

Bölüm 1: 8086 Mimarisi

Mikroişlemciler

8086'ya genel bakış

Sistem yolları

BIU ve EU

Önden getirme kuyruğu

Getir-çöz-yürüt

Yazmaç kümesi

Bayrak yazmacı

Kesimli bellek

Fiziksel adres hesabı

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Genel özellikler

8086'nın veri/adres yolu genişliğini, adresleyebildiği bellek miktarını ve x86 ailesindeki yerini açıklamak.

İç yapı

BIU ve EU birimlerinin görevlerini, önden getirme kuyruğunu ve boru hattı fikrini anlatmak.

Yazmaçlar

14 yazmacı gruplarına ayırmak; genel amaçlı, işaretçi/indis, kesim yazmaçlarını ve bayrakları tanımak.

Adres hesabı

$\text{kesim} \times 10h + \text{bağlı konum}$ ile 20 bitlik fiziksel adresi elle ve adım adım hesaplamak.

8086'ya Genel Bakış

- Intel'in **1978**'de tanıttığı **16 bitlik** mikroişlemci; **x86** ailesinin ilk üyesi.
- Yazmaçları, ALU'su ve veri yolu 16 bittir; aynı anda 16 bitlik veri işler.
- **20 bitlik adres yolu** → $2^{20} = 1\,048\,576$ bayt = **1 MB** bellek.
- **16 bitlik veri yolu**: işlemci, bellek ve G/Ç aygıtları arasında tek seferde 2 bayt taşır.
- **CISC** (karmaşık komut kümesi) mimarisi: değişken uzunlukta (1–6 bayt) komutlar, bellekte işlem yapabilen komutlar.
- Kişisel bilgisayarların temelini oluşturan komut kümesi bugün hâlâ geriye dönük uyumlulukla yaşıyor.

16 bit
veri yolu

20 bit
adres yolu

1 MB
bellek

40 pin
DIP paket

5–10 MHz
saat

~29 000
transistör

+5 V
tek besleme

HMOS
teknoloji

8 bitlik dış veri yolu kullanan kardeşi 8088'den farklı olarak 8086 tamamen 16 bittir.

GENEL BAKIŞ

x86 Ailesinde 8086

İşlemci	Yıl	İç / dış veri yolu	Adres yolu	Bellek	Not
8086	1978	16 / 16 bit	20 bit	1 MB	6 baytlık önden getirme kuyruğu
8088	1979	16 / 8 bit	20 bit	1 MB	4 baytlık kuyruk; IBM PC (1981)
80186	1982	16 / 16 bit	20 bit	1 MB	tümleşik çevre birimleri
80286	1982	16 / 16 bit	24 bit	16 MB	korumalı mod
80386	1985	32 / 32 bit	32 bit	4 GB	32 bit x86 (IA-32)

Sonraki tüm x86 işlemciler açılışta **gerçek modda** (real mode) 8086 gibi davranır. Bu derste emu8086 ile yazacağımız programlar da 8086 gerçek modunda çalışır.

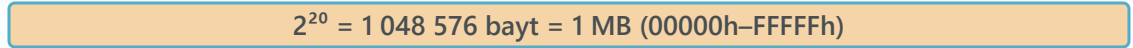
Neden 20 Bit Adres?

16 bit adres



$2^{16} = 65\,536$ bayt = 64 KB (0000h–FFFFh)

20 bit adres



$2^{20} = 1\,048\,576$ bayt = 1 MB (00000h–FFFFFh)

Ama 8086'nın yazmaçları 16 bit → tek yazmaç en çok 64 KB'yi gösterebilir.

Çözüm: iki 16 bitlik değer → kesim (segment) × 16 + bağlı konum (offset) = 20 bit.

n bit adres

2^n farklı bayt adreslenir. 1 KB = 2^{10} , 1 MB = 2^{20} bayt.

Veri yolu ≠ adres yolu

Veri yolu 16 bit (bir seferde ne kadar veri), adres yolu 20 bit (ne kadar bellek).

Onaltılık gösterim

20 bit = 5 onaltılık basamak: en düşük adres 00000h, en yüksek FFFFFh.

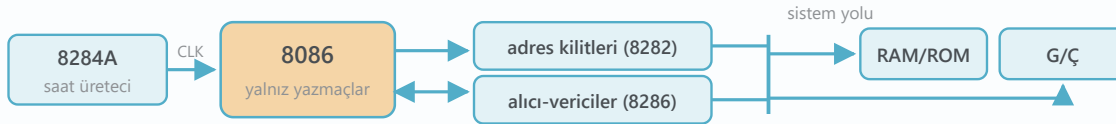
Bellek Yapısı: İşlemci Tek Başına Değil

Mikroişlemci (8086)

- CPU'nun tüm işlevlerini tek bir tümleşik devrede toplar.
- İçinde **RAM veya ROM yoktur**; yalnızca yazmaçları vardır.
- Ara ve nihai sonuçlar yazmaçlarda tutulur.
- Bellek ve G/Ç'ye **sistem yolu** (system bus) üzerinden erişir.
- Çalışmak için dışarıdan bellek, saat üretici, tampon/kilit devreleri gerekir.

