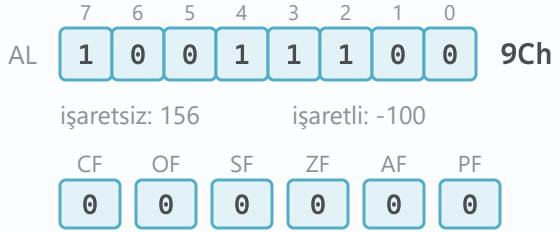
İşlemci toplamayı yalnızca bitler üzerinde yapar; **yorum programcınındır**. CF işaretsiz, OF işaretli yorumda sonucun doğru olup olmadığını söyler.

AL + işlenen	işaretsiz yorum	işaretli yorum	AL	CF	OF
`FFh + 01h`	255 + 1 = 256	-1 + 1 = 0	`00h`	**1**	**0**
`64h + 32h`	100 + 50 = 150	100 + 50 = 150	`96h`	**0**	**1**
`C8h + 64h`	200 + 100 = 300	-56 + 100 = 44	`2Ch`	**1**	**0**
`80h + 80h`	128 + 128 = 256	-128 + -128 = -256	`00h`	**1**	**1**
`05h + 0Ah`	5 + 10 = 15	5 + 10 = 15	`0Fh`	**0**	**0**

OF kuralı: aynı işaretli iki sayının toplamı farklı işaretli çıkarsa OF = 1. Farklı işaretli iki sayı toplanınca OF hiçbir zaman 1 olmaz.

Kod İzleme: CF, OF ve Diğer Bayraklar

```
▶ MOV  AL, 9Ch      ; 156 ya da -100
2 ADD  AL, 70h      ; + 112
3 MOV  AL, 50h      ; 80
4 ADD  AL, 50h      ; + 80
5 ADD  AL, AL       ; A0h + A0h
6 MOV  AL, 0FFh
7 INC  AL           ; CF'ye dokunmaz
8 CMC               ; CF tersine
9 RET
```



AL = 9Ch: işaretsiz 156, işaretli −100. Bayraklar başlangıçta 0; MOV onları değiştirmez.

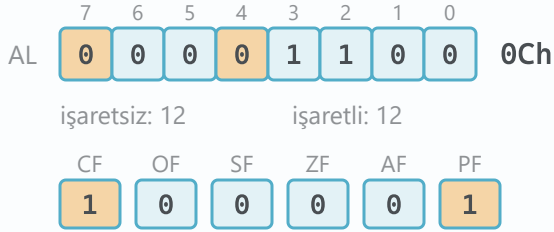
Adım 1 / 8

DEĞİŞKENLER

AL = 9Ch | CF = 0 | OF = 0 | ZF = 0

Kod İzleme: CF, OF ve Diğer Bayraklar

```
1 MOV  AL, 9Ch      ; 156 ya da -100
▶ ADD  AL, 70h      ; + 112
3 MOV  AL, 50h      ; 80
4 ADD  AL, 50h      ; + 80
5 ADD  AL, AL       ; A0h + A0h
6 MOV  AL, 0FFh
7 INC  AL           ; CF'ye dokunmaz
8 CMC               ; CF tersine
9 RET
```



156 + 112 = 268 > 255 → AL = 0Ch, **CF = 1**. İşaretli −100 + 112 = 12 doğru → OF = 0. 0Ch'de iki tane 1 → PF = 1.

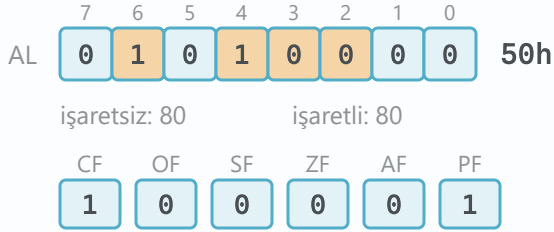
Adım 2 / 8

DEĞİŞKENLER

AL = 0Ch | CF = 1 | OF = 0 | ZF = 0

Kod İzleme: CF, OF ve Diğer Bayraklar

```
1 MOV  AL, 9Ch      ; 156 ya da -100
2 ADD  AL, 70h      ; + 112
▶ MOV  AL, 50h      ; 80
4 ADD  AL, 50h      ; + 80
5 ADD  AL, AL       ; A0h + A0h
6 MOV  AL, 0FFh
7 INC  AL           ; CF'ye dokunmaz
8 CMC               ; CF tersine
9 RET
```



AL = 50h (80). Bayraklar önceki ADD'den kalır.

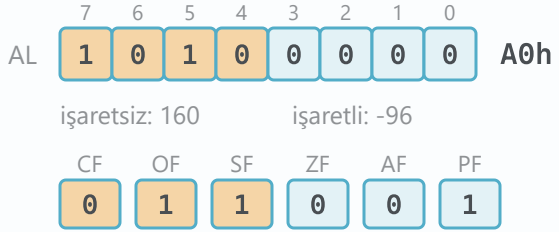
Adım 3 / 8

DEĞİŞKENLER

AL = 50h | CF = 1 | OF = 0 | ZF = 0

Kod İzleme: CF, OF ve Diğer Bayraklar

```
1 MOV  AL, 9Ch      ; 156 ya da -100
2 ADD  AL, 70h      ; + 112
3 MOV  AL, 50h      ; 80
▶ ADD  AL, 50h      ; + 80
5 ADD  AL, AL       ; A0h + A0h
6 MOV  AL, 0FFh
7 INC  AL           ; CF'ye dokunmaz
8 CMC               ; CF tersine
9 RET
```



80 + 80 = 160 işaretsizde doğru (CF = 0), ama iki pozitifin toplamı negatif çıktı (A0h = −96) → **OF = 1**, SF = 1.

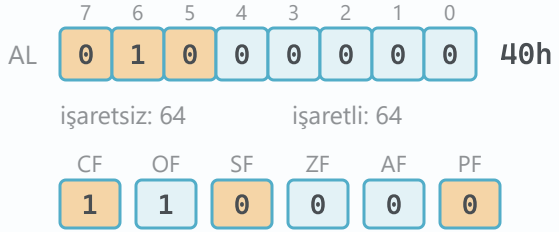
Adım 4 / 8

DEĞİŞKENLER

AL = **A0h** | CF = **0** | OF = **1** | ZF = **0**

Kod İzleme: CF, OF ve Diğer Bayraklar

```
1 MOV  AL, 9Ch      ; 156 ya da -100
2 ADD  AL, 70h      ; + 112
3 MOV  AL, 50h      ; 80
4 ADD  AL, 50h      ; + 80
▶ ADD  AL, AL       ; A0h + A0h
6 MOV  AL, 0FFh
7 INC  AL           ; CF'ye dokunmaz
8 CMC               ; CF tersine
9 RET
```



A0h + A0h: işaretsiz 320 (CF = 1), işaretli −96 + (−96) = −192 (OF = 1). AL = 40h: iki yorum da yanlış.

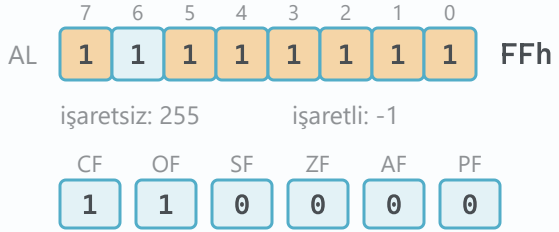
Adım 5 / 8

DEĞİŞKENLER

AL = 40h | CF = 1 | OF = 1 | ZF = 0

Kod İzleme: CF, OF ve Diğer Bayraklar

```
1 MOV  AL, 9Ch      ; 156 ya da -100
2 ADD  AL, 70h      ; + 112
3 MOV  AL, 50h      ; 80
4 ADD  AL, 50h      ; + 80
5 ADD  AL, AL       ; A0h + A0h
▶ MOV  AL, 0FFh
7 INC  AL           ; CF'ye dokunmaz
8 CMC               ; CF tersine
9 RET
```



AL = FFh. CF = 1 hâlâ duruyor.

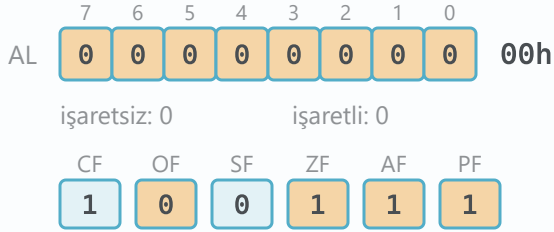
Adım 6 / 8

DEĞİŞKENLER

AL = **FFh** | CF = **1** | OF = **1** | ZF = **0**

Kod İzleme: CF, OF ve Diğer Bayraklar

```
1 MOV  AL, 9Ch      ; 156 ya da -100
2 ADD  AL, 70h      ; + 112
3 MOV  AL, 50h      ; 80
4 ADD  AL, 50h      ; + 80
5 ADD  AL, AL       ; A0h + A0h
6 MOV  AL, 0FFh
▶ INC  AL           ; CF'ye dokunmaz
8 CMC               ; CF tersine
9 RET
```



`INC AL` : FFh + 1 = 00h → ZF = 1, AF = 1; ama INC **CF'ye dokunmaz**: CF = 1 önceki toplamadan kaldı.

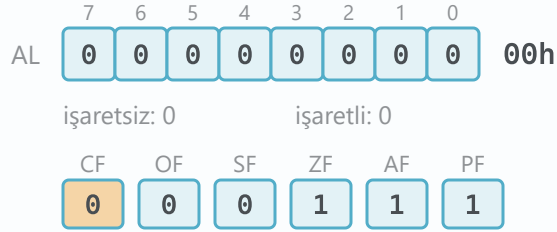
Adım 7 / 8

DEĞİŞKENLER

`AL = 00h` `CF = 1` `OF = 0` `ZF = 1`

Kod İzleme: CF, OF ve Diğer Bayraklar

```
1 MOV  AL, 9Ch      ; 156 ya da -100
2 ADD  AL, 70h      ; + 112
3 MOV  AL, 50h      ; 80
4 ADD  AL, 50h      ; + 80
5 ADD  AL, AL       ; A0h + A0h
6 MOV  AL, 0FFh
7 INC  AL           ; CF'ye dokunmaz
▶ CMC               ; CF tersine
9 RET
```



`CMC` CF'yi tersler: CF = 0. Diğer bayraklar değişmez.

Adım 8 / 8

DEĞİŞKENLER

AL = 00h | CF = 0 | OF = 0 | ZF = 1

ADD ile Bayraklar Adım Adım

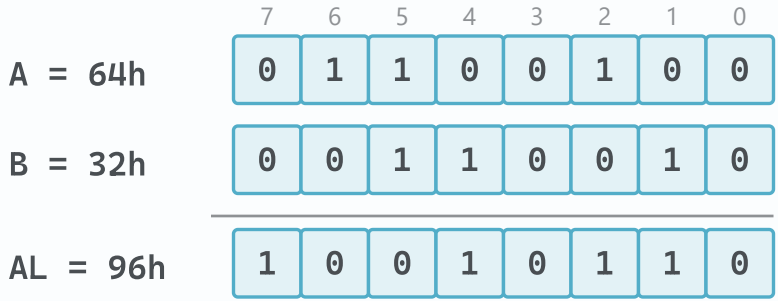


`MOV AL, 64h` (100) ve `ADD AL, 32h` (50). Bitler sağdan sola toplanır.

Adım 1 / 6

Sonuç: AL = 96h; CF = 0, ZF = 0, SF = 1, OF = 1, PF = 1, AF = 0.

ADD ile Bayraklar Adım Adım

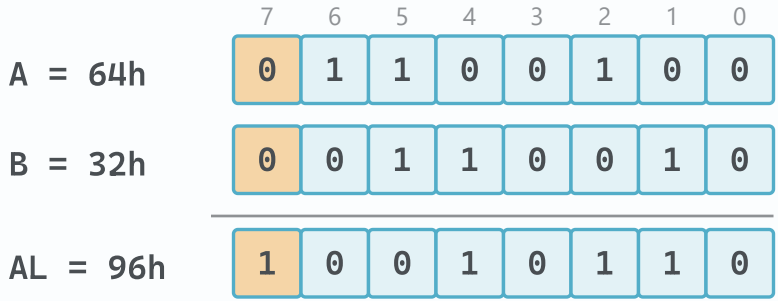


Sonuç **96h** = 1001 0110b. İşaretsiz yorumla 150 (doğru), işaretli yorumla −106 (yanlış!).

Adım 2 / 6

Sonuç: AL = 96h; CF = 0, ZF = 0, SF = 1, OF = 1, PF = 1, AF = 0.

ADD ile Bayraklar Adım Adım



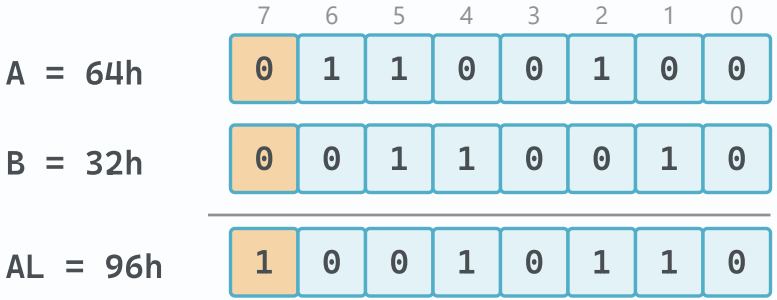
bit 7'den dışarı elde: 0

→ CF = 0

CF: en soldaki bitten dışarı elde çıktı mı? Hayır → CF = 0 (150 ≤ 255).

Sonuç: AL = 96h; CF = 0, ZF = 0, SF = 1, OF = 1, PF = 1, AF = 0.

ADD ile Bayraklar Adım Adım



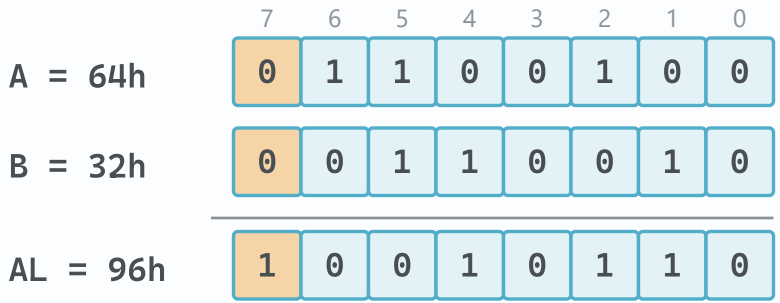
bit 7'ye giren elde 1, çıkan 0

→ OF = 1

OF: iki pozitif sayının toplamı negatif çıktı (bit 7'ye giren ve çıkan elde farklı) → OF = 1.

Sonuç: AL = 96h; CF = 0, ZF = 0, SF = 1, OF = 1, PF = 1, AF = 0.

ADD ile Bayraklar Adım Adım



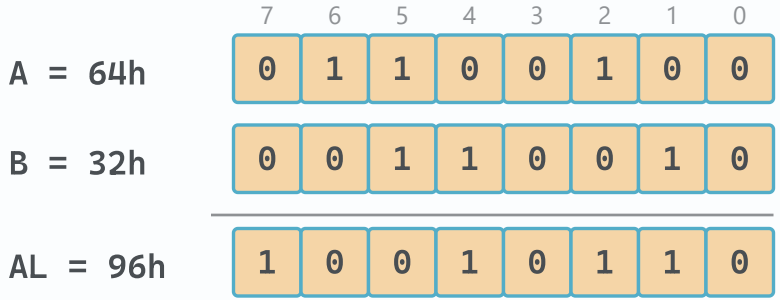
MSB = 1 → SF = 1

sonuç ≠ 0 → ZF = 0

SF = sonucun bit 7'si = 1. Sonuç sıfır değil: **ZF = 0**.

Sonuç: AL = 96h; CF = 0, ZF = 0, SF = 1, OF = 1, PF = 1, AF = 0.

ADD ile Bayraklar Adım Adım



dört tane 1 (çift) → PF = 1

4 + 2 nibble: elde yok → AF = 0

PF: 1001 0110b'de dört tane 1 → çift → PF = 1. **AF**: düşük nibble'lar 4 + 2 = 6, elde yok → AF = 0.

Sonuç: AL = 96h; CF = 0, ZF = 0, SF = 1, OF = 1, PF = 1, AF = 0.

Demo: Bayrak Hesaplayıcı

İki değer ve bir komut seçilince sonuç ve bayraklar hesaplanır. Değerler onluk (`-5`), onaltılık (`3Fh`) veya ikili (`1010b`) yazılabilir.

komut ADD boyut 8 bit hedef 100 kaynak 50

```
hedef  0110 0100  = 64h
kaynak 0011 0010  = 32h
sonuç  1001 0110  = 96h  (işaretsiz 150, işaretli -106)
```

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

Sonuç hedefe yazılır.

KOMUT GRUPLARI

Komutlar Üç Gruba Ayrılır

1. grup · iki işlenenli

`ADD` , `SUB` , `CMP` , `AND` , `TEST` , `OR` , `XOR`

hedef, kaynak

2. grup · çarpma ve bölme

`MUL` , `IMUL` , `DIV` , `IDIV`

tek işlenen + örtük AL / AX / DX

3. grup · tek işlenenli

`INC` , `DEC` , `NOT` , `NEG`

işlenen hem kaynak hem hedef

Artırma (INC) ve azaltma (DEC) **üçüncü grupta**dır. Gruplar, işlenen biçimine göre ayrılır.

hedef ← hedef ∘ kaynak

```
ADD AX, BX
```

AX ← AL × işlenen

```
MUL BL
```

işlenen ← f(işlenen)

```
INC CX
```

KOMUT GRUPLARI

Komutların Anlamı

1. grup	Ne yapar?
`ADD`	iki değeri toplar, sonucu hedefe yazar
`SUB`	kaynağı hedeften çıkarır, sonucu hedefe yazar
`CMP`	çıkarır ama sonucu **saklamaz**; yalnız bayraklar
`AND`	bit düzeyinde VE
`TEST`	bit düzeyinde VE; sonucu saklamaz
`OR`	bit düzeyinde VEYA
`XOR`	bit düzeyinde dışlayan VEYA

2. grup	Ne yapar?
`MUL`	işaretsiz çarpma
`IMUL`	işaretli çarpma
`DIV`	işaretsiz bölme (bölüm + kalan)
`IDIV`	işaretli bölme

3. grup	Ne yapar?
`INC`	1 artırır
`DEC`	1 azaltır
`NOT`	tüm bitleri tersler
`NEG`	işaretini değiştirir (ikiye tümleyen)

GRUP 1: İKİ İŞLENENLİ

ADD, SUB, CMP, AND, TEST, OR, XOR: İşlenenler

Biçim	Örnek
REG, memory	`ADD AX, [BX]`
memory, REG	`SUB var1, CL`
REG, REG	`XOR AX, AX`
memory, immediate	`AND byte ptr [SI], 0Fh`
REG, immediate	`CMP AL, 'A'`

- **REG**: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
- **memory**: `[BX]`, `[BX+SI+7]`, değişken ...
- **immediate**: `5`, `-24`, `3Fh`, `10001101b` ...

Geçersiz: `ADD [BX], [SI]` (iki bellek), `ADD AL, BX` (boyut farkı), `ADD 5, AL` (hedef sabit), `ADD DS, AX` (kesim yazmacı).

GRUP 1: İKİ İŞLENENLİ

Kurallar ve Etkilenen Bayraklar

- İki işlenenli işlemlerden sonra sonuç her zaman **ilk işlenende** (hedef) saklanır.
- `CMP` ve `TEST` yalnızca bayrakları etkiler, sonuç saklamaz.
- `ADD` / `SUB` : ikinci işleneni birinciye ekler / birinciden çıkarır.
- `CMP` : ikinci işleneni birinciden çıkarır, sonucu saklamaz.
- `AND` / `OR` / `XOR` : iki işlenenin karşılıklı bitleri arasında işlem yapar; `TEST` , AND gibidir ama yalnız bayrakları etkiler.

Komut	CF	ZF	SF	OF	PF	AF
`ADD` , `SUB` , `CMP`	✓	✓	✓	✓	✓	✓
`AND` , `OR` , `XOR` , `TEST`	0	✓	✓	0	✓	?

✓: sonuca göre ayarlanır · 0: her zaman temizlenir · ?: tanımsız. Mantıksal komutlar CF ve OF'yi daima 0 yapar.