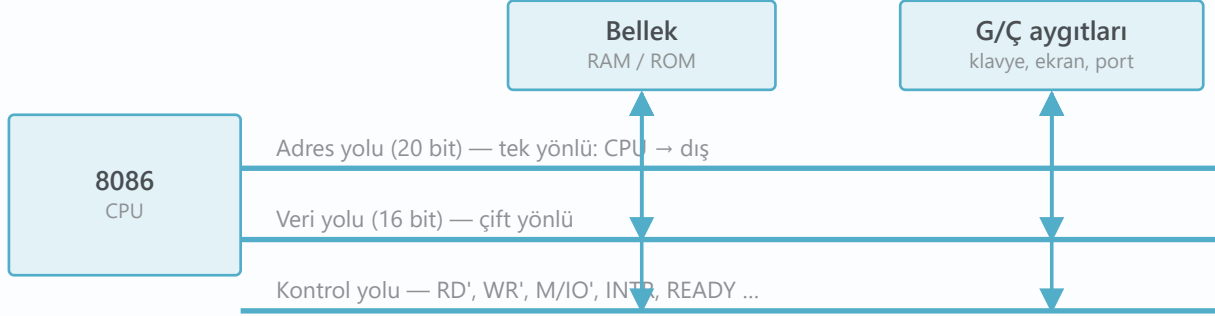
Mikrodenetleyici (ör. 8051, AVR)

- CPU + program belleği (ROM/Flash) + RAM + zamanlayıcı + G/Ç portları tek yongada.
- Tek başına bir sistem gibi çalışabilir.
- Gömülü uygulamalar: çamaşır makinesi, kumanda, sensör düğümü.
- Daha düşük işlem gücü, düşük maliyet ve güç tüketimi.



8086 tabanlı bir bilgisayar = 8086 + 8284A saat üretici + adres kilitleri + veri yolu alıcı-vericileri + bellek + G/Ç birimleri.

Sistem Yolları



Her yolun "geniřlięi" = içindeki paralel hat (bit) sayısı

Yol	Ne taşır?	8086'da
Adres	okunacak/yazılacak bellek konumu veya G/Ç port numarası	20 bit → 1 MB bellek; G/Ç için 16 bit → 64 K port
Veri	işlemci ile bellek/G/Ç arasındaki komut ve veri baytları	16 bit, tek seferde bir sözcük (2 bayt)
Kontrol	okuma, yazma, kesme istekleri, durum bilgisi	RD', WR', M/IO', ALE, INTR, READY ...

Kod İzleme: Yollar Üzerinde Bellek ve G/Ç

```
1 org 100h
```

```
▶ mov ax, 1000h
```

```
▶ mov ds, ax
```

```
4 mov bx, 0200h
```

```
5 mov al, [bx]      ; bellek oku
```

```
6 out 50h, al      ; G/Ç yaz
```

```
7 in al, 51h       ; G/Ç oku
```

```
8 mov [bx+1], al   ; bellek yaz
```

```
9 ret
```



Veri kesimi **DS = 1000h** (taban 10000h). Bellekte 10200h adresinde **5Ah**, 51h numaralı G/Ç portunda **3Ch** bulunuyor (örnek değerler).

Adım 1 / 6

DEĞİŞKENLER

DS = 1000h

BX = 0000h

AL = 00h

Kod İzleme: Yollar Üzerinde Bellek ve G/Ç

```
1 org 100h
2 mov ax, 1000h
3 mov ds, ax
▶ 4 mov bx, 0200h
5 mov al, [bx] ; bellek oku
6 out 50h, al ; G/Ç yaz
7 in al, 51h ; G/Ç oku
8 mov [bx+1], al ; bellek yaz
9 ret
```



BX = 0200h. Yalnız komut getirme; veri için yol kullanılmadı.

Adım 2 / 6

DEĞİŞKENLER

DS = 1000h

BX = 0200h

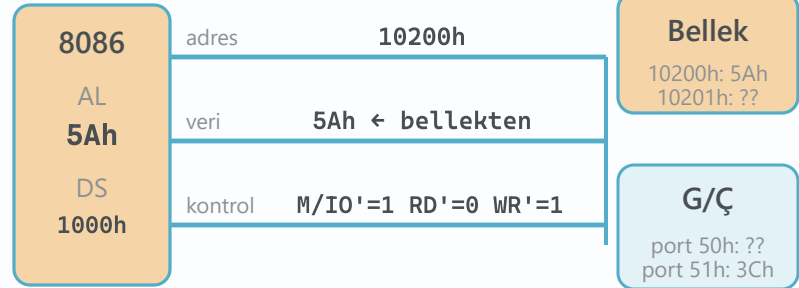
AL = 00h

Kod İzleme: Yollar Üzerinde Bellek ve G/Ç

```

1 org 100h
2 mov ax, 1000h
3 mov ds, ax
4 mov bx, 0200h
5 ► mov al, [bx] ; bellek oku
6 out 50h, al ; G/Ç yaz
7 in al, 51h ; G/Ç oku
8 mov [bx+1], al ; bellek yaz
9 ret

```



Bellek okuma: adres yolunda 20 bitlik **10200h**, **M/I0' = 1**, **RD' = 0**. Bellek 5Ah'yi veri yoluna koyar → AL = 5Ah.

Adım 3 / 6

DEĞİŞKENLER

DS = 1000h

BX = 0200h

AL = 5Ah

M/I0' = 1

RD' = 0

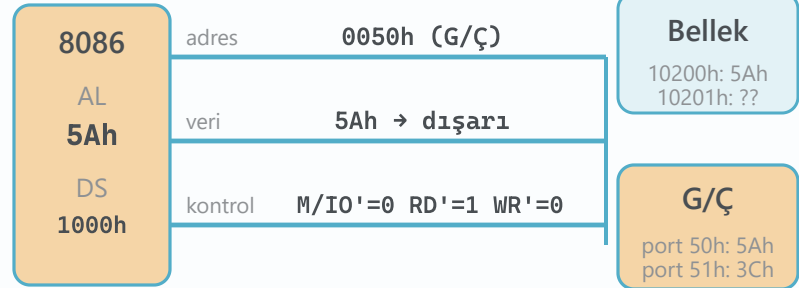
WR' = 1

Kod İzleme: Yollar Üzerinde Bellek ve G/Ç

```

1 org 100h
2 mov ax, 1000h
3 mov ds, ax
4 mov bx, 0200h
5 mov al, [bx]      ; bellek oku
▶ out 50h, al      ; G/Ç yaz
7 in al, 51h        ; G/Ç oku
8 mov [bx+1], al    ; bellek yaz
9 ret

```



G/Ç yazma: adres yolunda yalnız port numarası (**0050h**, 16 bit), **$M/IO' = 0$** , **$WR' = 0$** ; veri yolunda AL = 5Ah. Bellek bu döngüyü yok sayar.

Adım 4 / 6

DEĞİŞKENLER

DS = 1000h

BX = 0200h

AL = 5Ah

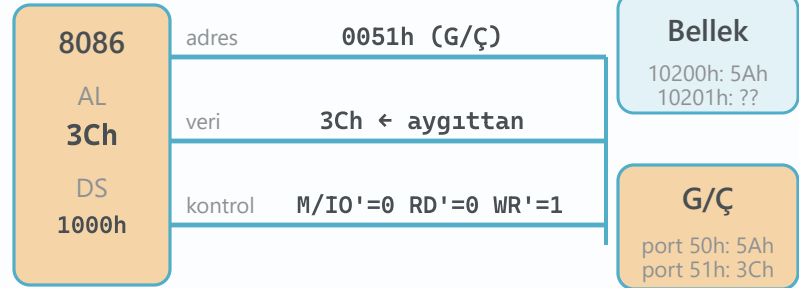
$M/IO' = 0$

$RD' = 1$

$WR' = 0$

Kod İzleme: Yollar Üzerinde Bellek ve G/Ç

```
1 org 100h
2 mov ax, 1000h
3 mov ds, ax
4 mov bx, 0200h
5 mov al, [bx]      ; bellek oku
6 out 50h, al      ; G/Ç yaz
7 ► in al, 51h      ; G/Ç oku
8 mov [bx+1], al   ; bellek yaz
9 ret
```



G/Ç okuma: port **0051h**, $M/IO' = 0$, $RD' = 0$. Aygıt 3Ch'yi veri yoluna koyar → **AL = 3Ch**.

Adım 5 / 6

DEĞİŞKENLER

DS = 1000h

BX = 0200h

AL = 3Ch

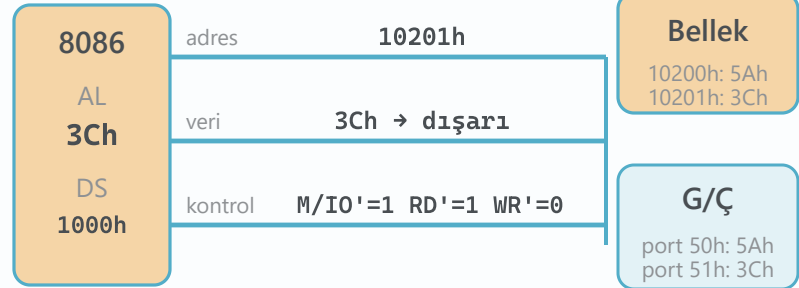
M/IO' = 0

RD' = 0

WR' = 1

Kod İzleme: Yollar Üzerinde Bellek ve G/Ç

```
1 org 100h
2 mov ax, 1000h
3 mov ds, ax
4 mov bx, 0200h
5 mov al, [bx]      ; bellek oku
6 out 50h, al      ; G/Ç yaz
7 in al, 51h       ; G/Ç oku
▶ 8 mov [bx+1], al  ; bellek yaz
9 ret
```



Bellek yazma: $EA = BX + 1 = 0201h \rightarrow FA = \mathbf{10201h}$, $M/IO' = 1$, $WR' = 0$, veri 3Ch. Aynı adres ve veri yolları hem bellek hem G/Ç için kullanılır; ayrımı M/IO' yapar.

Adım 6 / 6

DEĞİŞKENLER

DS = 1000h

BX = 0200h

AL = 3Ch

M/IO' = 1

RD' = 1

WR' = 0

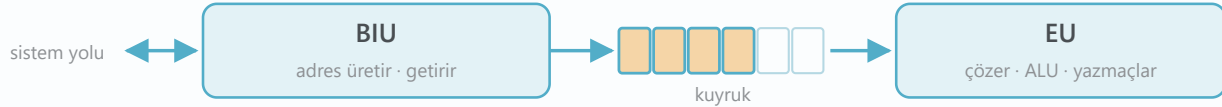
İki Temel Birim: BIU ve EU

Yol Arayüz Birimi (BIU · Bus Interface Unit)

- Dış dünyayla tüm iletişimi yürütür: komutları bellekten **getirir** (fetch).
- İşlenenleri bellekten okur, sonuçları belleğe/G/Ç'ye yazar.
- Kesim yazmaçları + IP ile **20 bitlik fiziksel adresi** üretir.
- 6 baytlık **önden getirme kuyruğunu** doldurur.