Kod İzleme: ADD, SUB, CMP

```
▶ mov al, 0F0h      ; 240
2 add al, 20h       ; 240 + 32 = 272
3 mov bl, 7
4 sub bl, 9         ; 7 - 9
5 cmp al, 10h       ; AL - 10h
6 cmp bl, 0
7 add bl, 2         ; FEh + 2
```

AL = F0h. MOV bayrakları değiştirmez (başlangıçta hepsi 0).

Adım 1 / 7

DEĞİŞKENLER

AL = **F0h** | BL = **00h** | CF = **0** | ZF = **0**

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

Kod İzleme: ADD, SUB, CMP

```
1 mov al, 0F0h      ; 240
▶ add al, 20h       ; 240 + 32 = 272
3 mov bl, 7
4 sub bl, 9         ; 7 - 9
5 cmp al, 10h       ; AL - 10h
6 cmp bl, 0
7 add bl, 2         ; FEh + 2
```

F0h + 20h = 110h; 8 bite **10h** sığar, dışarı taşan 1 → **CF = 1**. İşaretli: −16 + 32 = 16 doğru → OF = 0.

Adım 2 / 7

DEĞİŞKENLER

AL = **10h** | BL = **00h** | CF = **1** | ZF = **0**

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

Kod İzleme: ADD, SUB, CMP

```
1 mov al, 0F0h      ; 240
2 add al, 20h       ; 240 + 32 = 272
▶ mov bl, 7
4 sub bl, 9         ; 7 - 9
5 cmp al, 10h       ; AL - 10h
6 cmp bl, 0
7 add bl, 2         ; FEh + 2
```

BL = 7. Bayraklar önceki ADD'den kalır.

Adım 3 / 7

DEĞİŞKENLER

AL = **10h** | BL = **07h** | CF = **1** | ZF = **0**

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

Kod İzleme: ADD, SUB, CMP

```
1 mov al, 0F0h      ; 240
2 add al, 20h       ; 240 + 32 = 272
3 mov bl, 7
▶ sub bl, 9         ; 7 - 9
5 cmp al, 10h       ; AL - 10h
6 cmp bl, 0
7 add bl, 2         ; FEh + 2
```

7 − 9 = −2 = **FEh**. Ödünç alındı → **CF = 1**; MSB = 1 → SF = 1. 7 − 9'da düşük nibble ödünç → AF = 1.

Adım 4 / 7

DEĞİŞKENLER

AL = **10h** | BL = **FEh** | CF = **1** | ZF = **0**

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

Kod İzleme: ADD, SUB, CMP

```
1 mov al, 0F0h      ; 240
2 add al, 20h       ; 240 + 32 = 272
3 mov bl, 7
4 sub bl, 9         ; 7 - 9
▶ cmp al, 10h       ; AL - 10h
6 cmp bl, 0
7 add bl, 2         ; FEh + 2
```

`CMP AL, 10h` : 10h − 10h = 0 → **ZF = 1**. AL **değişmedi**: eşitlik böyle sınanır.

Adım 5 / 7

DEĞİŞKENLER

AL = **10h** | BL = **FEh** | CF = **0** | ZF = **1** | SF = **0** | OF = **0** | PF = **1** | AF = **0**

Kod İzleme: ADD, SUB, CMP

```
1 mov al, 0F0h      ; 240
2 add al, 20h       ; 240 + 32 = 272
3 mov bl, 7
4 sub bl, 9         ; 7 - 9
5 cmp al, 10h       ; AL - 10h
▶ cmp bl, 0
7 add bl, 2         ; FEh + 2
```

FEh − 0: ZF = 0, SF = 1, CF = 0. İşaretsiz FEh > 0, işaretli −2 < 0.

Adım 6 / 7

DEĞİŞKENLER

AL = **10h** | BL = **FEh** | CF = **0** | ZF = **0** | SF = **1** | OF = **0** | PF = **0** | AF = **0**

Kod İzleme: ADD, SUB, CMP

```
1 mov al, 0F0h      ; 240
2 add al, 20h       ; 240 + 32 = 272
3 mov bl, 7
4 sub bl, 9         ; 7 - 9
5 cmp al, 10h       ; AL - 10h
6 cmp bl, 0
▶ add bl, 2         ; FEh + 2
```

FEh + 2 = 100h → BL = **00h**, CF = 1, ZF = 1. İşaretli: −2 + 2 = 0 doğru → OF = 0.

Adım 7 / 7

DEĞİŞKENLER

AL = **10h** | BL = **00h** | CF = **1** | ZF = **1**

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

ADC ile 32 Bitlik Toplama

```
1 ; DX:AX = 0001FFFFh, CX:BX = 00020001h
▶ mov ax, 0FFFFh
3 mov dx, 0001h
4 mov bx, 0001h
5 mov cx, 0002h
6 add ax, bx      ; düşük sözcükler
7 adc dx, cx      ; yüksek + CF
```

32 bitlik ilk sayı DX:AX'e yüklenir: düşük sözcük FFFFh...

Adım 1 / 5

DEĞİŞKENLER

AX = **FFFFh** DX = **0000h** BX = **0000h**

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

SBB (subtract with borrow) aynı fikirle çok sözcüklü çıkarma yapar: önce SUB, sonra SBB.

ADC ile 32 Bitlik Toplama

```
1 ; DX:AX = 0001FFFFh, CX:BX = 00020001h
2 mov ax, 0FFFFh
▶ mov dx, 0001h
4 mov bx, 0001h
5 mov cx, 0002h
6 add ax, bx      ; düşük sözcükler
7 adc dx, cx      ; yüksek + CF
```

...yüksek sözcük 0001h → DX:AX = 0001FFFFh (131071).

Adım 2 / 5

DEĞİŞKENLER

AX = **FFFFh** | DX = **0001h** | BX = **0000h** | CX = **0000h** | CF = **0** | ZF = **0**

SBB (subtract with borrow) aynı fikirle çok sözcüklü çıkarma yapar: önce SUB, sonra SBB.

ADC ile 32 Bitlik Toplama

```
1 ; DX:AX = 0001FFFFh, CX:BX = 00020001h
2 mov ax, 0FFFFh
3 mov dx, 0001h
▶ mov bx, 0001h
▶ mov cx, 0002h
6 add ax, bx       ; düşük sözcükler
7 adc dx, cx       ; yüksek + CF
```

İkinci sayı CX:BX = 00020001h (131073).

Adım 3 / 5

DEĞİŞKENLER

AX = **FFFFh** | DX = **0001h** | BX = **0001h** | CX = **0002h** | CF = **0** | ZF = **0**

SBB (subtract with borrow) aynı fikirle çok sözcüklü çıkarma yapar: önce SUB, sonra SBB.

ADC ile 32 Bitlik Toplama

```
1 ; DX:AX = 0001FFFFh, CX:BX = 00020001h
2 mov ax, 0FFFFh
3 mov dx, 0001h
4 mov bx, 0001h
5 mov cx, 0002h
▶ add ax, bx      ; düşük sözcükler
7 adc dx, cx      ; yüksek + CF
```

FFFFh + 0001h = 10000h → AX = 0000h, **CF = 1**: yüksek sözcüğe taşınacak elde.

Adım 4 / 5

DEĞİŞKENLER

AX = **0000h** | DX = **0001h** | BX = **0001h** | CX = **0002h** | CF = **1** | ZF = **1**

SBB (subtract with borrow) aynı fikirle çok sözcüklü çıkarma yapar: önce SUB, sonra SBB.

ADC ile 32 Bitlik Toplama

```
1 ; DX:AX = 0001FFFFh, CX:BX = 00020001h
2 mov ax, 0FFFFh
3 mov dx, 0001h
4 mov bx, 0001h
5 mov cx, 0002h
6 add ax, bx      ; düşük sözcükler
▶ adc dx, cx      ; yüksek + CF
```

`ADC` = ADD + CF: 0001h + 0002h + 1 = **0004h**.
Sonuç DX:AX = 00040000h (262144) ✓

Adım 5 / 5

DEĞİŞKENLER

AX = **0000h** | DX = **0004h** | BX = **0001h**
CX = **0002h** | CF = **0** | ZF = **0**

SBB (subtract with borrow) aynı fikirle çok sözcüklü çıkarma yapar: önce SUB, sonra SBB.

GRUP 1: İKİ İŞLENENLİ · KODU ADIM ADIM ÇALIŞTIR

SBB ile 32 Bitlik Çıkarma

```
1 ; DX:AX = 00040000h, CX:BX = 00010001h
▶ mov ax, 0000h
▶ mov dx, 0004h
4 mov bx, 0001h
5 mov cx, 0001h
6 sub ax, bx      ; düşük sözcükler
7 sbb dx, cx      ; yüksek - CF
```

Birinci sayı DX:AX = 00040000h (262144): yüksek sözcük DX, düşük sözcük AX.

Adım 1 / 4

DEĞİŞKENLER

AX = 0000h | DX = 0004h | BX = 0000h | CX = 0000h | CF = 0 | ZF = 0

Kural: en düşük sözcükte `SUB`, sonraki her sözcükte `SBB`. Aradaki komutlar CF'yi bozmamalıdır (MOV, INC, LOOP CF'yi korur).

GRUP 1: İKİ İŞLENENLİ · KODU ADIM ADIM ÇALIŞTIR

SBB ile 32 Bitlik Çıkarma

```
1 ; DX:AX = 00040000h, CX:BX = 00010001h
2 mov ax, 0000h
3 mov dx, 0004h
▶ mov bx, 0001h
▶ mov cx, 0001h
6 sub ax, bx      ; düşük sözcükler
7 sbb dx, cx      ; yüksek - CF
```

Çıkarılan CX:BX = 00010001h (65537).

Adım 2 / 4

DEĞİŞKENLER

AX = 0000h | DX = 0004h | BX = 0001h | CX = 0001h | CF = 0 | ZF = 0

Kural: en düşük sözcükte `SUB`, sonraki her sözcükte `SBB`. Aradaki komutlar CF'yi bozmamalıdır (MOV, INC, LOOP CF'yi korur).

GRUP 1: İKİ İŞLENENLİ · KODU ADIM ADIM ÇALIŞTIR

SBB ile 32 Bitlik Çıkarma

```
1 ; DX:AX = 00040000h, CX:BX = 00010001h
2 mov ax, 0000h
3 mov dx, 0004h
4 mov bx, 0001h
5 mov cx, 0001h
▶ sub ax, bx      ; düşük sözcükler
7 sbb dx, cx      ; yüksek - CF
```

0000h − 0001h: düşük sözcük yetmez, **ödünç** alınır → AX = **FFFFh**, **CF = 1**.

Adım 3 / 4

DEĞİŞKENLER

AX = FFFFh | DX = 0004h | BX = 0001h | CX = 0001h | CF = 1 | ZF = 0

Kural: en düşük sözcükte `SUB`, sonraki her sözcükte `SBB`. Aradaki komutlar CF'yi bozmamalıdır (MOV, INC, LOOP CF'yi korur).

SBB ile 32 Bitlik Çıkarma

```
1 ; DX:AX = 00040000h, CX:BX = 00010001h
2 mov ax, 0000h
3 mov dx, 0004h
4 mov bx, 0001h
5 mov cx, 0001h
6 sub ax, bx      ; düşük sözcükler
▶ sbb dx, cx      ; yüksek - CF
```

`SBB` = SUB − CF: 0004h − 0001h − 1 = **0002h**.
Sonuç DX:AX = 0002FFFFh (196607) ✓

Adım 4 / 4

DEĞİŞKENLER

AX = **FFFFh** | DX = **0002h** | BX = **0001h**
CX = **0001h** | CF = **0** | ZF = **0**

Kural: en düşük sözcükte `SUB`, sonraki her sözcükte `SBB`. Aradaki komutlar CF'yi bozmamalıdır (MOV, INC, LOOP CF'yi korur).

Mantıksal İşlem Kuralları

AND (VE)

A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

İkisi de 1 ise 1.

OR (VEYA)

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

En az biri 1 ise 1.

XOR (dışlayan)

A	B	A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

Farklı ise 1.

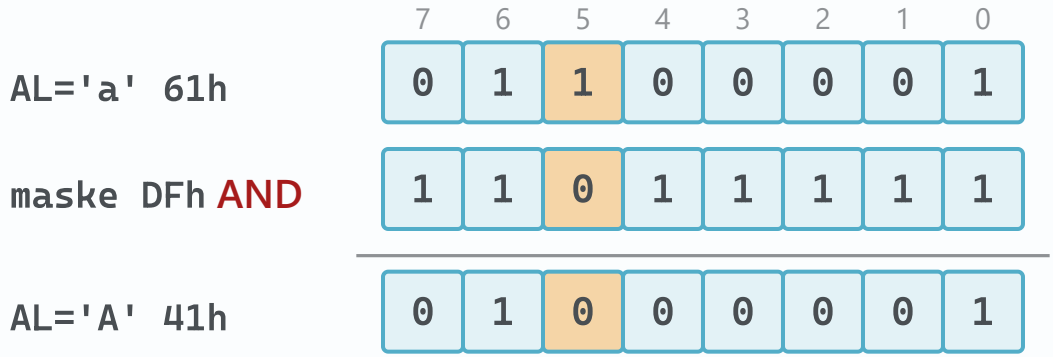
NOT (değil)

A	NOT A
1	0
0	1

Tersler. Tek işlenenli.

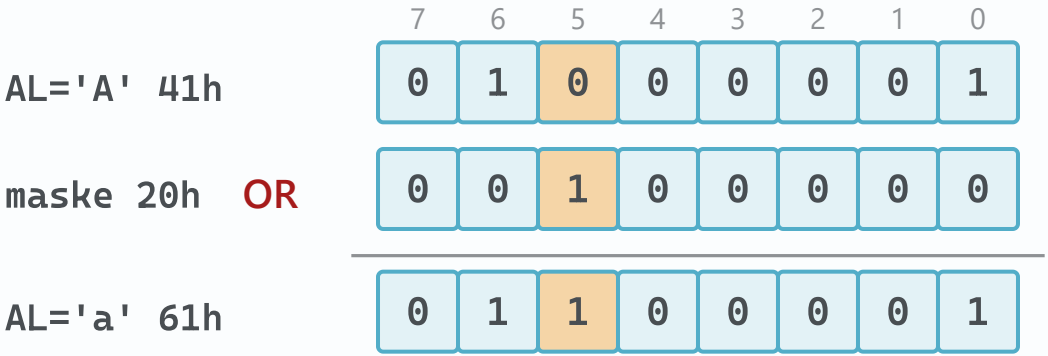
Bu kurallar 8 ya da 16 bitin **her birine ayrı ayrı** uygulanır: `AND AL, BL` aslında 8 tane bağımsız VE işlemidir.

Maskeleme: AND, OR, XOR, TEST



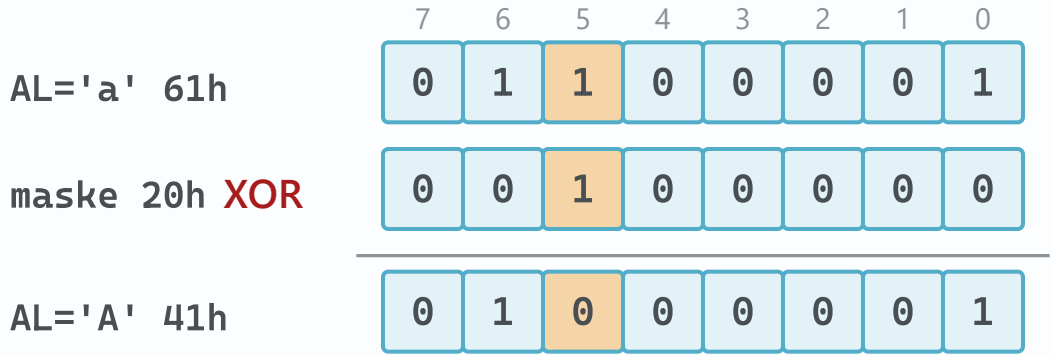
AND ile bit **temizleme**: maskede 0 olan bit 5 sıfırlanır, diğerleri korunur → küçük harf büyük olur.

Maskeleme: AND, OR, XOR, TEST



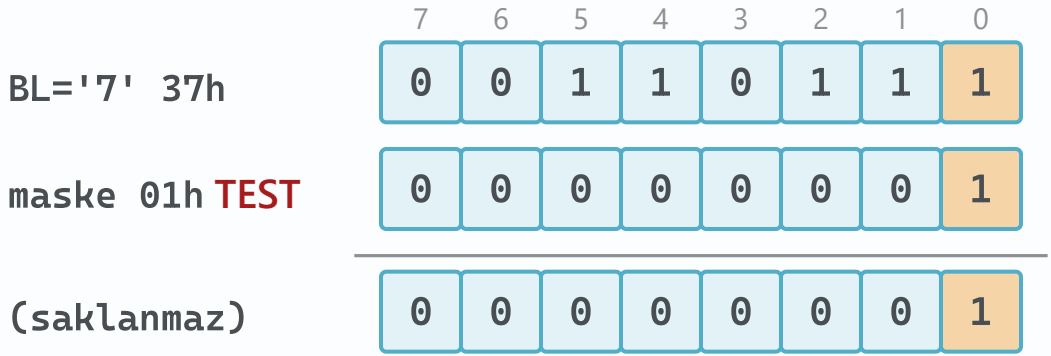
OR ile bit **kurma**: maskede 1 olan bit 5 kesin 1 olur → büyük harf küçük olur.

Maskeleme: AND, OR, XOR, TEST



XOR ile bit **tersleme**: maskede 1 olan bit değişir → harfin büyük/küçük hâli yer değiştirir.

Maskeleme: AND, OR, XOR, TEST



TEST ile bit **sınama**: sonuç 01h ≠ 0 → ZF = 0, yani bit 0 = 1, sayı tek. BL değişmez.

Sık Kullanılan Kalıplar

Kod	Etki	Neden?
`XOR AX, AX`	AX = 0, ZF = 1, CF = 0	`MOV AX, 0`'dan kısa (2 bayt); bayrakları da temizler
`AND AL, 0Fh`	'7' (37h) → 7	ASCII rakamı sayıya çevirir (yüksek nibble silinir)
`OR AL, 30h`	7 → '7' (37h)	sayıyı ASCII rakama çevirir
`AND AL, 0DFh`	'a' → 'A'	bit 5'i temizler: büyük harf
`OR AL, 20h`	'A' → 'a'	bit 5'i kurar: küçük harf
`TEST AL, 1`	ZF = 0 ise tek	AL değişmeden bit 0 sınanır
`TEST AL, 80h`	ZF = 0 ise negatif	işaret biti sınanır
`OR AX, AX`	AX değişmez	ZF/SF ayarlanır: AX sıfır mı, negatif mi?

Kod İzleme: AND, OR, XOR, TEST

```
▶ mov al, 'a'      ; 61h
2 and al, 0DFh     ; büyük harf
3 or  al, 20h      ; küçük harf
4 xor al, 20h      ; tersle
5 mov bl, '7'      ; 37h
6 and bl, 0Fh      ; sayı 7
7 test bl, 1       ; tek mi?
8 xor bl, bl       ; BL = 0
```

AL = 'a' = 61h = 0110 0001b.

Adım 1 / 8

DEĞİŞKENLER

AL = **61h** | BL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **0**

Kod İzleme: AND, OR, XOR, TEST

```
1 mov al, 'a'      ; 61h
▶ and al, 0DFh     ; büyük harf
3 or  al, 20h      ; küçük harf
4 xor al, 20h      ; tersle
5 mov bl, '7'      ; 37h
6 and bl, 0Fh      ; sayı 7
7 test bl, 1       ; tek mi?
8 xor bl, bl       ; BL = 0
```

61h AND DFh = **41h** = 'A'. 41h'de iki tane 1 → PF = 1. CF = OF = 0.

Adım 2 / 8

DEĞİŞKENLER

AL = **41h** | BL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **1**

Kod İzleme: AND, OR, XOR, TEST

```
1 mov al, 'a'      ; 61h
2 and al, 0DFh     ; büyük harf
▶ or  al, 20h      ; küçük harf
4 xor al, 20h      ; tersle
5 mov bl, '7'      ; 37h
6 and bl, 0Fh      ; sayı 7
7 test bl, 1       ; tek mi?
8 xor bl, bl       ; BL = 0
```

41h OR 20h = **61h** = 'a'. Üç tane 1 → PF = 0.

Adım 3 / 8

DEĞİŞKENLER

AL = **61h** | BL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **0**

Kod İzleme: AND, OR, XOR, TEST

```
1 mov al, 'a'      ; 61h
2 and al, 0DFh     ; büyük harf
3 or  al, 20h      ; küçük harf
▶ xor al, 20h      ; tersle
5 mov bl, '7'      ; 37h
6 and bl, 0Fh      ; sayı 7
7 test bl, 1       ; tek mi?
8 xor bl, bl       ; BL = 0
```

61h XOR 20h = **41h** = 'A' (bit 5 terslendi).

Adım 4 / 8

DEĞİŞKENLER

AL = **41h** | BL = **00h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **1**

Kod İzleme: AND, OR, XOR, TEST

```
1 mov al, 'a'      ; 61h
2 and al, 0DFh     ; büyük harf
3 or  al, 20h      ; küçük harf
4 xor al, 20h      ; tersle
▶ mov bl, '7'      ; 37h
6 and bl, 0Fh      ; sayı 7
7 test bl, 1       ; tek mi?
8 xor bl, bl       ; BL = 0
```

BL = '7' = 37h (ASCII rakam).

Adım 5 / 8

DEĞİŞKENLER

AL = **41h** | BL = **37h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **1**

Kod İzleme: AND, OR, XOR, TEST

```
1 mov al, 'a'      ; 61h
2 and al, 0DFh     ; büyük harf
3 or  al, 20h      ; küçük harf
4 xor al, 20h      ; tersle
5 mov bl, '7'      ; 37h
▶ and bl, 0Fh      ; sayı 7
7 test bl, 1       ; tek mi?
8 xor bl, bl       ; BL = 0
```

37h AND 0Fh = **07h**: artık sayısal 7.

Adım 6 / 8

DEĞİŞKENLER

AL = **41h** | BL = **07h** | CF = **0** | ZF = **0**

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

Kod İzleme: AND, OR, XOR, TEST

```
1 mov al, 'a'       ; 61h
2 and al, 0DFh      ; büyük harf
3 or  al, 20h       ; küçük harf
4 xor al, 20h       ; tersle
5 mov bl, '7'       ; 37h
6 and bl, 0Fh       ; sayı 7
▶ test bl, 1        ; tek mi?
8 xor bl, bl        ; BL = 0
```

07h AND 01h = 01h ≠ 0 → **ZF = 0**: sayı tek. BL aynı kaldı.

Adım 7 / 8

DEĞİŞKENLER

AL = **41h** | BL = **07h** | CF = **0** | ZF = **0** | SF = **0** | OF = **0** | PF = **0**

Kod İzleme: AND, OR, XOR, TEST

```
1 mov al, 'a'       ; 61h
2 and al, 0DFh      ; büyük harf
3 or  al, 20h       ; küçük harf
4 xor al, 20h       ; tersle
5 mov bl, '7'       ; 37h
6 and bl, 0Fh       ; sayı 7
7 test bl, 1        ; tek mi?
▶ xor bl, bl        ; BL = 0
```

Kendisiyle XOR → **00h**, ZF = 1, PF = 1 (sıfır tane 1, çift).

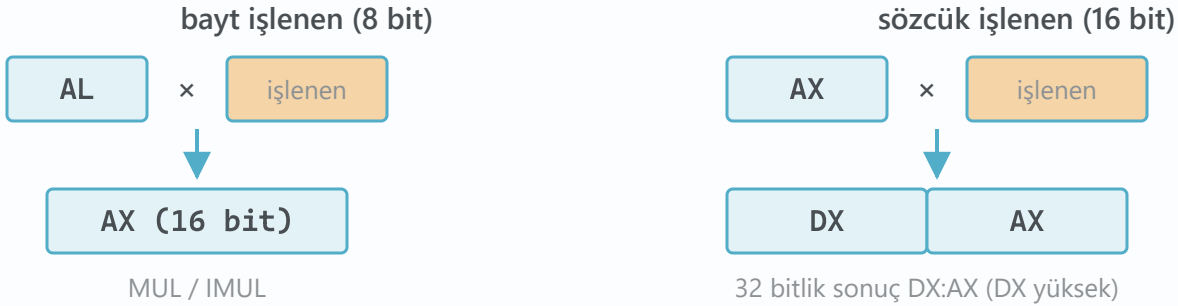
Adım 8 / 8

DEĞİŞKENLER

AL = **41h** | BL = **00h** | CF = **0** | ZF = **1** | SF = **0** | OF = **0** | PF = **1**

GRUP 2: ÇARPMA VE BÖLME

MUL ve IMUL



Komut	İşlenen bayt ise	İşlenen sözcük ise
`MUL` (işaretsiz)	AX = AL × işlenen	(DX AX) = AX × işlenen
`IMUL` (işaretli)	AX = AL × işlenen	(DX AX) = AX × işlenen

İşlenen: REG veya memory (`MUL BL` , `IMUL word ptr [SI]`). 8086'da **sabit işlenen yoktur**: `MUL 5` geçersizdir.

GRUP 2: ÇARPMA VE BÖLME

DIV ve IDIV

Komut	İşlenen bayt ise	İşlenen sözcük ise
`DIV` (işaretsiz)	AL = AX / işlenen AH = kalan (modulus)	AX = (DX AX) / işlenen DX = kalan (modulus)
`IDIV` (işaretli)	AL = AX / işlenen AH = kalan	AX = (DX AX) / işlenen DX = kalan



Bayraklar

MUL / IMUL yalnızca CF ve OF'yi anlamlı ayarlar: sonuç işlenen boyutunu aşarsa (üst yarı gerekliyse) CF = OF = 1. Diğerleri tanımsız.

DIV / IDIV sonrası tüm bayraklar tanımsızdır.

Bölme hatası

Bölen 0 ise ya da bölüm hedefe (AL / AX) sığmıyorsa işlemci **INT 0** üretir (bölme hatası).

IDIV bölümü sıfıra doğru keser; kalan bölünenle aynı işaretlidir.

GRUP 2: ÇARPMA VE BÖLME

Çarpma ve Bölme Adım Adım

AL C8h BL 03h

`MOV AL, 200`, `MOV BL, 3`. 200 = C8h, işaretli yorumda −56.

Çarpma ve Bölme Adım Adım

AX	CF	OF
0258h	1	1

`MUL BL` : 200 × 3 = 600 = **0258h** → AX. Sonuç AL'ye sığmadı (AH ≠ 0) → **CF = OF = 1**.

Çarpma ve Bölme Adım Adım

AX	CF	OF
FF58h	1	1

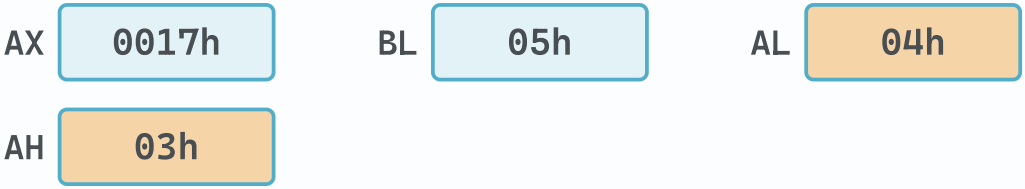
Aynı bitlerle `IMUL BL`: −56 × 3 = −168 = **FF58h**. Farklı sonuç! −168 bayta sığmaz → CF = OF = 1.

Çarpma ve Bölme Adım Adım

AX	CF	OF
FFF6h	0	0

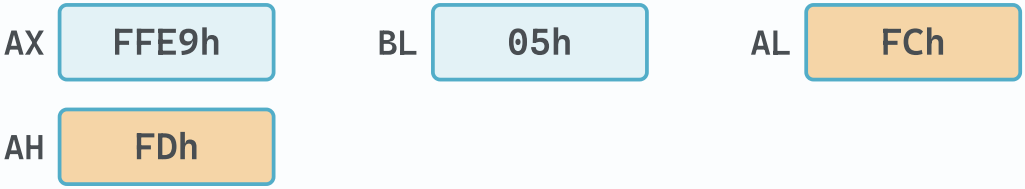
AL = FEh (−2), BL = 5: `IMUL BL` → −10 = **FFF6h**. AH yalnızca AL'nin işaret uzantısı → **CF = OF = 0**: sonuç AL'ye sığdı.

Çarpma ve Bölme Adım Adım



`DIV BL` : AX = 23, BL = 5 → bölüm **AL = 4**, kalan **AH = 3**. Yeni AX = 0304h.

Çarpma ve Bölme Adım Adım



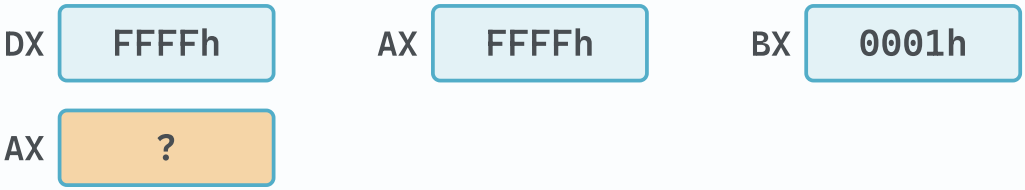
`IDIV BL` : −23 / 5 → bölüm **AL = FCh (−4)**, kalan **AH = FDh (−3)**. Kontrol: −4 × 5 + (−3) = −23.

Çarpma ve Bölme Adım Adım

AX	1000h	BL	02h	AL	?

AX = 1000h (4096), `DIV BL` (2): bölüm 2048 AL'ye sığmaz → **bölme hatası, INT 0**. Sözcük bölme (DX:AX / BX) gerekirdi.

Çarpma ve Bölme Adım Adım



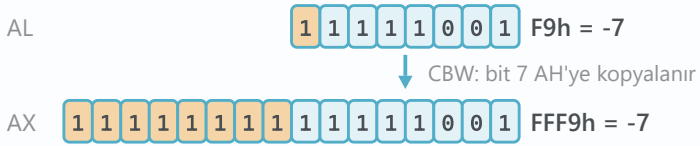
DX:AX = FFFFFFFFh, `DIV BX` (BX = 1): bölen sıfır değil, ama bölüm FFFFFFFFh AX'e sığmaz → yine **INT 0**. Kural: sözcük bölmede DX ≥ bölen ise bölüm sığmaz (bayt bölmede AH ≥ bölen).

Bölmeye Hazırlık: CBW, CWD

- Bölme, işlenenin **iki katı** genişlikte bir bölünen ister: bayt bölmede AX, sözcük bölmede DX:AX.
- **İşaretsiz** bölmeden önce üst yarı sıfırlanır: `MOV AH, 0` veya `XOR DX, DX`.
- **İşaretli** bölmeden önce işaret genişletilir: `CBW` (AL → AX), `CWD` (AX → DX:AX).
- DX'i unutmak klasik hatadır: DX'te kalan eski değer bölüneni büyütür ve bölme hatasına yol açabilir.

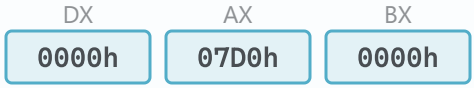
```
; işaretsiz: 1000 / 7
mov ax, 1000
xor dx, dx      ; DX:AX = 1000
mov bx, 7
div bx          ; AX = 142, DX = 6

; işaretli: -1000 / 7
mov ax, -1000
cwd             ; DX = FFFFh
idiv bx         ; AX = -142, DX = -6
```



Kod İzleme: DIV, CWD ve IDIV

```
▶ MOV  AX, 2000
2 XOR  DX, DX       ; DX:AX = 2000
3 MOV  BX, 9
4 DIV  BX           ; işaretsiz bölme
5 MOV  AX, -2000
6 CWD               ; işaret genişlet
7 IDIV BX           ; işaretli bölme
8 MOV  AX, 2000     ; DX temizlenmedi!
9 DIV  BX
10 RET
```



DX:AX = 000007D0h

işaretsiz 2000 · işaretli 2000

AX = 2000 = 07D0h. Sözcük bölmede bölünen DX:AX'tir; DX'te ne olduğu da önemlidir.

Adım 1 / 9

DEĞİŞKENLER

AX = **07D0h** DX = **0000h** BX = **0000h**

Kod İzleme: DIV, CWD ve IDIV

```
1  MOV  AX, 2000
▶ XOR  DX, DX       ; DX:AX = 2000
3  MOV  BX, 9
4  DIV  BX          ; işaretsiz bölme
5  MOV  AX, -2000
6  CWD              ; işaret genişlet
7  IDIV BX          ; işaretli bölme
8  MOV  AX, 2000    ; DX temizlenmedi!
9  DIV  BX
10 RET
```

DX	AX	BX
0000h	07D0h	0000h

DX:AX = 000007D0h

işaretsiz 2000 · işaretli 2000

`XOR DX, DX` : işaretsiz bölme için üst yarı sıfır → DX:AX = 2000.

Adım 2 / 9

DEĞİŞKENLER

AX = **07D0h** DX = **0000h** BX = **0000h**

Kod İzleme: DIV, CWD ve IDIV

```
1  MOV  AX, 2000
2  XOR  DX, DX       ; DX:AX = 2000
▶  MOV  BX, 9
4  DIV  BX           ; işaretsiz bölme
5  MOV  AX, -2000
6  CWD                ; işaret genişlet
7  IDIV BX           ; işaretli bölme
8  MOV  AX, 2000     ; DX temizlenmedi!
9  DIV  BX
10 RET
```

DX	AX	BX
0000h	07D0h	0009h

DX:AX = 000007D0h

işaretsiz 2000 · işaretli 2000

Bölen BX = 9.

Adım 3 / 9

DEĞİŞKENLER

AX = **07D0h** DX = **0000h** BX = **0009h**

Kod İzleme: DIV, CWD ve IDIV

```
1  MOV  AX, 2000
2  XOR  DX, DX       ; DX:AX = 2000
3  MOV  BX, 9
▶ DIV  BX            ; işaretsiz bölme
5  MOV  AX, -2000
6  CWD               ; işaret genişlet
7  IDIV BX           ; işaretli bölme
8  MOV  AX, 2000     ; DX temizlenmedi!
9  DIV  BX
10 RET
```

DX	AX	BX
0002h	00DEh	0009h

DX:AX = 000200DEh

işaretsiz 131294 · işaretli 131294

bölüm AX = 222, kalan DX = 2

`DIV BX` : 2000 / 9 → AX = 222 (00DEh), DX = 2. Bayraklar tanımsız.

Adım 4 / 9

DEĞİŞKENLER

AX = 00DEh DX = 0002h BX = 0009h

Kod İzleme: DIV, CWD ve IDIV

```
1  MOV  AX, 2000
2  XOR  DX, DX       ; DX:AX = 2000
3  MOV  BX, 9
4  DIV  BX           ; işaretsiz bölme
▶  MOV  AX, -2000
6  CWD               ; işaret genişlet
7  IDIV BX           ; işaretli bölme
8  MOV  AX, 2000     ; DX temizlenmedi!
9  DIV  BX
10 RET
```

DX	AX	BX
0002h	F830h	0009h

DX:AX = 0002F830h

işaretsiz 194608 · işaretli 194608

AX = −2000 = F830h. DX'te önceki kalan (2) duruyor: DX:AX şimdi yanlış bir bölünen.

Adım 5 / 9

DEĞİŞKENLER

AX = F830h | DX = 0002h | BX = 0009h

Kod İzleme: DIV, CWD ve IDIV

```
1  MOV  AX, 2000
2  XOR  DX, DX       ; DX:AX = 2000
3  MOV  BX, 9
4  DIV  BX           ; işaretsiz bölme
5  MOV  AX, -2000
▶  CWD               ; işaret genişlet
7  IDIV BX           ; işaretli bölme
8  MOV  AX, 2000     ; DX temizlenmedi!
9  DIV  BX
10 RET
```

DX	AX	BX
FFFFh	F830h	0009h

DX:AX = FFFFF830h

işaretsiz 4294965296 · işaretli -2000

`CWD` : AX'in işaret biti 1 → DX = FFFFh. DX:AX = FFFFF830h, 32 bitte −2000.

Adım 6 / 9

DEĞİŞKENLER

AX = **F830h** | DX = **FFFFh** | BX = **0009h**

Kod İzleme: DIV, CWD ve IDIV

```
1  MOV  AX, 2000
2  XOR  DX, DX       ; DX:AX = 2000
3  MOV  BX, 9
4  DIV  BX           ; işaretsiz bölme
5  MOV  AX, -2000
6  CWD               ; işaret genişlet
▶ IDIV BX            ; işaretli bölme
8  MOV  AX, 2000     ; DX temizlenmedi!
9  DIV  BX
10 RET
```

DX	AX	BX
FFFEh	FF22h	0009h

DX:AX = FFFEFF22h

işaretsiz 4294901538 · işaretli -65758

bölüm AX = -222, kalan DX = -2

`IDIV BX` : −2000 / 9 → AX = FF22h (−222), DX = FFFEh (−2). Kalan bölünenle aynı işaretli.

Adım 7 / 9

DEĞİŞKENLER

AX = **FF22h** DX = **FFFEh** BX = **0009h**

Kod İzleme: DIV, CWD ve IDIV

```
1  MOV  AX, 2000
2  XOR  DX, DX       ; DX:AX = 2000
3  MOV  BX, 9
4  DIV  BX           ; işaretsiz bölme
5  MOV  AX, -2000
6  CWD               ; işaret genişlet
7  IDIV BX           ; işaretli bölme
▶  MOV  AX, 2000     ; DX temizlenmedi!
9  DIV  BX
10 RET
```

DX	AX	BX
FFFEh	07D0h	0009h

DX:AX = FFFE07D0h

işaretsiz 4294838224 · işaretli -129072

AX = 2000, ama DX = FFFEh temizlenmedi: DX:AX = FFFE07D0h, yaklaşık 4,29 milyar.

Adım 8 / 9

DEĞİŞKENLER

AX = **07D0h** | DX = **FFFEh** | BX = **0009h**

Kod İzleme: DIV, CWD ve IDIV

```
1  MOV  AX, 2000
2  XOR  DX, DX       ; DX:AX = 2000
3  MOV  BX, 9
4  DIV  BX           ; işaretsiz bölme
5  MOV  AX, -2000
6  CWD                ; işaret genişlet
7  IDIV BX           ; işaretli bölme
8  MOV  AX, 2000     ; DX temizlenmedi!
▶  DIV  BX
10 RET
```

DX	AX	BX
FFFEh	07D0h	0009h

DX:AX = FFFE07D0h

işaretsiz 4294838224 · işaretli -129072

bölüm AX'e sığmıyor → INT 0

`DIV BX` : bölüm 16 bite sığmaz (DX ≥ bölen) → işlemci **INT 0** üretir, program durur. Çözüm: bölmeden önce `XOR DX, DX` .

Adım 9 / 9

DEĞİŞKENLER

AX = **07D0h** DX = **FFFEh** BX = **0009h**

İşaretli Taşma: JO ve CBW

```
▶ mov al, -92       ; A4h
▶ mov bl, -92
3  add al, bl       ; 8 bit toplama
4  jo  genislet     ; OF = 1 ise
5  jmp bitti
6  genislet:
7  mov al, -92      ; birinci sayı
8  cbw              ; AL -> AX
9  mov dx, ax
10 mov al, bl       ; ikinci sayı
11 cbw
12 add ax, dx       ; 16 bit toplama
13 bitti:
14 ret
```

AL = BL = A4h: −92 (256 − 92 = 164 = A4h).

Adım 1 / 6

DEĞİŞKENLER

AL = **A4h** | AX = **00A4h** | DX = **0000h** | CF = **0** | OF = **0** | SF = **0** | ZF = **0**

`JO` (jump if overflow) OF = 1 ise, `JNO` OF = 0 ise atlar. İşaretli bayt toplamı taşabiliyorsa işlenenler önce `CBW` ile sözcüğe genişletilir.

İşaretli Taşma: JO ve CBW

```
1  mov al, -92      ; A4h
2  mov bl, -92
▶  add al, bl       ; 8 bit toplama
4  jo  genislet     ; OF = 1 ise
5  jmp bitti
6  genislet:
7  mov al, -92      ; birinci sayı
8  cbw              ; AL -> AX
9  mov dx, ax
10 mov al, bl       ; ikinci sayı
11 cbw
12 add ax, dx       ; 16 bit toplama
13 bitti:
14 ret
```

A4h + A4h = 148h → AL = **48h** = +72. İki negatifin toplamı pozitif çıktı → **OF = 1** (CF = 1 de oldu ama işaretli yorumda önemsiz).