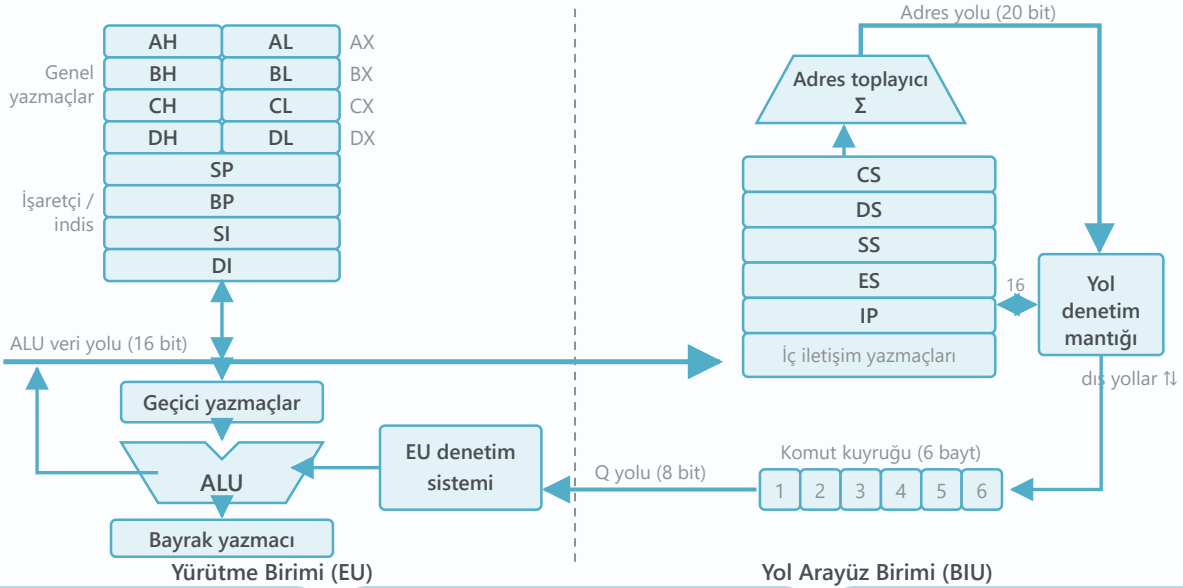
Yürütme Birimi (EU · Execution Unit)

- Kuyruktan sıradaki komut baytını alır, **çözer** (decode) ve **yürütür**.
- ALU ile aritmetik ve mantıksal işlemleri yapar.
- Genel amaçlı yazmaçları ve bayrakları günceller.
- Bellekte bir işlenen gerekiyorsa BIU'dan ister; dış yollara doğrudan bağlı değildir.



İki birim **paralel** çalışır: EU bir komutu yürütürken BIU sonraki komut baytlarını getirir. Bu, 8086'nın basit **boru hattıdır** (pipelining).