Adım 2 / 6

DEĞİŞKENLER

AL = **48h** | AX = **0048h** | DX = **0000h** | CF = **1** | OF = **1** | SF = **0** | ZF = **0**

`JO` (jump if overflow) OF = 1 ise, `JNO` OF = 0 ise atlar. İşaretli bayt toplamı taşabiliyorsa işlenenler önce `CBW` ile sözcüğe genişletilir.

İşaretli Taşma: JO ve CBW

```
1  mov al, -92      ; A4h
2  mov bl, -92
3  add al, bl       ; 8 bit toplama
▶  jo  genislet     ; OF = 1 ise
5  jmp bitti
6  genislet:
7  mov al, -92      ; birinci sayı
8  cbw              ; AL -> AX
9  mov dx, ax
10 mov al, bl       ; ikinci sayı
11 cbw
12 add ax, dx       ; 16 bit toplama
13 bitti:
14 ret
```

`JO` OF = 1 olduğu için `genislet` etiketine atlar. Doğru sonuç −184, −128 … 127 aralığında değil.

Adım 3 / 6

DEĞİŞKENLER

AL = **48h** | AX = **0048h** | DX = **0000h** | CF = **1** | OF = **1** | SF = **0** | ZF = **0**

`JO` (jump if overflow) OF = 1 ise, `JNO` OF = 0 ise atlar. İşaretli bayt toplamı taşabiliyorsa işlenenler önce `CBW` ile sözcüğe genişletilir.

İşaretli Taşma: JO ve CBW

```
1  mov al, -92      ; A4h
2  mov bl, -92
3  add al, bl       ; 8 bit toplama
4  jo  genislet     ; OF = 1 ise
5  jmp bitti
6  genislet:
▶  mov al, -92      ; birinci sayı
▶  cbw              ; AL -> AX
9  mov dx, ax
10 mov al, bl       ; ikinci sayı
11 cbw
12 add ax, dx       ; 16 bit toplama
13 bitti:
14 ret
```

AL = A4h, `CBW` : MSB = 1 → AH = FFh, AX = **FFA4h** (16 bitte −92).

Adım 4 / 6

DEĞİŞKENLER

AL = **A4h** | AX = **FFA4h** | DX = **0000h** | CF = **1** | OF = **1** | SF = **0** | ZF = **0**

`JO` (jump if overflow) OF = 1 ise, `JNO` OF = 0 ise atlar. İşaretli bayt toplamı taşabiliyorsa işlenenler önce `CBW` ile sözcüğe genişletilir.

İşaretli Taşma: JO ve CBW

```
1  mov al, -92      ; A4h
2  mov bl, -92
3  add al, bl       ; 8 bit toplama
4  jo  genislet     ; OF = 1 ise
5  jmp bitti
6  genislet:
7  mov al, -92      ; birinci sayı
8  cbw              ; AL -> AX
▶  mov dx, ax
▶  mov al, bl       ; ikinci sayı
▶  cbw
12 add ax, dx       ; 16 bit toplama
13 bitti:
14 ret
```

Birinci sayı DX'e saklanır; ikinci sayı (BL) da AL'ye alınıp genişletilir: AX = FFA4h.

Adım 5 / 6

DEĞİŞKENLER

AL = **A4h** | AX = **FFA4h** | DX = **FFA4h** | CF = **1** | OF = **1** | SF = **0** | ZF = **0**

`JO` (jump if overflow) OF = 1 ise, `JNO` OF = 0 ise atlar. İşaretli bayt toplamı taşabiliyorsa işlenenler önce `CBW` ile sözcüğe genişletilir.

İşaretli Taşma: JO ve CBW

```
1  mov al, -92      ; A4h
2  mov bl, -92
3  add al, bl       ; 8 bit toplama
4  jo  genislet     ; OF = 1 ise
5  jmp bitti
6  genislet:
7  mov al, -92      ; birinci sayı
8  cbw              ; AL -> AX
9  mov dx, ax
10 mov al, bl       ; ikinci sayı
11 cbw
▶ add ax, dx        ; 16 bit toplama
13 bitti:
14 ret
```

FFA4h + FFA4h = 1FF48h → AX = **FF48h** = −184 ✓. Sonuç 16 bite sığdı: **OF = 0**.

Adım 6 / 6

DEĞİŞKENLER

AL = **48h** | AX = **FF48h** | DX = **FFA4h** | CF = **1** | OF = **0** | SF = **1** | ZF = **0**

`JO` (jump if overflow) OF = 1 ise, `JNO` OF = 0 ise atlar. İşaretli bayt toplamı taşabiliyorsa işlenenler önce `CBW` ile sözcüğe genişletilir.

CBW ile İşaretli Bayt Toplamı

```
1  org 100h
2  jmp basla
3  dizi db 100, -30, 90, -7
4  toplam dw 0
5  basla:
▶  lea si, dizi
▶  mov cx, 4
▶  xor bx, bx       ; BX = 16 bit toplam
9  dongu:
10 mov al, [si]
11 cbw              ; AL -> AX
   (işaretle)
12 add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx   ; 153 = 0099h
16 ret
```

SI dizinin başını gösterir, CX = 4 eleman, BX = 0 (toplam sıfırlandı).

Adım 1 / 10

DEĞİŞKENLER

AL = **00h** | AX = **0000h** | BX = **0000h** | CX = **0004h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
1  org 100h
2  jmp basla
3  dizi db 100, -30, 90, -7
4  toplam dw 0
5  basla:
6  lea si, dizi
7  mov cx, 4
8  xor bx, bx      ; BX = 16 bit toplam
9  dongu:
▶  mov al, [si]
▶  cbw             ; AL -> AX (işaretle)
12 add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx  ; 153 = 0099h
16 ret
```

AL = 64h (+100). MSB = 0 → `CBW` AH = 00h, AX = 0064h.

Adım 2 / 10

DEĞİŞKENLER

AL = **64h** | AX = **0064h** | BX = **0000h** | CX = **0004h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
 1 org 100h
 2 jmp basla
 3 dizi db 100, -30, 90, -7
 4 toplam dw 0
 5 basla:
 6 lea si, dizi
 7 mov cx, 4
 8 xor bx, bx      ; BX = 16 bit toplam
 9 dongu:
10 mov al, [si]
11 cbw             ; AL -> AX (işaretle)
▶ add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx  ; 153 = 0099h
16 ret
```

BX = BX + AX = **0064h** (100).

Adım 3 / 10

DEĞİŞKENLER

AL = **64h** | AX = **0064h** | BX = **0064h** | CX = **0004h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
1  org 100h
2  jmp basla
3  dizi db 100, -30, 90, -7
4  toplam dw 0
5  basla:
6  lea si, dizi
7  mov cx, 4
8  xor bx, bx      ; BX = 16 bit toplam
9  dongu:
▶  mov al, [si]
▶  cbw             ; AL -> AX (işaretle)
12 add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx  ; 153 = 0099h
16 ret
```

AL = E2h (−30). MSB = 1 → `CBW` AH = FFh, AX = **FFE2h**.

Adım 4 / 10

DEĞİŞKENLER

AL = **E2h** | AX = **FFE2h** | BX = **0064h** | CX = **0003h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
 1 org 100h
 2 jmp basla
 3 dizi db 100, -30, 90, -7
 4 toplam dw 0
 5 basla:
 6 lea si, dizi
 7 mov cx, 4
 8 xor bx, bx      ; BX = 16 bit toplam
 9 dongu:
10 mov al, [si]
11 cbw             ; AL -> AX (işaretle)
▶ add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx  ; 153 = 0099h
16 ret
```

BX = BX + AX = **0046h** (70).

Adım 5 / 10

DEĞİŞKENLER

AL = **E2h** | AX = **FFE2h** | BX = **0046h** | CX = **0003h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
1  org 100h
2  jmp basla
3  dizi db 100, -30, 90, -7
4  toplam dw 0
5  basla:
6  lea si, dizi
7  mov cx, 4
8  xor bx, bx      ; BX = 16 bit toplam
9  dongu:
▶  mov al, [si]
▶  cbw             ; AL -> AX (işaretle)
12 add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx  ; 153 = 0099h
16 ret
```

AL = 5Ah (+90). MSB = 0 → `CBW` AH = 00h, AX = 005Ah.

Adım 6 / 10

DEĞİŞKENLER

AL = **5Ah** | AX = **005Ah** | BX = **0046h** | CX = **0002h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
1  org 100h
2  jmp basla
3  dizi db 100, -30, 90, -7
4  toplam dw 0
5  basla:
6  lea si, dizi
7  mov cx, 4
8  xor bx, bx      ; BX = 16 bit toplam
9  dongu:
10 mov al, [si]
11 cbw             ; AL -> AX (işaretle)
▶  add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx  ; 153 = 0099h
16 ret
```

BX = **00A0h** (160). Bayt toplasaydık 70 + 90 = 160 > 127 → AL = A0h (−96), OF = 1 olurdu.

Adım 7 / 10

DEĞİŞKENLER

AL = **5Ah** | AX = **005Ah** | BX = **00A0h** | CX = **0002h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
1  org 100h
2  jmp basla
3  dizi db 100, -30, 90, -7
4  toplam dw 0
5  basla:
6  lea si, dizi
7  mov cx, 4
8  xor bx, bx      ; BX = 16 bit toplam
9  dongu:
▶  mov al, [si]
▶  cbw             ; AL -> AX (işaretle)
12 add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx  ; 153 = 0099h
16 ret
```

AL = F9h (−7). MSB = 1 → `CBW` AH = FFh, AX = **FFF9h**.

Adım 8 / 10

DEĞİŞKENLER

AL = **F9h** | AX = **FFF9h** | BX = **00A0h** | CX = **0001h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
 1 org 100h
 2 jmp basla
 3 dizi db 100, -30, 90, -7
 4 toplam dw 0
 5 basla:
 6 lea si, dizi
 7 mov cx, 4
 8 xor bx, bx      ; BX = 16 bit toplam
 9 dongu:
10 mov al, [si]
11 cbw             ; AL -> AX (işaretle)
▶ add bx, ax
13 inc si
14 loop dongu
15 mov toplam, bx  ; 153 = 0099h
16 ret
```

BX = BX + AX = **0099h** (153).

Adım 9 / 10

DEĞİŞKENLER

AL = **F9h** | AX = **FFF9h** | BX = **0099h** | CX = **0001h** | toplam = **0000h**

CBW ile İşaretli Bayt Toplamı

```
1  org 100h
2  jmp basla
3  dizi db 100, -30, 90, -7
4  toplam dw 0
5  basla:
6  lea si, dizi
7  mov cx, 4
8  xor bx, bx      ; BX = 16 bit toplam
9  dongu:
10 mov al, [si]
11 cbw             ; AL -> AX (işaretle)
12 add bx, ax
13 inc si
14 loop dongu
▶ 15 mov toplam, bx  ; 153 = 0099h
16 ret
```

Döngü bitti, toplam = **0099h** = 153 ✓ belleğe yazılır. CBW yerine `MOV AH, 0` kullanılsaydı −30 = E2h, 226 olarak eklenirdi.

Adım 10 / 10

DEĞİŞKENLER

`AL = F9h` `AX = FFF9h` `BX = 0099h` `CX = 0000h` `toplam = 0099h`

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
▶  lea si, dizi
▶  mov cx, 3
8  dongu:
9  mov ax, [si]
10 cwd                   ; AX -> DX:AX
11 add [toplam], ax      ; düşük sözcük
12 adc [toplam+2], dx    ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

SI = dizi, CX = 3 sözcük. Toplam (32 bit) bellekte 0000:0000.

Adım 1 / 10

DEĞİŞKENLER

AX = **0000h** | DX = **0000h** | CF = **0**

toplam = **0000h** | toplam+2 = **0000h**

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
▶  mov ax, [si]
▶  cwd                   ; AX -> DX:AX
11 add [toplam], ax      ; düşük sözcük
12 adc [toplam+2], dx    ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

AX = 7530h (30000). Pozitif → `CWD` DX = 0000h.

Adım 2 / 10

DEĞİŞKENLER

`AX = 7530h` `DX = 0000h` `CF = 0`

`toplam = 0000h` `toplam+2 = 0000h`

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
9  mov ax, [si]
10 cwd                   ; AX -> DX:AX
▶ add [toplam], ax       ; düşük sözcük
12 adc [toplam+2], dx    ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

Düşük sözcük: 0000h + 7530h = 7530h, CF = 0.

Adım 3 / 10

DEĞİŞKENLER

AX = **7530h** | DX = **0000h** | CF = **0**

toplam = **7530h** | toplam+2 = **0000h**

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
9  mov ax, [si]
10 cwd                   ; AX -> DX:AX
11 add [toplam], ax      ; düşük sözcük
▶ 12 adc [toplam+2], dx   ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

Yüksek sözcük: 0000h + 0000h + CF(0) = 0000h. Toplam 00007530h.

Adım 4 / 10

DEĞİŞKENLER

AX = **7530h** | DX = **0000h** | CF = **0**

toplam = **7530h** | toplam+2 = **0000h**

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
▶  mov ax, [si]
▶  cwd                   ; AX -> DX:AX
11 add [toplam], ax      ; düşük sözcük
12 adc [toplam+2], dx    ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

AX = 61A8h (25000), DX = 0000h.

Adım 5 / 10

DEĞİŞKENLER

AX = **61A8h** | DX = **0000h** | CF = **0**

toplam = **7530h** | toplam+2 = **0000h**

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
9  mov ax, [si]
10 cwd                   ; AX -> DX:AX
▶ add [toplam], ax       ; düşük sözcük
12 adc [toplam+2], dx    ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

7530h + 61A8h = **D6D8h** (55000). 16 bit işaretli yorumda negatif (OF = 1) ama toplam 32 bit tutulduğu için sorun yok.

Adım 6 / 10

DEĞİŞKENLER

AX = **61A8h** | DX = **0000h** | CF = **0**

toplam = **D6D8h** | toplam+2 = **0000h**

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
9  mov ax, [si]
10 cwd                   ; AX -> DX:AX
11 add [toplam], ax      ; düşük sözcük
▶ 12 adc [toplam+2], dx   ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

Yüksek sözcük 0000h kalır: toplam 0000D6D8h = 55000.

Adım 7 / 10

DEĞİŞKENLER

`AX = 61A8h` `DX = 0000h` `CF = 0` `toplam = D6D8h` `toplam+2 = 0000h`

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
▶  mov ax, [si]
▶  cwd                   ; AX -> DX:AX
11 add [toplam], ax      ; düşük sözcük
12 adc [toplam+2], dx    ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

AX = F060h (−4000). MSB = 1 → `CWD` **DX = FFFFh**: DX:AX = FFFFF060h (32 bitte −4000).

Adım 8 / 10

DEĞİŞKENLER

`AX = F060h` `DX = FFFFh` `CF = 0`

`toplam = D6D8h` `toplam+2 = 0000h`

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
9  mov ax, [si]
10 cwd                   ; AX -> DX:AX
▶ 11 add [toplam], ax     ; düşük sözcük
12 adc [toplam+2], dx    ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

D6D8h + F060h = 1C738h → düşük sözcük **C738h**, **CF = 1**.

Adım 9 / 10

DEĞİŞKENLER

AX = **F060h** | DX = **FFFFh** | CF = **1**

toplam = **C738h** | toplam+2 = **0000h**

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

CWD ve ADC ile 32 Bitlik Toplam

```
1  org 100h
2  jmp basla
3  dizi dw 30000, 25000, -4000
4  toplam dw 0, 0        ; 32 bit toplam
5  basla:
6  lea si, dizi
7  mov cx, 3
8  dongu:
9  mov ax, [si]
10 cwd                   ; AX -> DX:AX
11 add [toplam], ax      ; düşük sözcük
▶ 12 adc [toplam+2], dx   ; yüksek + CF
13 add si, 2
14 loop dongu
15 ret
```

FFFFh + 0000h + CF(1) = 10000h → yüksek sözcük **0000h**. Toplam 0000C738h = **51000** ✓

Adım 10 / 10

DEĞİŞKENLER

AX = **F060h** | DX = **FFFFh** | CF = **1**

toplam = **C738h** | toplam+2 = **0000h**

İşaretli sözcük dizisi: her eleman `CWD` ile 32 bite genişletilir, düşük sözcük `ADD`, yüksek sözcük `ADC` ile toplanır.

GRUP 2: ÇARPMA VE BÖLME

Demo: Çarpma / Bölme Hesaplayıcı

Komut, boyut ve değerler seçilince sonucun hangi yazmaçlara yazıldığı gösterilir. Bölmede bölünen, bayt için AX; sözcük için DX:AX'tir.

komut MUL boyut bayt AL 200 işlenen 3

200 × 3 = **600**

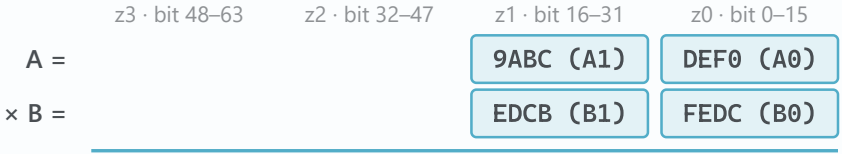
AX = 0258h AH = 02h AL = 58h **CF = OF = 1**

Sonuç üst yarıya taştı → CF = OF = 1.

GRUP 2: ÇARPMA VE BÖLME · ADIM ADIM

32 × 32 Bit Çarpma: Kısmi Çarpımlar

8086 en çok 16 × 16 bit çarpar. 32 bitlik sayılar sözcüklere bölünür: A × B = A1·B1·2^{32} + (A1·B0 + A0·B1)·2^{16} + A0·B0. Dört `MUL`, sonuçlar sözcük konumuna göre `ADD` / `ADC` ile toplanır; sonuç 64 bittir.



A = 9ABCDEF0h ve B = EDCBFEDCh, 16 bitlik yarılara ayrılır: A = A1:A0, B = B1:B0.

32 × 32 Bit Çarpma: Kısmi Çarpımlar

8086 en çok 16 × 16 bit çarpar. 32 bitlik sayılar sözcüklere bölünür: $A \times B = A1 \cdot B1 \cdot 2^{32} + (A1 \cdot B0 + A0 \cdot B1) \cdot 2^{16} + A0 \cdot B0$. Dört `MUL`, sonuçlar sözcük konumuna göre `ADD` / `ADC` ile toplanır; sonuç 64 bittir.



A0 × B0 kaydırılmaz: sonucun en düşük iki sözcüğüne (z1:z0) düşer.

32 × 32 Bit Çarpma: Kısmi Çarpımlar

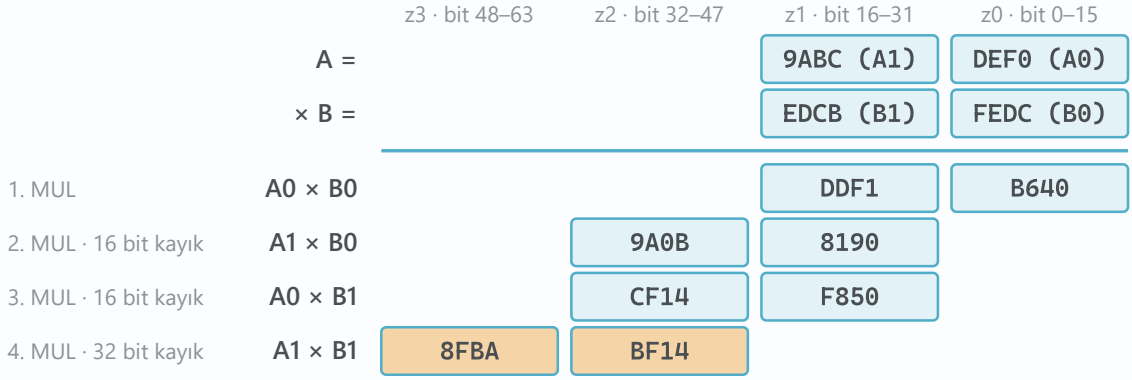
8086 en çok 16 × 16 bit çarpar. 32 bitlik sayılar sözcüklere bölünür: $A \times B = A1{\cdot}B1{\cdot}2^{32} + (A1{\cdot}B0 + A0{\cdot}B1){\cdot}2^{16} + A0{\cdot}B0$. Dört `MUL`, sonuçlar sözcük konumuna göre `ADD` / `ADC` ile toplanır; sonuç 64 bittir.

		z3 · bit 48–63	z2 · bit 32–47	z1 · bit 16–31	z0 · bit 0–15
	A =			9ABC (A1)	DEF0 (A0)
	× B =			EDCB (B1)	FEDC (B0)
1. MUL	A0 × B0			DDF1	B640
2. MUL · 16 bit kayık	A1 × B0		9A0B	8190	
3. MUL · 16 bit kayık	A0 × B1		CF14	F850	

A1 × B0 ve A0 × B1 ikisi de 2^{16} ile çarpılmıştır: z2:z1 sütunlarına yazılır.

32 × 32 Bit Çarpma: Kısmi Çarpımlar

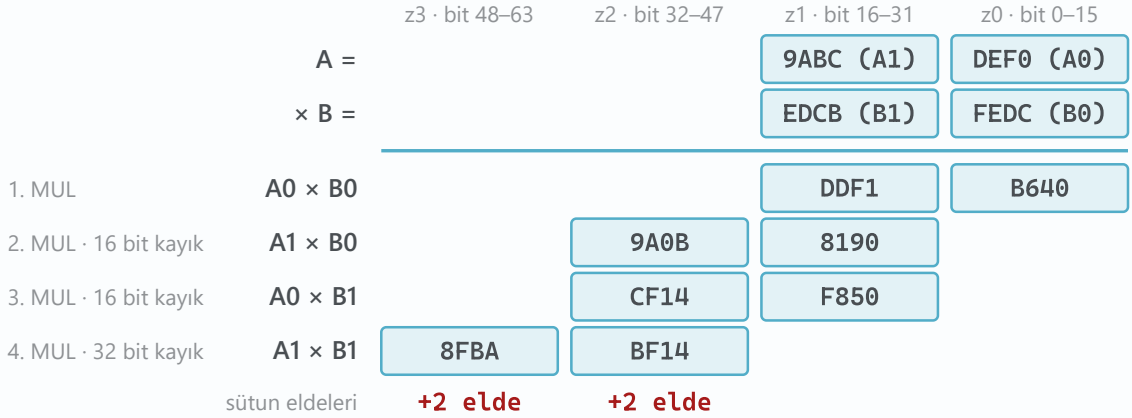
8086 en çok 16 × 16 bit çarpar. 32 bitlik sayılar sözcüklere bölünür: $A \times B = A1 \cdot B1 \cdot 2^{32} + (A1 \cdot B0 + A0 \cdot B1) \cdot 2^{16} + A0 \cdot B0$. Dört `MUL`, sonuçlar sözcük konumuna göre `ADD` / `ADC` ile toplanır; sonuç 64 bittir.



A1 × B1 2^{32} ile çarpılmıştır: z3:z2 sütunlarına yazılır.

32 × 32 Bit Çarpma: Kısmi Çarpımlar

8086 en çok 16 × 16 bit çarpar. 32 bitlik sayılar sözcüklere bölünür: A × B = A1·B1·2^{32} + (A1·B0 + A0·B1)·2^{16} + A0·B0. Dört `MUL`, sonuçlar sözcük konumuna göre `ADD` / `ADC` ile toplanır; sonuç 64 bittir.



Sütun toplamları: z1 = 257D1h → yaz 57D1h, **2 elde**; z2 = 22835h → yaz 2835h, **2 elde**. Her ADD en çok 1 elde verir: kodda ADC zinciri bu yüzden var.

32 × 32 Bit Çarpma: Kısmi Çarpımlar

8086 en çok 16 × 16 bit çarpar. 32 bitlik sayılar sözcüklere bölünür: $A \times B = A1 \cdot B1 \cdot 2^{32} + (A1 \cdot B0 + A0 \cdot B1) \cdot 2^{16} + A0 \cdot B0$. Dört `MUL`, sonuçlar sözcük konumuna göre `ADD` / `ADC` ile toplanır; sonuç 64 bittir.

		z3 · bit 48–63	z2 · bit 32–47	z1 · bit 16–31	z0 · bit 0–15
	A =			9ABC (A1)	DEF0 (A0)
	× B =			EDCB (B1)	FEDC (B0)
1. MUL	A0 × B0			DDF1	B640
2. MUL · 16 bit kayık	A1 × B0		9A0B	8190	
3. MUL · 16 bit kayık	A0 × B1		CF14	F850	
4. MUL · 32 bit kayık	A1 × B1	8FBA	BF14		
	sütun eldeleri	+2 elde	+2 elde		
	A × B =	8FBC	2835	57D1	B640

Sonuç: **8FBC 2835 57D1 B640h**. Kontrol (onluk): 2596069104 × 3989569244 = 10357197452617037376 ✓

32 × 32 Bit Çarpma: Kod İzleme

```
1  org 100h
▶ mov ax, [x]        ; A0
▶ mul [y]            ; A0*B0
4  mov [z], ax
5  mov [z+2], dx
6  mov ax, [x+2]     ; A1
7  mul [y]           ; A1*B0
8  add [z+2], ax
9  adc [z+4], dx
10 mov ax, [x]       ; A0
11 mul [y+2]         ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]     ; A1
16 mul [y+2]         ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

AX = A0 = DEF0h, `MUL [y]`: A0 × B0 = DEF0h × FEDCh = **DDF1:B640**h (DX:AX). DX ≠ 0 olduğundan MUL CF = 1 yapar.

Adım 1 / 12

DEĞİŞKENLER

AX = **B640h** | DX = **DDF1h** | CF = **1** | z3 = **0000h** | z2 = **0000h** | z1 = **0000h** | z0 = **0000h**

32 × 32 Bit Çarpma: Kod İzleme

```
1  org 100h
2  mov ax, [x]        ; A0
3  mul [y]            ; A0*B0
▶  mov [z], ax
▶  mov [z+2], dx
6  mov ax, [x+2]      ; A1
7  mul [y]            ; A1*B0
8  add [z+2], ax
9  adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

İlk kısmi çarpım z0 (bit 0–15) ve z1 (bit 16–31) sözcüklerine yazılır.

Adım 2 / 12

DEĞİŞKENLER

AX = **B640h** | DX = **DDF1h** | CF = **1** | z3 = **0000h**

z2 = **0000h** | z1 = **DDF1h** | z0 = **B640h**

32 × 32 Bit Çarpma: Kod İzleme

```
1  org 100h
2  mov ax, [x]        ; A0
3  mul [y]            ; A0*B0
4  mov [z], ax
5  mov [z+2], dx
▶  mov ax, [x+2]      ; A1
▶  mul [y]            ; A1*B0
8  add [z+2], ax
9  adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

A1 × B0 = 9ABCh × FEDCh = **9A0B:8190**h. 16 bit sola kaymış: z1 ve z2'ye eklenecek.

Adım 3 / 12

DEĞİŞKENLER

AX = **8190h** | DX = **9A0Bh** | CF = **1** | z3 = **0000h** | z2 = **0000h** | z1 = **DDF1h** | z0 = **B640h**

32 × 32 Bit Çarpma: Kod İzleme

```
1  org 100h
2  mov ax, [x]        ; A0
3  mul [y]            ; A0*B0
4  mov [z], ax
5  mov [z+2], dx
6  mov ax, [x+2]      ; A1
7  mul [y]            ; A1*B0
▶  add [z+2], ax
9  adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

z1 + AX = DDF1h + 8190h → z1 = **5F81h**, CF = 1.

Adım 4 / 12

DEĞİŞKENLER

AX = **8190h**	DX = **9A0Bh**	CF = **1**	z3 = **0000h**
z2 = **0000h**	z1 = **5F81h**	z0 = **B640h**	

32 × 32 Bit Çarpma: Kod İzleme

```
1  org 100h
2  mov ax, [x]        ; A0
3  mul [y]            ; A0*B0
4  mov [z], ax
5  mov [z+2], dx
6  mov ax, [x+2]      ; A1
7  mul [y]            ; A1*B0
8  add [z+2], ax
▶  adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

z2 = 0 + DX + CF = **9A0Ch**. z2 önceden 0 olduğu için burada yeni elde oluşmaz.

Adım 5 / 12

DEĞİŞKENLER

AX = **8190h** | DX = **9A0Bh** | CF = **0** | z3 = **0000h**

z2 = **9A0Ch** | z1 = **5F81h** | z0 = **B640h**

32 × 32 Bit Çarpma: Kod İzleme

```
org 100h
mov ax, [x]        ; A0
mul [y]            ; A0*B0
mov [z], ax
mov [z+2], dx
mov ax, [x+2]      ; A1
mul [y]            ; A1*B0
add [z+2], ax
adc [z+4], dx
▶ mov ax, [x]      ; A0
▶ mul [y+2]        ; A0*B1
add [z+2], ax
adc [z+4], dx
adc [z+6], 0
mov ax, [x+2]      ; A1
mul [y+2]          ; A1*B1
add [z+4], ax
adc [z+6], dx
ret
x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
z dw 4 dup(0)       ; 64 bit sonuç
```

A0 × B1 = DEF0h × EDCBh = **CF14:F850**h. Bu da z1–z2 konumuna eklenir.

Adım 6 / 12

DEĞİŞKENLER

AX = **F850h** DX = **CF14h** CF = **1** z3 = **0000h**

z2 = **9A0Ch** z1 = **5F81h** z0 = **B640h**

32 × 32 Bit Çarpma: Kod İzleme

```
 1 org 100h
 2 mov ax, [x]        ; A0
 3 mul [y]            ; A0*B0
 4 mov [z], ax
 5 mov [z+2], dx
 6 mov ax, [x+2]      ; A1
 7 mul [y]            ; A1*B0
 8 add [z+2], ax
 9 adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
▶ add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

z1 = 5F81h + F850h → **57D1h**, CF = 1.

Adım 7 / 12

DEĞİŞKENLER

AX = **F850h**	DX = **CF14h**	CF = **1**	z3 = **0000h**
z2 = **9A0Ch**	z1 = **57D1h**	z0 = **B640h**	

32 × 32 Bit Çarpma: Kod İzleme

```
1  org 100h
2  mov ax, [x]        ; A0
3  mul [y]            ; A0*B0
4  mov [z], ax
5  mov [z+2], dx
6  mov ax, [x+2]      ; A1
7  mul [y]            ; A1*B0
8  add [z+2], ax
9  adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
▶ adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

z2 = 9A0Ch + CF14h + CF → **6921h**, CF = 1: elde z3'e gitmeli!

Adım 8 / 12

DEĞİŞKENLER

AX = **F850h** | DX = **CF14h** | CF = **1** | z3 = **0000h**

z2 = **6921h** | z1 = **57D1h** | z0 = **B640h**

32 × 32 Bit Çarpma: Kod İzleme

```
1  org 100h
2  mov ax, [x]        ; A0
3  mul [y]            ; A0*B0
4  mov [z], ax
5  mov [z+2], dx
6  mov ax, [x+2]      ; A1
7  mul [y]            ; A1*B0
8  add [z+2], ax
9  adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
▶ 14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

`ADC [z+6], 0` : yalnızca eldeyi taşır → z3 = **0001h**.

Adım 9 / 12

DEĞİŞKENLER

AX = **F850h** | DX = **CF14h** | CF = **0** | z3 = **0001h**

z2 = **6921h** | z1 = **57D1h** | z0 = **B640h**

32 × 32 Bit Çarpma: Kod İzleme

```
 1 org 100h
 2 mov ax, [x]        ; A0
 3 mul [y]            ; A0*B0
 4 mov [z], ax
 5 mov [z+2], dx
 6 mov ax, [x+2]      ; A1
 7 mul [y]            ; A1*B0
 8 add [z+2], ax
 9 adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
▶ mov ax, [x+2]      ; A1
▶ mul [y+2]          ; A1*B1
17 add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

A1 × B1 = 9ABCh × EDCBh = **8FBA:BF14**h, 32 bit sola kaymış: z2–z3.

Adım 10 / 12

DEĞİŞKENLER

AX = **BF14h** DX = **8FBAh** CF = **1** z3 = **0001h**

z2 = **6921h** z1 = **57D1h** z0 = **B640h**

32 × 32 Bit Çarpma: Kod İzleme

```
 1 org 100h
 2 mov ax, [x]        ; A0
 3 mul [y]            ; A0*B0
 4 mov [z], ax
 5 mov [z+2], dx
 6 mov ax, [x+2]      ; A1
 7 mul [y]            ; A1*B0
 8 add [z+2], ax
 9 adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
▶ add [z+4], ax
18 adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

z2 = 6921h + BF14h → **2835h**, CF = 1.

Adım 11 / 12

DEĞİŞKENLER

AX = **BF14h** | DX = **8FBAh** | CF = **1** | z3 = **0001h**

z2 = **2835h** | z1 = **57D1h** | z0 = **B640h**

32 × 32 Bit Çarpma: Kod İzleme

```
1  org 100h
2  mov ax, [x]        ; A0
3  mul [y]            ; A0*B0
4  mov [z], ax
5  mov [z+2], dx
6  mov ax, [x+2]      ; A1
7  mul [y]            ; A1*B0
8  add [z+2], ax
9  adc [z+4], dx
10 mov ax, [x]        ; A0
11 mul [y+2]          ; A0*B1
12 add [z+2], ax
13 adc [z+4], dx
14 adc [z+6], 0
15 mov ax, [x+2]      ; A1
16 mul [y+2]          ; A1*B1
17 add [z+4], ax
▶ adc [z+6], dx
19 ret
20 x dw 0DEF0h, 9ABCh  ; A = 9ABCDEF0h
21 y dw 0FEDCh, 0EDCBh ; B = EDCBFEDCh
22 z dw 4 dup(0)       ; 64 bit sonuç
```

z3 = 0001h + 8FBAh + CF = **8FBCh**. Sonuç z3:z2:z1:z0 = **8FBC 2835 57D1 B640h** ✓

Adım 12 / 12

DEĞİŞKENLER

AX = **BF14h** | DX = **8FBAh** | CF = **0** | z3 = **8FBCh**

z2 = **2835h** | z1 = **57D1h** | z0 = **B640h**

GRUP 3: TEK İŞLENENLİ

INC, DEC, NOT, NEG

- İşlenenler: **REG** (AX, BX, … SP) veya **memory** (`[BX]` , `[BX+SI+7]` , değişken). Sabit olamaz.
- `INC` ve `DEC` yalnızca **ZF, SF, OF, PF, AF**'yi etkiler; **CF'ye dokunmaz**.
- `NOT` hiçbir bayrağı etkilemez! İşlenenin her bitini ters çevirir (birler tümleyeni).
- `NEG` CF, ZF, SF, OF, PF, AF'yi etkiler. İşleneni negatif yapar (**ikiye tümleyen**): her biti tersler, ardından 1 ekler. Ör. 5 → −5, −2 → 2.

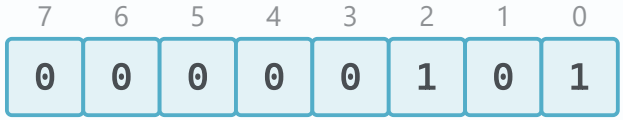
Komut	CF	ZF	SF	OF	PF	AF
`INC` , `DEC`	—	✓	✓	✓	✓	✓
`NOT`	—	—	—	—	—	—
`NEG`	✓ (işlenen ≠ 0 ise 1)	✓	✓	✓	✓	✓

—: değişmez. INC/DEC'in CF'yi korması, döngü içinde ADC ile çok baytlı toplama yaparken sayaç artırmayı mümkün kılar.

GRUP 3: TEK İŞLENENLİ

NEG: İkiye Tümleyen Adım Adım

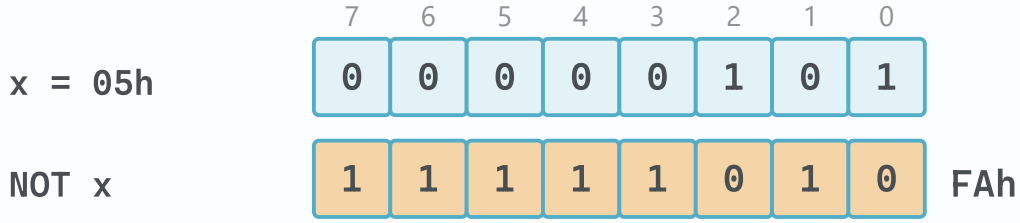
`x = 05h`



x = 5 = 0000 0101b.

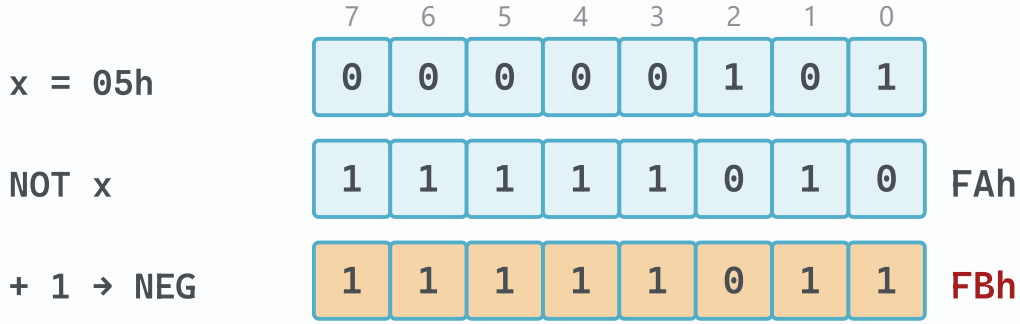
Adım 1 / 6

NEG: İkiye Tümleyen Adım Adım



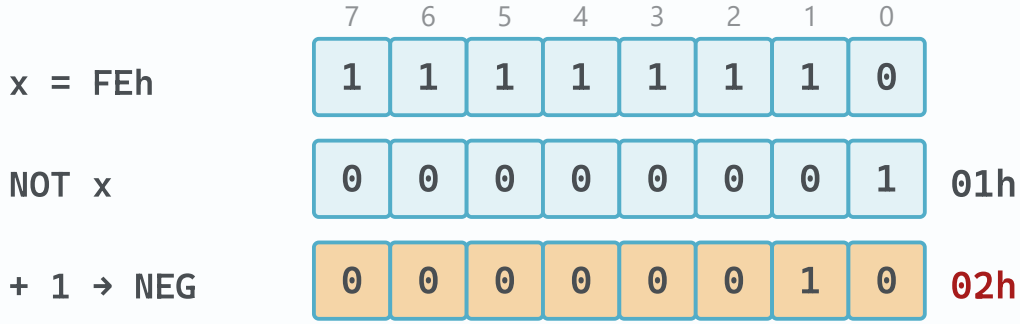
NOT: her bit terslenir → 1111 1010b = FAh (−6). Birler tümleyeni.

NEG: İkiye Tümleyen Adım Adım



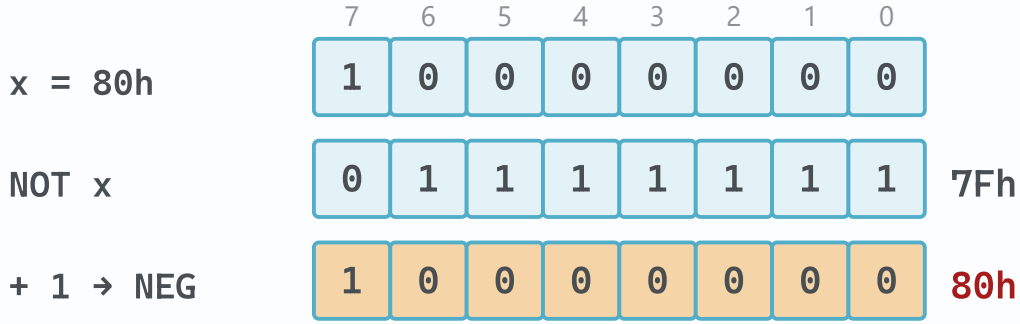
+1: FBh = 1111 1011b = **−5**. `NEG` bu iki adımı tek komutta yapar; CF = 1 (x ≠ 0).