8086 Blok Diyagramı



Sol: EU

genel yazmaçlar, ALU,
bayraklar, kontrol sistemi

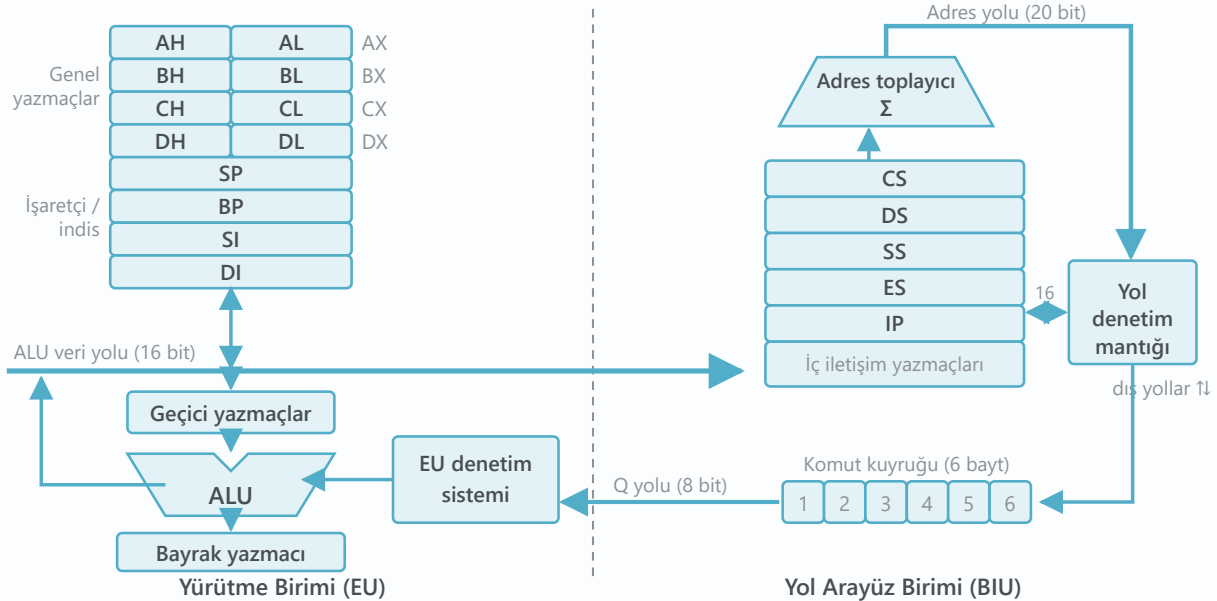
Sağ: BIU

kesim yazmaçları, IP, adres
toplayıcı, kuyruk, yol kontrolü

Köprü

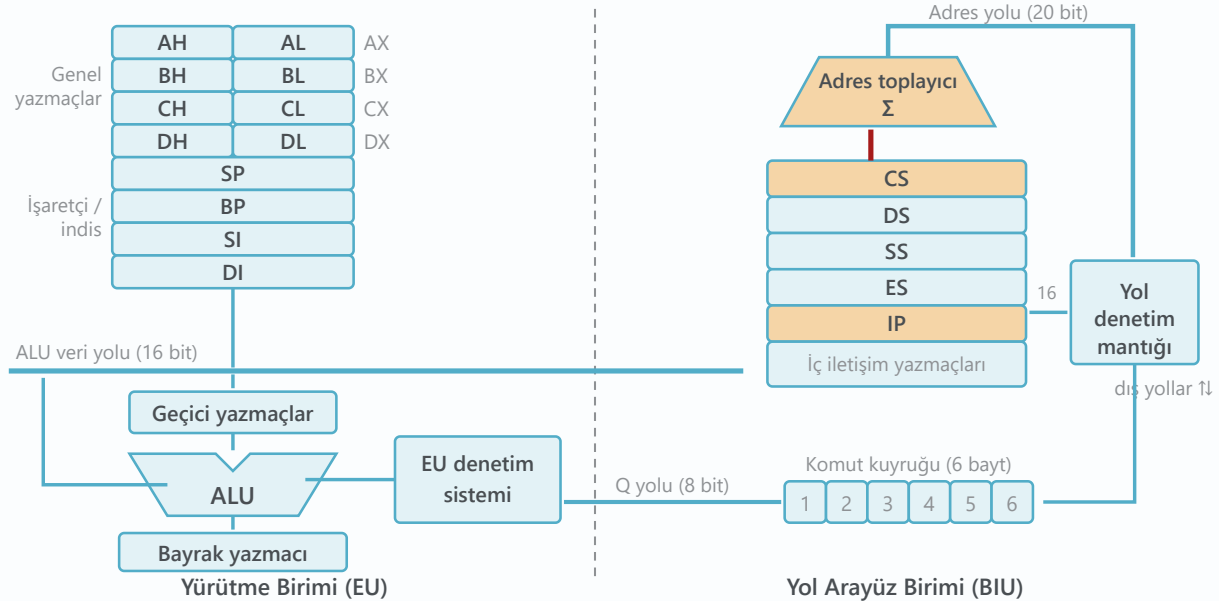
EU, kuyruktan 8 bitlik Q
yoluyla komut baytı alır

8086 Mimarisinin İşleyişi



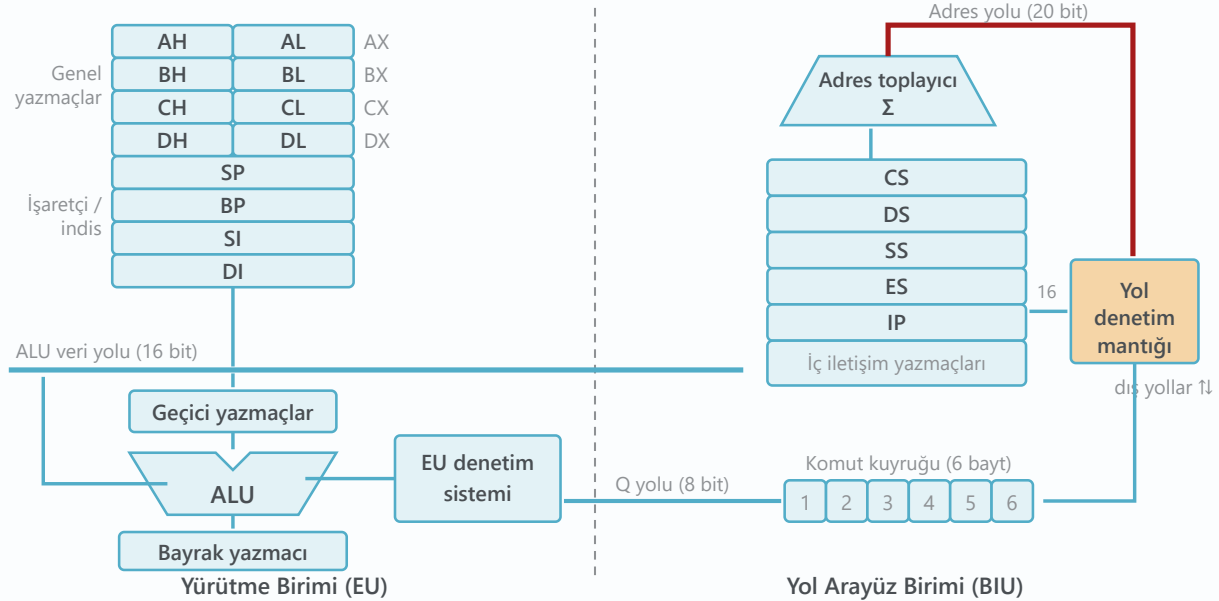
Genel görünüm. Bir komutun yolculuğunu adım adım izleyelim: **ADD AX, BX** .