NEG: İkiye Tümleyen Adım Adım



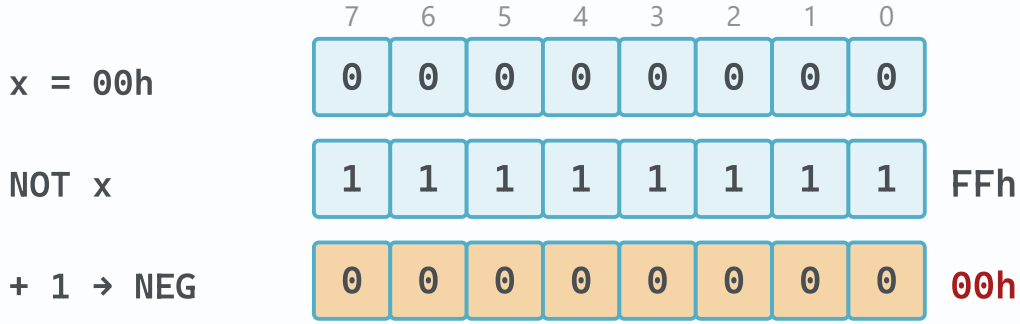
x = FEh (−2): NOT → 01h, +1 → **02h**. Negatifin negatifi pozitiftir.

NEG: İkiye Tümleyen Adım Adım



Özel durum: x = 80h (−128). NOT → 7Fh, +1 → **80h**: +128 bayta sığmaz → **OF = 1**.

NEG: İkiye Tümleyen Adım Adım



x = 0: sonuç 0, **CF = 0** (NEG'de CF = 1 olmayan tek durum), ZF = 1.

Kod İzleme: INC, DEC, NOT, NEG

```
▶ mov al, 0FFh
2 inc al          ; FFh + 1
3 dec al          ; 00h - 1
4 mov bl, 7Fh     ; +127
5 inc bl          ; +128 ?
6 not bl
7 neg bl
```

AL = FFh (255 / −1).

Adım 1 / 7

DEĞİŞKENLER

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

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

Kod İzleme: INC, DEC, NOT, NEG

```
1 mov al, 0FFh
▶ inc al          ; FFh + 1
3 dec al          ; 00h - 1
4 mov bl, 7Fh     ; +127
5 inc bl          ; +128 ?
6 not bl
7 neg bl
```

FFh + 1 = **00h**, ZF = 1. ADD olsaydı CF = 1 olurdu; **INC CF'yi değiştirmez** → CF = 0 kaldı.

Adım 2 / 7

DEĞİŞKENLER

AL = **00h** | BL = **00h** | CF = **0** | ZF = **1**

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

Kod İzleme: INC, DEC, NOT, NEG

```
1 mov al, 0FFh
2 inc al          ; FFh + 1
▶ dec al          ; 00h - 1
4 mov bl, 7Fh     ; +127
5 inc bl          ; +128 ?
6 not bl
7 neg bl
```

00h − 1 = **FFh**, SF = 1. Ödünç olsa da CF yine değişmedi.

Adım 3 / 7

DEĞİŞKENLER

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

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

Kod İzleme: INC, DEC, NOT, NEG

```
1 mov al, 0FFh
2 inc al          ; FFh + 1
3 dec al          ; 00h - 1
▶ mov bl, 7Fh     ; +127
5 inc bl          ; +128 ?
6 not bl
7 neg bl
```

BL = 7Fh = +127, en büyük pozitif bayt.

Adım 4 / 7

DEĞİŞKENLER

AL = **FFh** | BL = **7Fh** | CF = **0** | ZF = **0** | SF = **1** | OF = **0** | PF = **1** | AF = **1**

Kod İzleme: INC, DEC, NOT, NEG

```
1 mov al, 0FFh
2 inc al          ; FFh + 1
3 dec al          ; 00h - 1
4 mov bl, 7Fh     ; +127
▶ inc bl          ; +128 ?
6 not bl
7 neg bl
```

7Fh + 1 = **80h** = −128: işaretli taşma → **OF = 1**, SF = 1.

Adım 5 / 7

DEĞİŞKENLER

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

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

Kod İzleme: INC, DEC, NOT, NEG

```
1 mov al, 0FFh
2 inc al          ; FFh + 1
3 dec al          ; 00h - 1
4 mov bl, 7Fh     ; +127
5 inc bl          ; +128 ?
▶ not bl
7 neg bl
```

`NOT` : 80h → **7Fh**. Bayraklar **hiç değişmedi** (SF hâlâ 1 görünüyor).

Adım 6 / 7

DEĞİŞKENLER

AL = **FFh** | BL = **7Fh** | CF = **0** | ZF = **0**

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

Kod İzleme: INC, DEC, NOT, NEG

```
1 mov al, 0FFh
2 inc al          ; FFh + 1
3 dec al          ; 00h - 1
4 mov bl, 7Fh     ; +127
5 inc bl          ; +128 ?
6 not bl
▶ neg bl
```

`NEG` : −127 = **81h**. CF = 1 (işlenen ≠ 0), SF = 1, OF = 0.

Adım 7 / 7

DEĞİŞKENLER

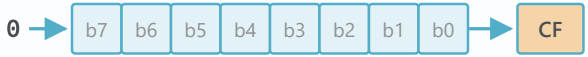
AL = **FFh** | BL = **81h** | CF = **1** | ZF = **0**

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

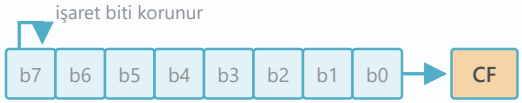
Kaydırma: SHL/SAL, SHR, SAR



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



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



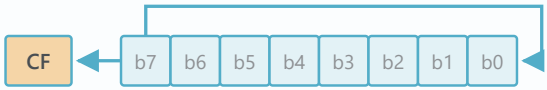
`SAR`: aritmetik sağa kaydır, işaret biti korunur. İşaretli ÷ **2** (−∞'a yuvarlar).

Kaydırma sayısı

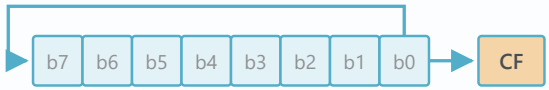
8086'da yalnızca `1` ya da `CL`: `shl ax, 1`, `shr al, cl`. `shl al, 4` gibi anlık sayı 80186 ve sonrasında geçerlidir. OF yalnızca sayı 1 iken tanımlıdır.

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

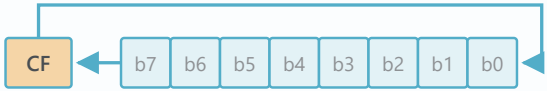
Döndürme: ROL, ROR, RCL, RCR



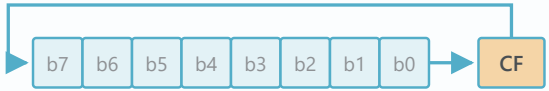
ROL: sola döndür; b7 hem b0'a hem CF'ye gider.



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 düğmesi seçilince sonuç burada görünür.

Her düğme komutu `op al, 1` olarak uygular. SHL art arda seçilince sayı her adımda iki katına çıkar; SAR işaret bitini korur.

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**: kâğıt üzerindeki çarpmada olduğu gibi bitler tek tek incelenir.

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 × 3 = AX×2 + AX
mov bx, ax      ; sakla
shl ax, 1       ; AX × 2
add ax, bx      ; ×2 + ×1

; işaretsiz ÷ 8 ve kalan
mov dx, ax
and dx, 0007h   ; kalan = alt 3 bit
mov cl, 3
shr ax, cl      ; bölüm

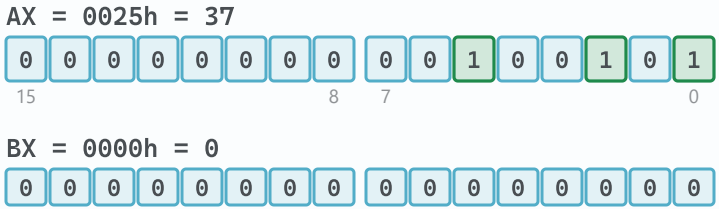
; işaretli ÷ 4
mov cl, 2
sar ax, cl      ; -7 → -2
```

- 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 bölünende SAR $-\infty$'a, IDIV sıfıra doğru yuvarlar: −7 için SAR 2 → **−2**, IDIV 4 → **−1**.

Örnek: AX = 25 → BX = 25, AX = 50 → 50 + 25 = **75**.

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 = 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
```

AX = 004Ah = 74 — 37 → 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

BX = 0000h = 0

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	0	0	0	0	0	0	0

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 saklanır.

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.

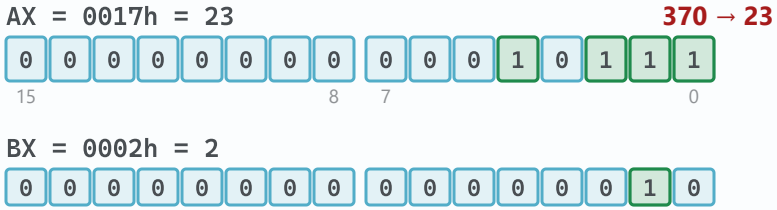
Adım 9 / 11

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**

ÖRNEK: DİZİ TOPLAMA

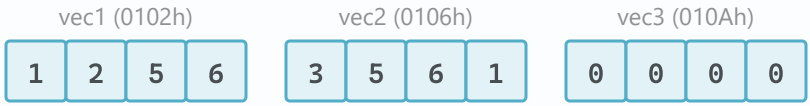
Dizi Elemanları Toplama

```
org 100h
jmp start
vec1 db 1, 2, 5, 6
vec2 db 3, 5, 6, 1
vec3 db ?, ?, ?, ?
start:
lea si, vec1
lea bx, vec2
lea di, vec3
mov cx, 4
sum:
mov al, [si]
add al, [bx]
mov [di], al
inc si
inc bx
inc di
loop sum
ret
```

- `vec3[i] = vec1[i] + vec2[i]`, i = 0…3.
- `JMP start` veri tanımlarının üzerinden atlar; aksi hâlde işlemci veri baytlarını komut sanardı.
- SI, BX, DI üç dizinin **işaretçileri**dir; `LEA` adresi (bağıl konumu) yükler.
- `[SI]`: SI'nin gösterdiği bellek baytı (dolaylı adresleme).
- `LOOP sum`: CX'i 1 azaltır, CX ≠ 0 ise `sum`'a döner → 4 tur.
- Beklenen sonuç: vec3 = 4, 7, 11, 7.

Kod İzleme: Dizi Toplama

```
1  org 100h
▶ 2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



`JMP start` : veriler (0102h–010Dh) atlanır, 010Eh'ye gidilir.

Adım 1 / 17

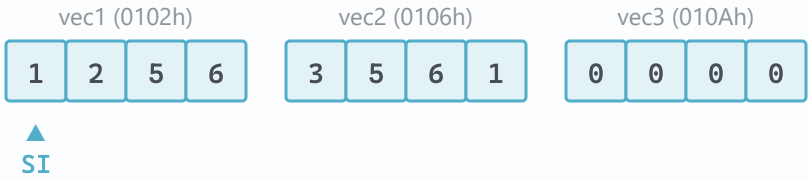
DEĞİŞKENLER

`SI = 0000h` `BX = 0000h` `DI = 0000h` `CX = 0000h`

`AL = 00h` `CF = 0` `ZF = 0`

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
▶  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



SI = vec1'in adresi = **0102h**.

Adım 2 / 17

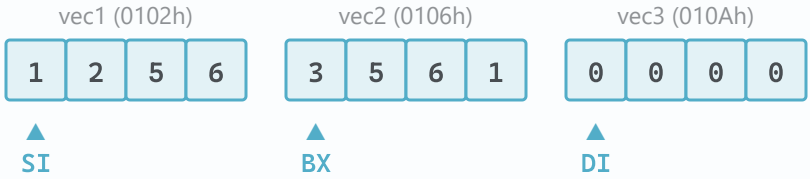
DEĞİŞKENLER

SI = **0102h** | BX = **0000h** | DI = **0000h** | CX = **0000h**

AL = **00h** | CF = **0** | ZF = **0**

Kod İzleme: Dizi Toplama

```
 1 org 100h
 2 jmp start
 3 vec1 db 1, 2, 5, 6
 4 vec2 db 3, 5, 6, 1
 5 vec3 db ?, ?, ?, ?
 6 start:
 7 lea si, vec1
▶ lea bx, vec2
▶ lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



BX = 0106h (vec2), DI = 010Ah (vec3).

Adım 3 / 17

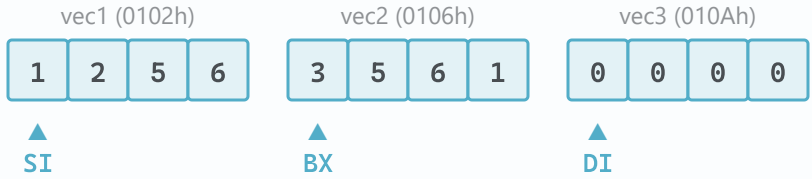
DEĞİŞKENLER

SI = **0102h** | BX = **0106h** | DI = **010Ah** | CX = **0000h**

AL = **00h** | CF = **0** | ZF = **0**

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
▶ mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



CX = 4 tur.

Adım 4 / 17

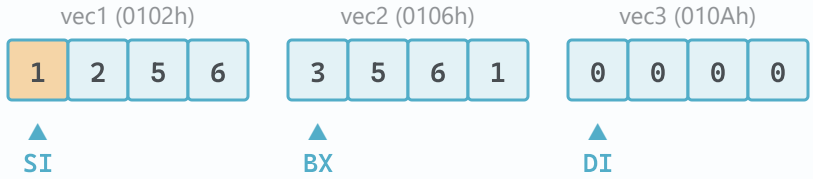
DEĞİŞKENLER

SI = 0102h | BX = 0106h | DI = 010Ah | CX = 0004h

AL = 00h | CF = 0 | ZF = 0

Kod İzleme: Dizi Toplama

```
 1 org 100h
 2 jmp start
 3 vec1 db 1, 2, 5, 6
 4 vec2 db 3, 5, 6, 1
 5 vec3 db ?, ?, ?, ?
 6 start:
 7 lea si, vec1
 8 lea bx, vec2
 9 lea di, vec3
10 mov cx, 4
11 sum:
▶ mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



Tur 1: AL ← [SI] = [0102h] = **1**.

Adım 5 / 17

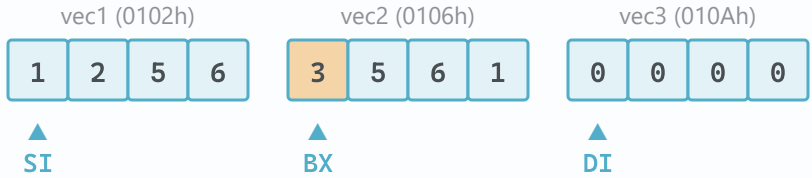
DEĞİŞKENLER

SI = **0102h** | BX = **0106h** | DI = **010Ah** | CX = **0004h**

AL = **01h** | CF = **0** | ZF = **0**

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
▶ add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



AL ← AL + [BX] = 1 + 3 = **4** (04h).

Adım 6 / 17

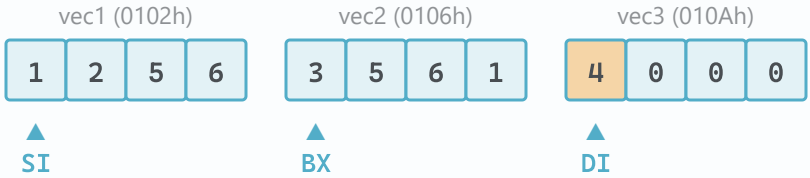
DEĞİŞKENLER

SI = **0102h** · BX = **0106h** · DI = **010Ah** · CX = **0004h**

AL = **04h** · CF = **0** · ZF = **0**

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
▶ 14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



[DI] ← AL: vec3[0] = **4**.

Adım 7 / 17

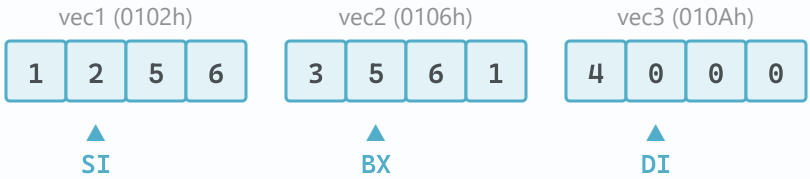
DEĞİŞKENLER

SI = **0102h** BX = **0106h** DI = **010Ah** CX = **0004h**

AL = **04h** CF = **0** ZF = **0**

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
   inc si
   inc bx
   inc di
18 loop sum
19 ret
```



Üç işaretçi de bir bayt ilerler (diziler bayt dizisidir; sözcük dizisi olsaydı 2 eklenirdi).

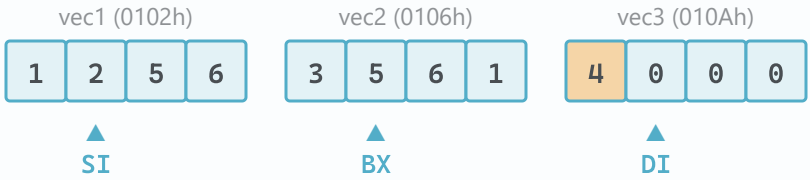
Adım 8 / 17

DEĞİŞKENLER

SI = 0103h | BX = 0107h | DI = 010Bh | CX = 0004h | AL = 04h | CF = 0 | ZF = 0

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
▶ 18 loop sum
19 ret
```



`LOOP` : CX = 3 ≠ 0 → `sum` 'a dön.

Adım 9 / 17

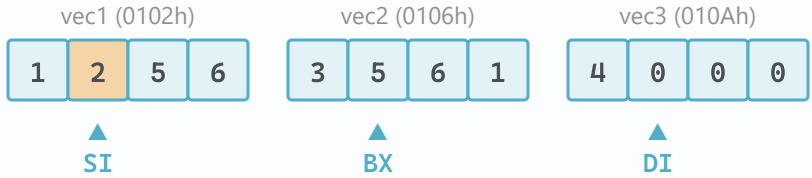
DEĞİŞKENLER

SI = **0103h** | BX = **0107h** | DI = **010Bh** | CX = **0003h**

AL = **04h** | CF = **0** | ZF = **0**

Kod İzleme: Dizi Toplama

```
 1 org 100h
 2 jmp start
 3 vec1 db 1, 2, 5, 6
 4 vec2 db 3, 5, 6, 1
 5 vec3 db ?, ?, ?, ?
 6 start:
 7 lea si, vec1
 8 lea bx, vec2
 9 lea di, vec3
10 mov cx, 4
11 sum:
▶ mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



Tur 2: AL ← [SI] = [0103h] = **2**.

Adım 10 / 17

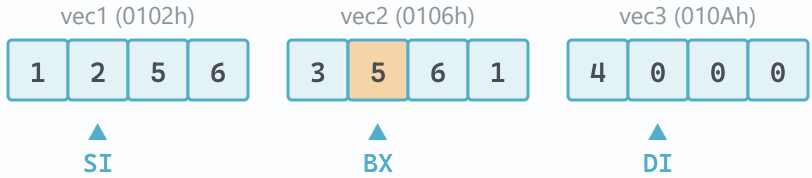
DEĞİŞKENLER

SI = **0103h** · BX = **0107h** · DI = **010Bh** · CX = **0003h**

AL = **02h** · CF = **0** · ZF = **0**

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
▶ add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



AL ← AL + [BX] = 2 + 5 = **7** (07h).

Adım 11 / 17

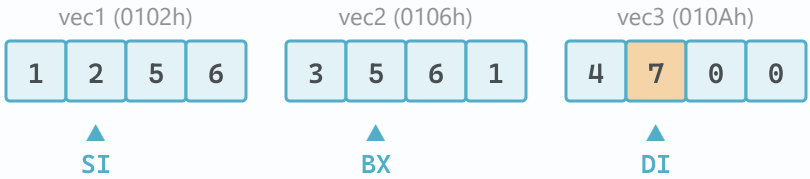
DEĞİŞKENLER

SI = **0103h** · BX = **0107h** · DI = **010Bh** · CX = **0003h**

AL = **07h** · CF = **0** · ZF = **0**

Kod İzleme: Dizi Toplama

```
 1 org 100h
 2 jmp start
 3 vec1 db 1, 2, 5, 6
 4 vec2 db 3, 5, 6, 1
 5 vec3 db ?, ?, ?, ?
 6 start:
 7 lea si, vec1
 8 lea bx, vec2
 9 lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
▶ mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
19 ret
```



[DI] ← AL: vec3[1] = **7**.