8086 Mimarisinin İşleyişi



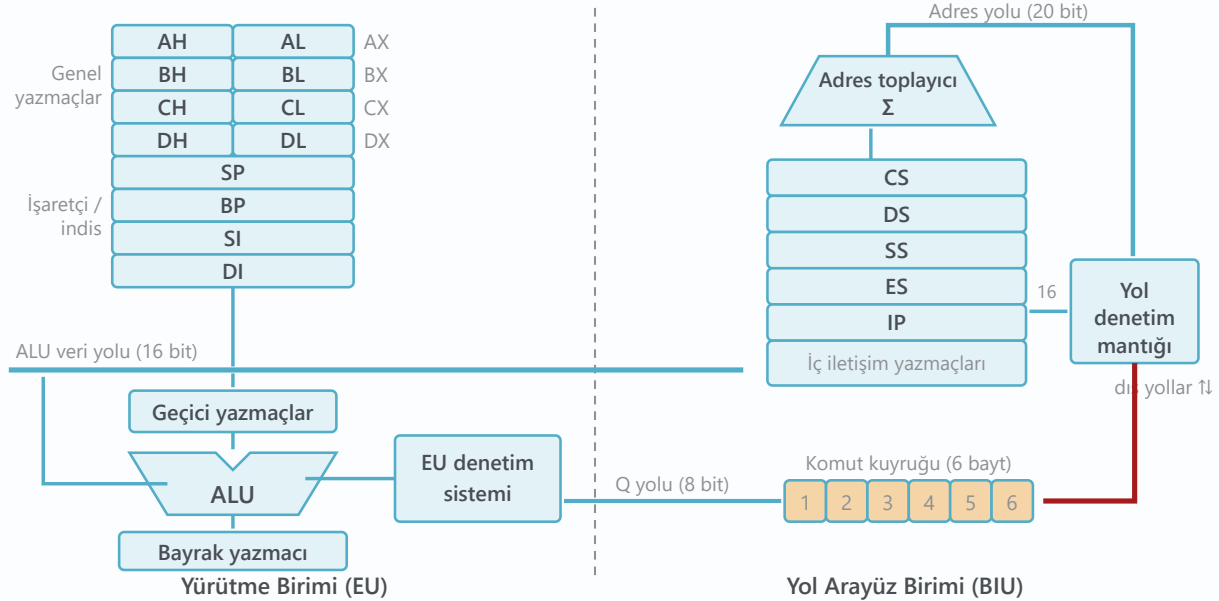
1. BIU, getirilecek komutun fiziksel adresini hesaplar: **$CS \times 10h + IP$** (adres toplayıcı).

8086 Mimarisinin İşleyişi



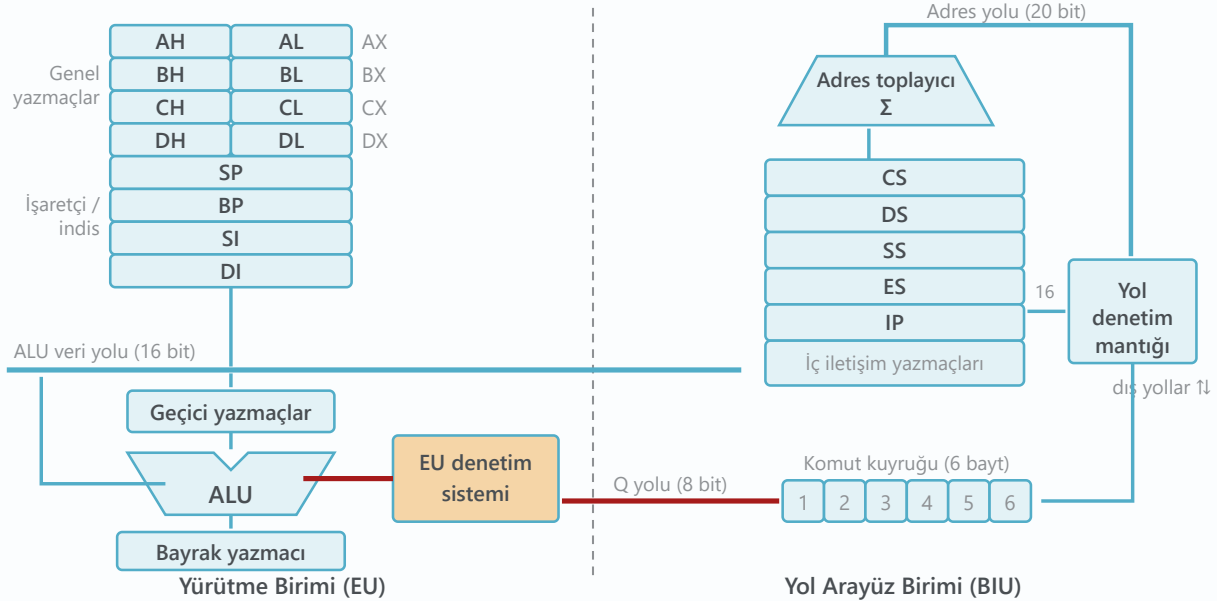
2. 20 bitlik adres dış adres yoluna konur; yol kontrol mantığı bir **bellek okuma döngüsü** başlatır.