Adım 12 / 17

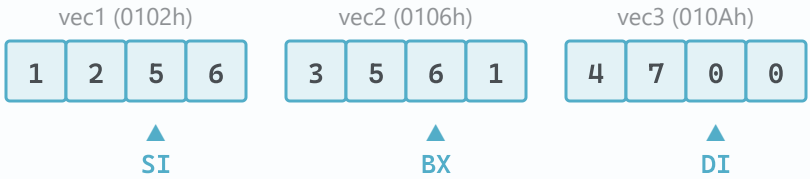
DEĞİŞKENLER

SI = **0103h** · BX = **0107h** · DI = **010Bh** · CX = **0003h**

AL = **07h** · CF = **0** · ZF = **0**

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
   inc si
   inc bx
   inc di
18 loop sum
19 ret
```



Üç işaretçi de bir bayt ilerler (diziler bayt dizisidir; sözcük dizisi olsaydı 2 eklenirdi).

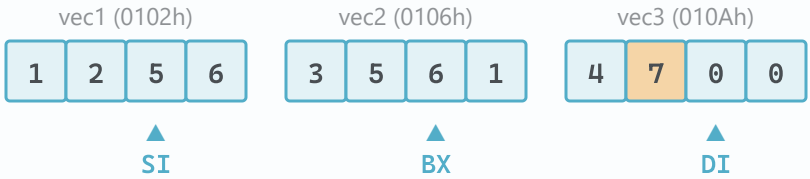
Adım 13 / 17

DEĞİŞKENLER

SI = 0104h · BX = 0108h · DI = 010Ch · CX = 0003h · AL = 07h · CF = 0 · ZF = 0

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
▶ loop sum
19 ret
```



`LOOP` : CX = 2 ≠ 0 → `sum` 'a dön.

Adım 14 / 17

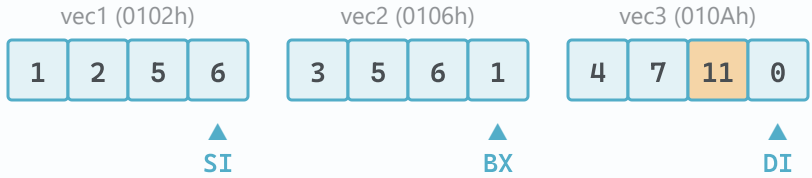
DEĞİŞKENLER

SI = **0104h** | BX = **0108h** | DI = **010Ch** | CX = **0002h**

AL = **07h** | CF = **0** | ZF = **0**

Kod İzleme: Dizi Toplama

```
 1 org 100h
 2 jmp start
 3 vec1 db 1, 2, 5, 6
 4 vec2 db 3, 5, 6, 1
 5 vec3 db ?, ?, ?, ?
 6 start:
 7 lea si, vec1
 8 lea bx, vec2
 9 lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
▶ loop sum
19 ret
```



Tur 3 (özet): 5 + 6 = **11** → vec3[2]. `LOOP` : CX = 1 ≠ 0 → `sum` 'a dön.

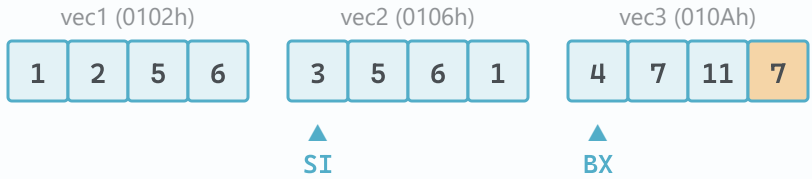
Adım 15 / 17

DEĞİŞKENLER

`SI = 0105h` `BX = 0109h` `DI = 010Dh` `CX = 0001h` `AL = 0Bh` `CF = 0` `ZF = 0`

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
▶ loop sum
19 ret
```



Tur 4 (özet): 6 + 1 = **7** → vec3[3]. `LOOP` : CX = 0 → döngü biter.

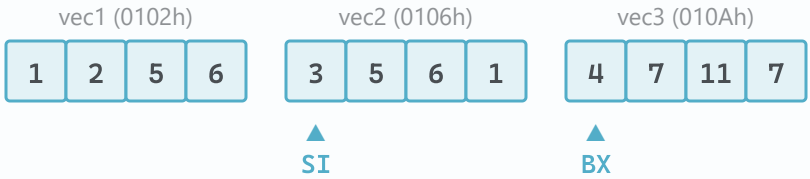
Adım 16 / 17

DEĞİŞKENLER

`SI = 0106h` `BX = 010Ah` `DI = 010Eh` `CX = 0000h` `AL = 07h` `CF = 0` `ZF = 0`

Kod İzleme: Dizi Toplama

```
1  org 100h
2  jmp start
3  vec1 db 1, 2, 5, 6
4  vec2 db 3, 5, 6, 1
5  vec3 db ?, ?, ?, ?
6  start:
7  lea si, vec1
8  lea bx, vec2
9  lea di, vec3
10 mov cx, 4
11 sum:
12 mov al, [si]
13 add al, [bx]
14 mov [di], al
15 inc si
16 inc bx
17 inc di
18 loop sum
▶  ret
```



`RET` : vec3 = **4, 7, 11, 7** (04h 07h 0Bh 07h).

Adım 17 / 17

DEĞİŞKENLER

`SI = 0106h` `BX = 010Ah` `DI = 010Eh` `CX = 0000h`

`AL = 07h` `CF = 0` `ZF = 0`

ÖRNEK: DİZİ TOPLAMA

Bellekte Program ve Veri

Adres	Baytlar	Anlamı
`0100`	`EB 0C`	`JMP 010Eh` +0Ch: 0102h + 0Ch
`0102`	`01 02 05 06`	`vec1` veri
`0106`	`03 05 06 01`	`vec2` veri
`010A`	`04 07 0B 07`	`vec3` çalıştıktan sonra
`010E`	`BE 02 01`	`MOV SI, 0102h` LEA → MOV
`0111`	`BB 06 01`	`MOV BX, 0106h`
`0114`	`BF 0A 01`	`MOV DI, 010Ah`

Adres	Baytlar	Anlamı
`0117`	`B9 04 00`	`MOV CX, 0004h` little-endian
`011A`	`8A 04`	`MOV AL, [SI]` sum:
`011C`	`02 07`	`ADD AL, [BX]`
`011E`	`88 05`	`MOV [DI], AL`
`0120`	`46 / 43 / 47`	`INC SI / BX / DI` 1'er bayt
`0123`	`E2 F5`	`LOOP 011Ah` F5h = −11
`0125`	`C3`	`RET`

emu8086 bellek penceresi (CS = DS = 0700h). Kod ve veri aynı kesimdedir; assembler LEA'yı daha kısa olan `MOV reg, sabit` olarak kodlamıştır.

BCD ARİTMETİĞİ

İkili Kodlu Onluk (BCD)

- **BCD** (Binary Coded Decimal): her onluk basamak ayrı bir 4 bitlik grupta (0000–1001) tutulur.
- **Paketsiz** (unpacked) BCD: bayt başına bir basamak. **Paketli** (packed): bayt başına iki basamak.
- İşlemci toplamayı ikili yapar; sonucun BCD'ye düzeltilmesi için özel komutlar vardır.

Gösterim	14 sayısı	Bayt(lar)
İkili	0000 1110b	`0Eh`
Paketsiz BCD	0000 0001 · 0000 0100	`01h 04h` (AH, AL)
Paketli BCD	0001 0100b	`14h`
ASCII	'1' '4'	`31h 34h`

BCD ARİTMETİĞİ

BCD Düzeltme Komutları

Komut	Ne zaman?	Etki
`AAA`	paketsiz toplamadan sonra	AL'yi BCD'ye düzelt, elde AH'ye
`AAS`	paketsiz çıkarmadan sonra	AL'yi düzelt, ödünç AH'den
`AAM`	paketsiz çarpmadan sonra	AH = AL / 10, AL = AL mod 10
`AAD`	paketsiz bölmeden **önce**	AL = AH × 10 + AL, AH = 0
`DAA` / `DAS`	paketli toplama / çıkarmadan sonra	AL'deki iki basamağı düzelt

Düzeltme komutları yalnızca **AL** (ve AH) üzerinde çalışır; bu yüzden BCD toplama/çıkarma bayt bayt yapılır. Örnek: 83h − 38h = 4Bh → `DAS` → **45h** (83 − 38 = 45).

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
▶ mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
19 ret
```

AH = 09h: birinci basamak 9.

Adım 1 / 12

DEĞİŞKENLER

AX = **0900h** BL = **00h** CF = **0** AF = **0** ZF = **0**

ÇIKTI

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
▶  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
19 ret
```

AL = 05h: ikinci basamak 5. AX = 0905h.

Adım 2 / 12

DEĞİŞKENLER

AX = 0905h | BL = 00h | CF = 0 | AF = 0 | ZF = 0

ÇIKTI

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
▶  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
19 ret
```

09h + 05h = **0Eh** (14): ikili doğru ama BCD değil (E > 9). Nibble eldesi yok → AF = 0.

Adım 3 / 12

DEĞİŞKENLER

AX = 090Eh | BL = 00h | CF = 0 | AF = 0 | ZF = 0

ÇIKTI

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
▶ xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
19 ret
```

AH = 0: onlar basamağı temizlendi. AX = 000Eh.

Adım 4 / 12

DEĞİŞKENLER

AX = 000Eh | BL = 00h | CF = 0 | AF = 0 | ZF = 1

ÇIKTI

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
▶  aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
19 ret
```

`AAA`: AL'nin düşük nibble'ı E > 9 → AL + 6 = 14h, **AH + 1**, AF = CF = 1; AL'nin üst nibble'ı silinir → **AX = 0104h** → "14".

Adım 5 / 12

DEĞİŞKENLER

AX = **0104h** BL = **00h** CF = **1** AF = **1** ZF = **1**

ÇIKTI

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
▶ or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
19 ret
```

Her basamağa 30h eklenir: AX = **3134h** = '1','4'.

Adım 6 / 12

DEĞİŞKENLER

AX = **3134h** | BL = **00h** | CF = **0** | AF = **1** | ZF = **0**

ÇIKTI

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
▶ mov     bl, al      ; '4' sakla
▶ mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
19 ret
```

BL = '4' saklanır, AL = '1'.

Adım 7 / 12

DEĞİŞKENLER

AX = **3131h** BL = **34h** CF = **0** AF = **1** ZF = **0**

ÇIKTI

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
▶ mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
19 ret
```

AH = 0Eh (teletype).

Adım 8 / 12

DEĞİŞKENLER

AX = **0E31h** | BL = **34h** | CF = **0** | AF = **1** | ZF = **0**

ÇIKTI

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
▶ int     10h
17 mov     al, bl
18 int     10h
19 ret
```

'1' yazılır.

Adım 9 / 12

DEĞİŞKENLER

AX = **0E31h** | BL = **34h** | CF = **0** | AF = **1** | ZF = **0**

ÇIKTI

```
1
```

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
▶ mov     al, bl
18 int     10h
19 ret
```

AL = '4'.

Adım 10 / 12

DEĞİŞKENLER

AX = 0E34h | BL = 34h | CF = 0 | AF = 1 | ZF = 0

ÇIKTI

```
1
```

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
▶  int     10h
19 ret
```

'4' yazılır. Ekranda **14**.

Adım 11 / 12

DEĞİŞKENLER

AX = **0E34h** BL = **34h** CF = **0** AF = **1** ZF = **0**

ÇIKTI

```
14
```

Kod İzleme: AAA ile 9 + 5

```
1  ; first number '9':
2  mov     ah, 09h
3  ; second number '5':
4  mov     al, 05h
5  ; al = al + ah = 09h + 05h = 0eh
6  add     al, ah
7  ; clear tens byte of bcd
8  xor     ah, ah
9  ; adjust result to bcd form, ah = 1, al = 4 -> '14'
10 aaa
11 ; --- sonucu yazdır (ek) ---
12 or      ax, 3030h   ; ASCII'ye çevir
13 mov     bl, al      ; '4' sakla
14 mov     al, ah      ; '1'
15 mov     ah, 0Eh
16 int     10h
17 mov     al, bl
18 int     10h
▶  ret
```

`RET` .

Adım 12 / 12

DEĞİŞKENLER

AX = **0E34h** BL = **34h** CF = **0** AF = **1** ZF = **0**

ÇIKTI

```
14
```

Kod İzleme: AAS ile 5 − 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
▶ mov     al, 05h
3  mov     bl, 08h
4  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
8  aas
9  ; --- yazdır ---
10 or      al, 30h
11 mov     ah, 0Eh
12 int     10h
13 mov     ah, 0
14 int     16h
15 ret
```

AL = 05h. emu8086 programı AX = 0000h ile başlatır.

Adım 1 / 9

DEĞİŞKENLER

AX = **0005h** BL = **00h** CF = **0** AF = **0** SF = **0**

ÇIKTI

Kod İzleme: AAS ile 5 – 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
2  mov     al, 05h
▶ mov     bl, 08h
4  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
8  aas
9  ; --- yazdır ---
10 or      al, 30h
11 mov     ah, 0Eh
12 int     10h
13 mov     ah, 0
14 int     16h
15 ret
```

BL = 08h.

Adım 2 / 9

DEĞİŞKENLER

AX = **0005h** | BL = **08h** | CF = **0** | AF = **0** | SF = **0**

ÇIKTI

Kod İzleme: AAS ile 5 − 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
2  mov     al, 05h
3  mov     bl, 08h
▶  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
8  aas
9  ; --- yazdır ---
10 or      al, 30h
11 mov     ah, 0Eh
12 int     10h
13 mov     ah, 0
14 int     16h
15 ret
```

5 − 8 = −3 = **FDh**. Ödünç: CF = 1; düşük nibble'da da ödünç: AF = 1.

Adım 3 / 9

DEĞİŞKENLER

AX = **00FDh** BL = **08h** CF = **1** AF = **1** SF = **1**

ÇIKTI

Kod İzleme: AAS ile 5 − 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
2  mov     al, 05h
3  mov     bl, 08h
4  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
▶  aas
9  ; --- yazdır ---
10 or      al, 30h
11 mov     ah, 0Eh
12 int     10h
13 mov     ah, 0
14 int     16h
15 ret
```

`AAS`: AF = 1 → AL − 6 = F7h, **AH − 1 = FFh**, CF = AF = 1; üst nibble silinir → **AX = FF07h**.

Adım 4 / 9

DEĞİŞKENLER

AX = **FF07h** | BL = **08h** | CF = **1** | AF = **1** | SF = **1**

ÇIKTI

Kod İzleme: AAS ile 5 − 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
2  mov     al, 05h
3  mov     bl, 08h
4  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
8  aas
9  ; --- yazdır ---
▶ or      al, 30h
11 mov     ah, 0Eh
12 int     10h
13 mov     ah, 0
14 int     16h
15 ret
```

AL = 37h = '7'.

Adım 5 / 9

DEĞİŞKENLER

AX = **FF37h** BL = **08h** CF = **0** AF = **1** SF = **0**

ÇIKTI

Kod İzleme: AAS ile 5 − 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
2  mov     al, 05h
3  mov     bl, 08h
4  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
8  aas
9  ; --- yazdır ---
10 or      al, 30h
▶  mov     ah, 0Eh
12 int     10h
13 mov     ah, 0
14 int     16h
15 ret
```

AH = 0Eh.

Adım 6 / 9

DEĞİŞKENLER

AX = **0E37h** | BL = **08h** | CF = **0** | AF = **1** | SF = **0**

ÇIKTI

Kod İzleme: AAS ile 5 – 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
2  mov     al, 05h
3  mov     bl, 08h
4  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
8  aas
9  ; --- yazdır ---
10 or      al, 30h
11 mov     ah, 0Eh
▶ 12 int     10h
13 mov     ah, 0
14 int     16h
15 ret
```

'7' yazılır.

Adım 7 / 9

DEĞİŞKENLER

AX = **0E37h** | BL = **08h** | CF = **0** | AF = **1** | SF = **0**

ÇIKTI

```
7
```

Kod İzleme: AAS ile 5 – 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
2  mov     al, 05h
3  mov     bl, 08h
4  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
8  aas
9  ; --- yazdır ---
10 or      al, 30h
11 mov     ah, 0Eh
12 int     10h
▶  mov     ah, 0
▶  int     16h
15 ret
```

Tuş beklenir.

Adım 8 / 9

DEĞİŞKENLER

AX = 0037h | BL = 08h | CF = 0 | AF = 1 | SF = 0

ÇIKTI

```
7
```

Kod İzleme: AAS ile 5 − 8

```
1  ; make 5 - 8, al = 0fdh (not in bcd form)
2  mov     al, 05h
3  mov     bl, 08h
4  sub     al, bl
5  ; convert to bcd, al = 7
6  ; and 1 is borrowed from ah,
7  ; like calculating 15 - 8:
8  aas
9  ; --- yazdır ---
10 or      al, 30h
11 mov     ah, 0Eh
12 int     10h
13 mov     ah, 0
14 int     16h
▶  ret
```

Sonuç “7, ödünç 1”: 15 − 8 = 7 gibi. Gerçek fark −3'tür; CF = 1 sonucun negatif olduğunu, 7'nin −3'ün 10'a tümleyeni olduğunu söyler.

Adım 9 / 9

DEĞİŞKENLER

AX = **0037h** BL = **08h** CF = **0** AF = **1** SF = **0**

ÇIKTI

```
7
```

BCD ARİTMETİĞİ

AAM ve AAD

AAM · çarpmadan sonra

```
mov al, 7
mov bl, 8
mul bl       ; AX = 0038h (56)
aam          ; AH = 05h, AL = 06h
```

AL'yi 10'a böler: bölüm AH'ye, kalan AL'ye. 56 → “5”, “6”.

AAD · bölmeden önce

```
mov ax, 0504h  ; '54' paketsiz
aad            ; AX = 0036h (54)
mov bl, 6
div bl         ; AL = 09h, AH = 00h
```

AL = AH × 10 + AL, AH = 0: iki basamak ikili sayıya çevrilir, sonra normal DIV.

Paketli BCD için `DAA`: `MOV AL,38h / ADD AL,45h` → AL = 7Dh, `DAA` → AL = **83h** (38 + 45 = 83).

Kod İzleme: AAM, AAD ve DAA

```
▶ MOV  AL, 6
▶ MOV  BL, 9
3  MUL  BL          ; AX = 6 × 9
4  AAM              ; basamaklara ayır
5  MOV  AX, 0702h   ; '72' paketsiz
6  AAD              ; ikiliye çevir
7  MOV  BL, 8
8  DIV  BL
9  MOV  AL, 19h     ; paketli 19
10 ADD  AL, 28h     ; + 28
11 DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
00h	06h	09h	0	0

AX = 0006h (ikili 6)

AL nibble'ları: 0 | 6

İki tek basamaklı sayı: AL = 6, BL = 9.

Adım 1 / 9

DEĞİŞKENLER

AX = **0006h** BL = **09h**

Kod İzleme: AAM, AAD ve DAA

```
1  MOV  AL, 6
2  MOV  BL, 9
▶  MUL  BL          ; AX = 6 × 9
4  AAM              ; basamaklara ayır
5  MOV  AX, 0702h   ; '72' paketsiz
6  AAD              ; ikiliye çevir
7  MOV  BL, 8
8  DIV  BL
9  MOV  AL, 19h     ; paketli 19
10 ADD  AL, 28h     ; + 28
11 DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
00h	36h	09h	?	0

AX = 0036h (ikili 54)

AL nibble'ları: 3 | 6

`MUL BL` : AX = 6 × 9 = 54 = 0036h. İkili sonuç; ekrana yazmak için basamaklara ayrılmalı.

Adım 2 / 9

DEĞİŞKENLER

AX = 0036h BL = 09h

Kod İzleme: AAM, AAD ve DAA

```
1  MOV  AL, 6
2  MOV  BL, 9
3  MUL  BL          ; AX = 6 × 9
▶  AAM              ; basamaklara ayır
5  MOV  AX, 0702h   ; '72' paketsiz
6  AAD              ; ikiliye çevir
7  MOV  BL, 8
8  DIV  BL
9  MOV  AL, 19h     ; paketli 19
10 ADD  AL, 28h     ; + 28
11 DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
05h	04h	09h	?	?

AX = 0504h (ikili 1284)

AL nibble'ları: 0 | 4

`AAM` : AL / 10 → AH = 5 (bölüm), AL = 4 (kalan). AX = 0504h: paketsiz “54”.

Adım 3 / 9

DEĞİŞKENLER

AX = **0504h** BL = **09h**

Kod İzleme: AAM, AAD ve DAA

```
1  MOV  AL, 6
2  MOV  BL, 9
3  MUL  BL          ; AX = 6 × 9
4  AAM              ; basamaklara ayır
▶  MOV  AX, 0702h   ; '72' paketsiz
6  AAD              ; ikiliye çevir
7  MOV  BL, 8
8  DIV  BL
9  MOV  AL, 19h     ; paketli 19
10 ADD  AL, 28h     ; + 28
11 DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
07h	02h	09h	?	?

AX = 0702h (ikili 1794)

AL nibble'ları: 0 | 2

AX = 0702h: paketsiz BCD “72” (AH = 7, AL = 2).

Adım 4 / 9

DEĞİŞKENLER

AX = 0702h BL = 09h

Kod İzleme: AAM, AAD ve DAA

```
1  MOV  AL, 6
2  MOV  BL, 9
3  MUL  BL          ; AX = 6 × 9
4  AAM              ; basamaklara ayır
5  MOV  AX, 0702h   ; '72' paketsiz
▶  AAD              ; ikiliye çevir
7  MOV  BL, 8
8  DIV  BL
9  MOV  AL, 19h     ; paketli 19
10 ADD  AL, 28h     ; + 28
11 DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
00h	48h	09h	?	?

AX = 0048h (ikili 72)

AL nibble'ları: 4 | 8

`AAD` : AL = 7 × 10 + 2 = 72 = 48h, AH = 0. Bölmeden önce ikiliye çevrildi.

Adım 5 / 9

DEĞİŞKENLER

AX = **0048h** BL = **09h**

Kod İzleme: AAM, AAD ve DAA

```
1  MOV  AL, 6
2  MOV  BL, 9
3  MUL  BL          ; AX = 6 × 9
4  AAM              ; basamaklara ayır
5  MOV  AX, 0702h   ; '72' paketsiz
6  AAD              ; ikiliye çevir
▶  MOV  BL, 8
▶  DIV  BL
9  MOV  AL, 19h     ; paketli 19
10 ADD  AL, 28h     ; + 28
11 DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
00h	09h	08h	?	?

AX = 0009h (ikili 9)

AL nibble'ları: 0 | 9

`DIV BL` : 72 / 8 → AL = 9, AH = 0 (kalan). AAD olmasaydı 0702h = 1794 bölünürdü.

Adım 6 / 9

DEĞİŞKENLER

AX = 0009h BL = 08h

Kod İzleme: AAM, AAD ve DAA

```
1  MOV  AL, 6
2  MOV  BL, 9
3  MUL  BL          ; AX = 6 × 9
4  AAM              ; basamaklara ayır
5  MOV  AX, 0702h   ; '72' paketsiz
6  AAD              ; ikiliye çevir
7  MOV  BL, 8
8  DIV  BL
▶ MOV  AL, 19h      ; paketli 19
10 ADD  AL, 28h     ; + 28
11 DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
00h	19h	08h	?	?

AX = 0019h (ikili 25)

AL nibble'ları: 1 | 9

AL = 19h: paketli BCD “19” (bir baytta iki basamak).

Adım 7 / 9

DEĞİŞKENLER

AX = 0019h BL = 08h

Kod İzleme: AAM, AAD ve DAA

```
1  MOV  AL, 6
2  MOV  BL, 9
3  MUL  BL          ; AX = 6 × 9
4  AAM              ; basamaklara ayır
5  MOV  AX, 0702h   ; '72' paketsiz
6  AAD              ; ikiliye çevir
7  MOV  BL, 8
8  DIV  BL
9  MOV  AL, 19h     ; paketli 19
▶ ADD  AL, 28h      ; + 28
11 DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
00h	41h	08h	1	0

AX = 0041h (ikili 65)

AL nibble'ları: 4 | 1

19h + 28h = 41h: ikili toplama. 9 + 8 = 17 alt nibble'dan taştı → **AF = 1**; 41h BCD olarak yanlış.

Adım 8 / 9

DEĞİŞKENLER

AX = 0041h BL = 08h

Kod İzleme: AAM, AAD ve DAA

```
1  MOV  AL, 6
2  MOV  BL, 9
3  MUL  BL          ; AX = 6 × 9
4  AAM              ; basamaklara ayır
5  MOV  AX, 0702h   ; '72' paketsiz
6  AAD              ; ikiliye çevir
7  MOV  BL, 8
8  DIV  BL
9  MOV  AL, 19h     ; paketli 19
10 ADD  AL, 28h     ; + 28
▶  DAA              ; BCD düzelt
12 RET
```

AH	AL	BL	AF	CF
00h	47h	08h	1	0

AX = 0047h (ikili 71)

AL nibble'ları: 4 | 7

`DAA` : AF = 1 → AL + 06h = 47h. Sonuç “47” = 19 + 28 ✓

Adım 9 / 9

DEĞİŞKENLER

AX = **0047h** BL = **08h**

Kod İzleme: °C → °F

```
1  org 100h
2  start:
3  ;celsius to fahrenheit according to f = c * 9 / 5 + 32
▶  mov cl, 15
5  mov al, 9
6  imul cl
7  mov cl, 5
8  idiv cl
9  add al, 32
10 ret
```

CL = 15 (°C) = 0Fh.

Adım 1 / 7

DEĞİŞKENLER

AX = **0000h** CL = **0Fh**

CF = **0** OF = **0**

f = c × 9 / 5 + 32. Önce çarpma, sonra bölme: 9/5 tam sayı bölmesinde 1 olurdu ve sonuç yanlış çıkardı.

Kod İzleme: °C → °F

```
1  org 100h
2  start:
3  ;celsius to fahrenheit according to f = c * 9 / 5 + 32
4  mov cl, 15
▶  mov al, 9
6  imul cl
7  mov cl, 5
8  idiv cl
9  add al, 32
10 ret
```

AL = 9.

Adım 2 / 7

DEĞİŞKENLER

AX = 0009h | CL = 0Fh | CF = 0 | OF = 0

f = c × 9 / 5 + 32. Önce çarpma, sonra bölme: 9/5 tam sayı bölmesinde 1 olurdu ve sonuç yanlış çıkardı.

Kod İzleme: °C → °F

```
1  org 100h
2  start:
3  ;celsius to fahrenheit according to f = c * 9 / 5 + 32
4  mov cl, 15
5  mov al, 9
6  imul cl
7  mov cl, 5
8  idiv cl
9  add al, 32
10 ret
```

`IMUL CL`: AX = 9 × 15 = 135 = **0087h**. 135 işaretli bayta sığmaz (>127) → **CF = OF = 1**; ama sonuç AX'te eksiksiz.

Adım 3 / 7

DEĞİŞKENLER

AX = **0087h** | CL = **0Fh** | CF = **1** | OF = **1**

f = c × 9 / 5 + 32. Önce çarpma, sonra bölme: 9/5 tam sayı bölmesinde 1 olurdu ve sonuç yanlış çıkardı.

Kod İzleme: °C → °F

```
1  org 100h
2  start:
3  ;celsius to fahrenheit according to f = c * 9 / 5 + 32
4  mov cl, 15
5  mov al, 9
6  imul cl
▶  mov cl, 5
8  idiv cl
9  add al, 32
10 ret
```

CL = 5 (bölen).

Adım 4 / 7

DEĞİŞKENLER

AX = **0087h** CL = **05h**

CF = **1** OF = **1**

f = c × 9 / 5 + 32. Önce çarpma, sonra bölme: 9/5 tam sayı bölmesinde 1 olurdu ve sonuç yanlış çıkardı.

Kod İzleme: °C → °F

```
1  org 100h
2  start:
3  ;celsius to fahrenheit according to f = c * 9 / 5 + 32
4  mov cl, 15
5  mov al, 9
6  imul cl
7  mov cl, 5
▶ idiv cl
9  add al, 32
10 ret
```

`IDIV CL` : AX / 5 = 135 / 5 → **AL = 1Bh (27)**, AH = 0 (kalan). Bayraklar tanımsız.

Adım 5 / 7

DEĞİŞKENLER

AX = **001Bh** | CL = **05h** | CF = **1** | OF = **1**

f = c × 9 / 5 + 32. Önce çarpma, sonra bölme: 9/5 tam sayı bölmesinde 1 olurdu ve sonuç yanlış çıkardı.

Kod İzleme: °C → °F

```
1  org 100h
2  start:
3  ;celsius to fahrenheit according to f = c * 9 / 5 + 32
4  mov cl, 15
5  mov al, 9
6  imul cl
7  mov cl, 5
8  idiv cl
▶  add al, 32
10 ret
```

27 + 32 = **59 = 3Bh** → AX = 003Bh. 15 °C = 59 °F ✓

Adım 6 / 7

DEĞİŞKENLER

AX = **003Bh** CL = **05h**

CF = **0** OF = **0**

f = c × 9 / 5 + 32. Önce çarpma, sonra bölme: 9/5 tam sayı bölmesinde 1 olurdu ve sonuç yanlış çıkardı.

Kod İzleme: °C → °F

```
1 org 100h
2 start:
3 ;celsius to fahrenheit according to f = c * 9 / 5 + 32
4 mov cl, 15
5 mov al, 9
6 imul cl
7 mov cl, 5
8 idiv cl
9 add al, 32
▶ ret
```

`RET` . emu8086'da son durum: AX = 003Bh, CX = 0005h.

Adım 7 / 7

DEĞİŞKENLER

AX = **003Bh** CL = **05h**

CF = **0** OF = **0**

f = c × 9 / 5 + 32. Önce çarpma, sonra bölme: 9/5 tam sayı bölmesinde 1 olurdu ve sonuç yanlış çıkardı.

Kod İzleme: °F → °C

```
1  org 100h
2  start:
3  ;fahrenheit to celsius according to: c = (f - 32) * 5 / 9
▶  mov cl, 59
5  sub cl, 32
6  mov al, 5
7  imul cl
8  mov cl, 9
9  idiv cl
10 ret
```

CL = 59 (°F) = 3Bh.

Adım 1 / 7

DEĞİŞKENLER

AX = **0000h** CL = **3Bh**

CF = **0** OF = **0**

Kod İzleme: °F → °C

```
1  org 100h
2  start:
3  ;fahrenheit to celsius according to: c = (f - 32) * 5 / 9
4  mov cl, 59
5  sub cl, 32
6  mov al, 5
7  imul cl
8  mov cl, 9
9  idiv cl
10 ret
```

59 − 32 = **27 = 1Bh**.

Adım 2 / 7

DEĞİŞKENLER

AX = **0000h** | CL = **1Bh** | CF = **0** | OF = **0**

Kod İzleme: °F → °C

```
1  org 100h
2  start:
3  ;fahrenheit to celsius according to: c = (f - 32) * 5 / 9
4  mov cl, 59
5  sub cl, 32
▶  mov al, 5
7  imul cl
8  mov cl, 9
9  idiv cl
10 ret
```

AL = 5.

Adım 3 / 7

DEĞİŞKENLER

AX = **0005h** CL = **1Bh**

CF = **0** OF = **0**

Kod İzleme: °F → °C

```
1  org 100h
2  start:
3  ;fahrenheit to celsius according to: c = (f - 32) * 5 / 9
4  mov cl, 59
5  sub cl, 32
6  mov al, 5
▶  imul cl
8  mov cl, 9
9  idiv cl
10 ret
```

AX = 5 × 27 = 135 = **0087h**; CF = OF = 1 (AL'ye sığmadı).

Adım 4 / 7

DEĞİŞKENLER

AX = **0087h** | CL = **1Bh** | CF = **1** | OF = **1**

Kod İzleme: °F → °C

```
1  org 100h
2  start:
3  ;fahrenheit to celsius according to: c = (f - 32) * 5 / 9
4  mov cl, 59
5  sub cl, 32
6  mov al, 5
7  imul cl
▶  mov cl, 9
9  idiv cl
10 ret
```

CL = 9.

Adım 5 / 7

DEĞİŞKENLER

AX = **0087h** CL = **09h**

CF = **1** OF = **1**

Kod İzleme: °F → °C

```
1  org 100h
2  start:
3  ;fahrenheit to celsius according to: c = (f - 32) * 5 / 9
4  mov cl, 59
5  sub cl, 32
6  mov al, 5
7  imul cl
8  mov cl, 9
▶  idiv cl
10 ret
```

135 / 9 = **15 = 0Fh** → AL, kalan AH = 0.
59 °F = 15 °C ✓

Adım 6 / 7

DEĞİŞKENLER

AX = **000Fh** | CL = **09h** | CF = **1** | OF = **1**

Kod İzleme: °F → °C

```
1 org 100h
2 start:
3 ;fahrenheit to celsius according to: c = (f - 32) * 5 / 9
4 mov cl, 59
5 sub cl, 32
6 mov al, 5
7 imul cl
8 mov cl, 9
9 idiv cl
▶ ret
```

`RET` .

Adım 7 / 7

DEĞİŞKENLER

AX = **000Fh** CL = **09h**

CF = **1** OF = **1**

Neden IMUL / IDIV? Sınırlar

Negatif sıcaklıklar

c = −10: `IMUL` → AX = FFA6h (−90), `IDIV` → AL = EEh (−18), +32 → **0Eh = 14** °F ✓

MUL/DIV kullanılsaydı F6h, 246 sanılırdı ve sonuç anlamsız olurdu.

Tam sayı bölmesi

IDIV kesirli kısmı atar (sıfıra doğru): c = 1 → 9/5 = 1 → 33 °F (doğrusu 33,8). Yuvarlama için kalan (AH) incelenebilir.

°C → °F geçerli aralık

Sonuç işaretli bayt AL'de: **c** ∈ **[−71, 53]**. c = 54'te AL = 81h (OF = 1, yanlış); c = 72'de bölüm AL'ye sığmaz → **bölme hatası**.

°F → °C geçerli aralık

f − 32 bayta sığmalı: **f** ∈ **[−96, 127]**. Daha geniş aralık için sözcük işlemleri (AX, CWD, IDIV BX) kullanılır.

Laboratuvar: Aritmetik ve Mantık

CF ve OF

KOD · **16 bayt**

```
 1 ORG 100h
 2 MOV  AL, 9Ch      ; 156 ya …
 3 ADD  AL, 70h      ; + 112
 4 MOV  AL, 50h      ; 80
 5 ADD  AL, 50h      ; + 80
 6 ADD  AL, AL       ; A0h + A…
 7 MOV  AL, 0FFh
 8 INC  AL           ; CF'ye d…
 9 CMC               ; CF ters…
10 RET
```

YAZMAÇLAR

AX	0000	AH	00	AL	00
BX	0000	BH	00	BL	00
CX	0010	CH	00	CL	10
DX	0000	DH	00	DL	00

SI	0000	DI	0000	BP	0000
SP	FFFE	CS	0700	IP	0100
DS	0700	ES	0700	SS	0700

BAYRAKLAR

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

CS:IP = 0700:0100 · adım 0

BELLEK **DS = 0700h**

```
0100 B0 9C 04 70 B0 50 04 50
0108 02 C0 B0 FF FE C0 F5 C3
0110 00 00 00 00 00 00 00 00
0118 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00
```

YIĞIN **SS:SP**

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

ÇIKTI

Program yüklendi: 16 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (0010h) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet: Hangi Komut Hangi Bayrağı Etkiler?

Komut	CF	ZF	SF	OF	PF	AF
`ADD` `SUB` `CMP` `ADC` `SBB`	✓	✓	✓	✓	✓	✓
`NEG`	✓	✓	✓	✓	✓	✓
`INC` `DEC`	—	✓	✓	✓	✓	✓
`AND` `OR` `XOR` `TEST`	0	✓	✓	0	✓	?
`NOT`	—	—	—	—	—	—
`MUL` `IMUL`	✓	?	?	✓	?	?
`DIV` `IDIV`	?	?	?	?	?	?
`SHL` `SHR` `SAR`	✓	✓	✓	✓[1]	✓	?
`ROL` `ROR` `RCL` `RCR`	✓	—	—	✓[1]	—	—
`AAA` `AAS`	✓	?	?	?	?	✓
`MOV` `LEA` `LOOP`	—	—	—	—	—	—

✓: sonuca göre · 0: temizlenir · —: değişmez · ?: tanımsız · [1]: yalnız kaydırma sayısı 1 iken tanımlı.

ÖZET

Özet

Bayraklar

CF işaretsiz, OF işaretli taşma; ZF sıfır; SF MSB; PF düşük baytta çift sayıda 1; AF nibble eldesi.

1. grup

Sonuç ilk işlenende; CMP/TEST yalnız bayrak. Mantıksal komutlar CF = OF = 0.

2. grup

Örtük AL/AX/DX. Bayt: AX = AL × op, AL = AX / op (AH kalan). Sözcük: DX:AX.

3. grup

INC/DEC CF'yi korur; NOT bayrak değiştirmez; NEG ikiye tümleyen.

Kaydırma ve döndürme

Sayı 1 ya da CL. SHL ×2, SHR işaretsiz ÷2, SAR işaretli ÷2 (−∞'a yuvarlar); döndürmede bit kaybolmaz, RCL/RCR CF ile 9 (17) bitlik halka kurar.

BCD

AAA/AAS toplama/çıkarma sonrası, AAM çarpma sonrası, AAD bölme öncesi.

Tuzaklar

Bölmeden önce AH/DX hazırla; işaretli veride IMUL/IDIV; aralık taşmalarını bayraklarla denetle.

ÖZET

Alıştırmalar

1. `MOV AL, 0C0h` / `ADD AL, 0C0h` sonrası AL, CF, OF, SF, ZF, PF nedir?
2. `CMP AL, BL` sonrası işaretsiz "AL < BL" hangi bayrakla, işaretli "AL < BL" hangi bayraklarla anlaşılır?
3. AX = 0FFFFh, BX = 0FFFFh iken `MUL BX` sonrası DX, AX, CF nedir?
4. AX = 0100h, BL = 0 iken `DIV BL` ne olur?
5. Dizi toplama programını sözcük (DW) dizileri için değiştirin.
6. Bir bayttaki 1 bitlerini sayan programı SHL/TEST ve döngüyle yazın. PF ile karşılaştırın.

▶ **İpuçları / cevaplar**

ÖZET

Alıştırmalar (2)

7. DX = 0005h, AX = 0000h, BX = 0005h iken `DIV BX` ne olur?
8. `MOV AL, 70h` / `ADD AL, 30h` sonrası AL ve OF nedir? Doğru toplamı 16 bitte nasıl elde edersiniz?
9. 00012345h − 0000ABCDh işlemini SUB/SBB ile yapın. SUB sonrası CF kaçtır?
10. 32 × 32 bit çarpmada dört kısmi çarpım sonucun hangi sözcüklerine eklenir?
11. DX = 0000h, AX = 0FFFh, CF = 0 iken `SHL AX, 1` ve `RCL DX, 1` sonrası DX:AX nedir?

▶ **İpuçları / cevaplar**

Kaynaklar

- Intel, *The 8086 Family User's Manual*, 1979 — Instruction Set bölümü.
- B. B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009 — Bölüm 5: Aritmetik, mantık, kaydırma ve döndürme komutları.
- emu8086 yardım belgeleri: “8086 instructions” ve “Arithmetic and logic instructions” sayfaları.
- K. R. Irvine, *Assembly Language for x86 Processors*, 7. baskı, Pearson, 2014 — Bölüm 7: Kaydırma ve döndürme komutları.
- A. S. Tanenbaum, *Modern Operating Systems*, 3. baskı, Pearson, 2008.
- M. Bodur, *Mikroişlemcilere Giriş: Assembler ile Yazılım ve Arayüz*, EMO, 2014.
- Ş. Ç. Öztürk, O. Ergin, *Intel 8086 ile Mikroişlemci Programlamaya Giriş*, 2009.

Son

Bölüm 6: Aritmetik, Mantık ve Kaydırma Komutları

Sonraki bölüm: Bölüm 7: Akış Kontrolü · Terim sözlüğü