8086 Mimarisinin İşleyişi



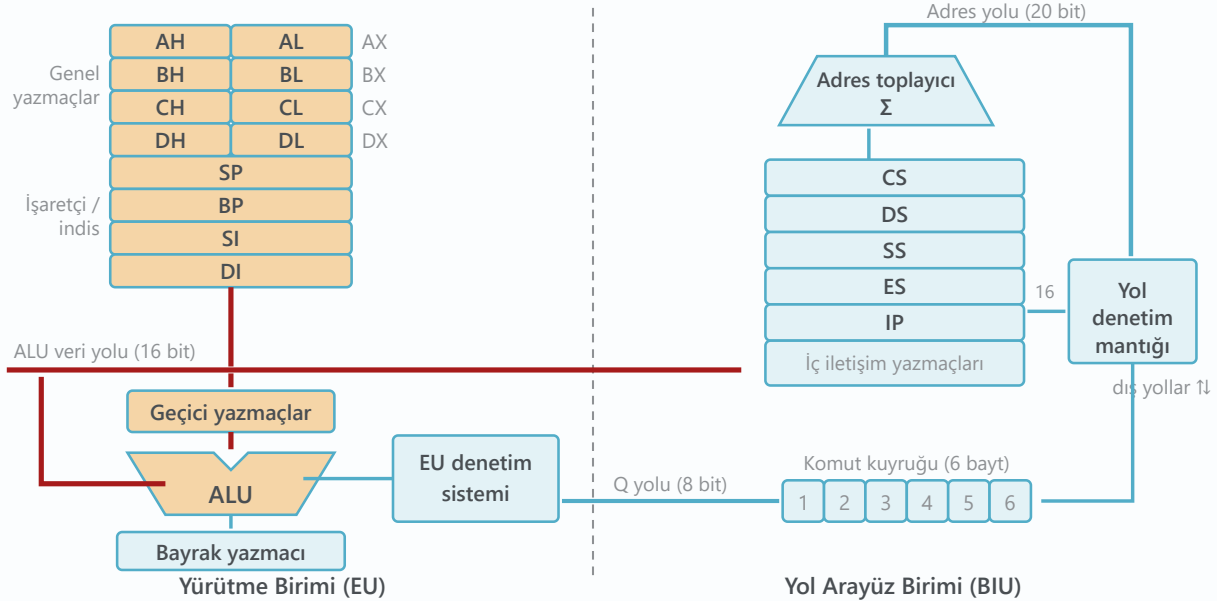
3. Gelen komut baytları 6 baytlık **FIFO kuyruğa** yazılır; IP getirilen bayt sayısı kadar ilerler.

8086 Mimarisinin İşleyişi



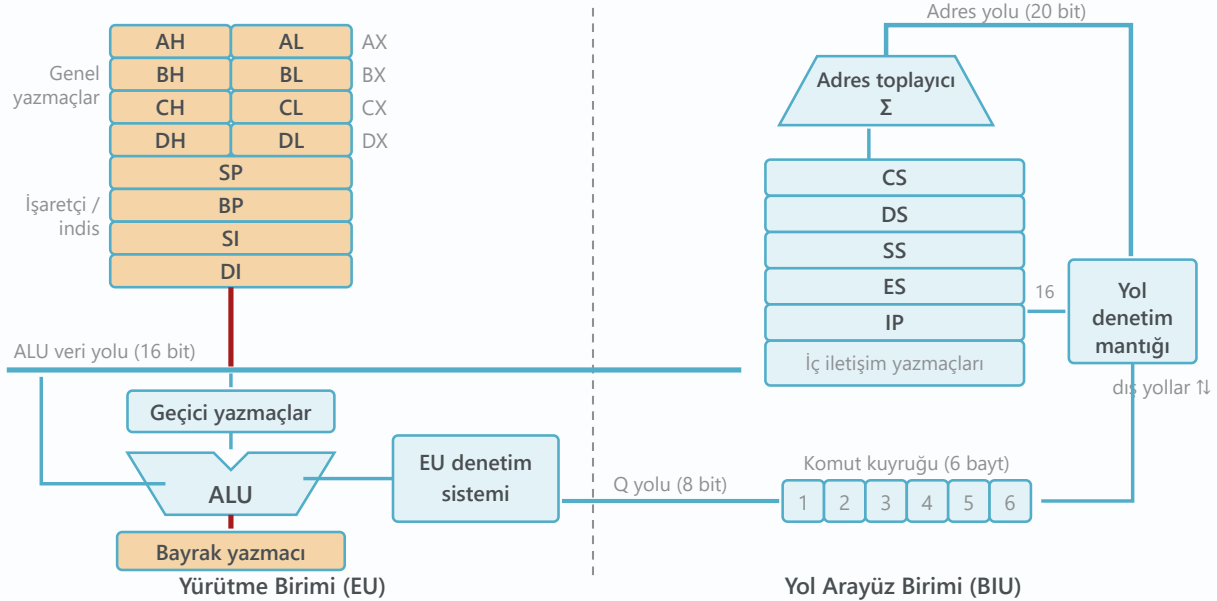
4. Sıra komuta gelince EU, Q yolu üzerinden işlem kodunu (opcode) alır; kontrol sistemi komutu **çözer**.

8086 Mimarisinin İşleyişi



5. Kontrol sinyalleriyle AX ve BX ALU veri yolundan geçici yazmaçlara, oradan **ALU**'ya gelir; toplama yapılır.

8086 Mimarisinin İşleyişi



6. Sonuç AX'e yazılır, **bayraklar** (CF, ZF, SF, OF, PF, AF) güncellenir. Bu sırada BIU sonraki baytları getirmeyi sürdürür.

Yol Arayüz Birimi (BIU)

- Harici bellek ve G/Ç aygıtlarına **sistem yolu** üzerinden arayüz sağlar.
- Kesim yazmacı + bağlı konumdan **20 bitlik fiziksel adres** üretir (adres toplayıcı).
- Bellekten komutları **getirir** ve kuyruğa yazar.
- EU'nun istediği işlenenleri bellekten/G/Ç'den okur, sonuçları yazar.
- 6 baytlık **önden getirme kuyruğunu** (instruction queue) yönetir → boru hattını mümkün kılar.

BIU'nun içindekiler

- Kesim yazmaçları: **CS**, **DS**, **SS**, **ES**
- Komut işaretçisi: **IP**
- Adres toplayıcı (Σ)
- 6 baytlık komut kuyruğu
- Yol kontrol mantığı

Önden Getirme Kuyruğu

- 6 baytlık bir **FIFO** (ilk giren ilk çıkar) kuyruk: komut baytları getirildiği sırayla tüketilir.
- EU bir komutu yürütürken BIU sonraki komut baytlarını önden getirir → **boru hattı**.
- Kuyrukta en az **2 bayt boş yer** açıldığında BIU sıradaki sözcüğü (2 bayt) getirmeye başlar.
- Bir **dallanma** (JMP, CALL, RET, koşullu atlama, kesme) yürütülünce kuyruk **temizlenir**; önden getirilen baytlar artık yanlış yoldadır.
- 8086 komutları 1–6 bayt uzunluğundadır (örnekler hariç); kuyruk en uzun komutu tümüyle tutabilir.



Kazanç

EU çoğu zaman beklemeden kuyruktan komut alır; bellek erişim süresi gizlenir.

Kayıp

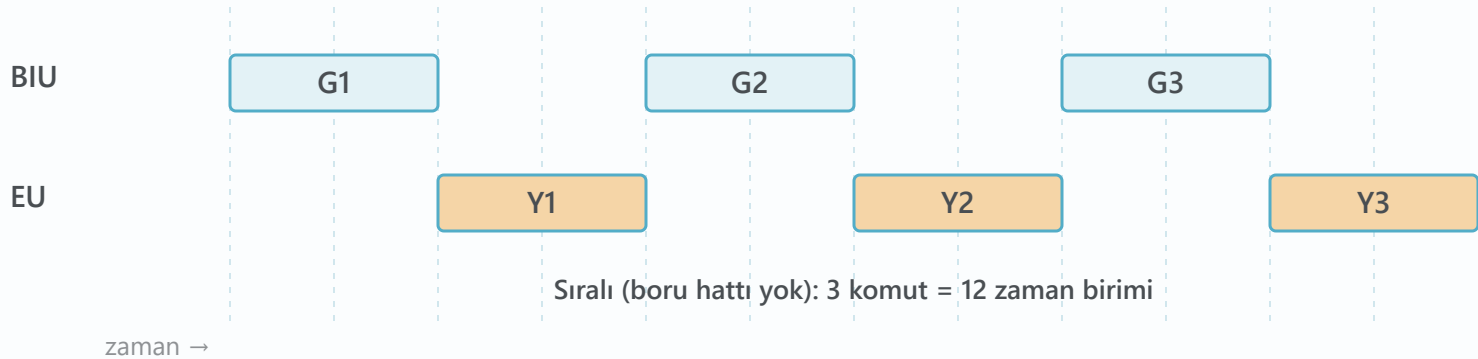
Dallanmada kuyruk boşaltılır, yeniden dolana kadar EU bekler.

8088 farkı

8 bitlik dış yol nedeniyle 4 baytlık kuyruk; 1 bayt boşalınca getirir.

Boru Hattı (Pipelining)

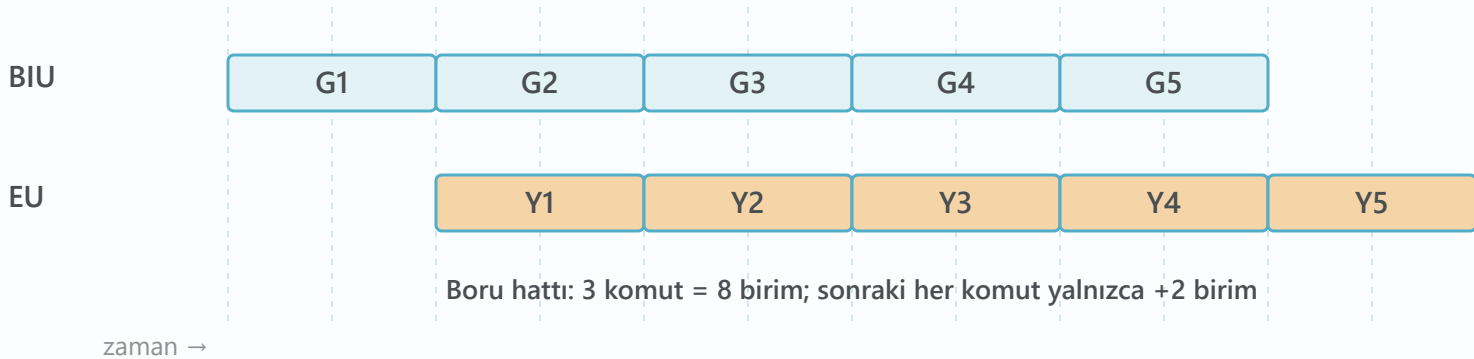
G = getir (fetch, BIU), Y = çöz + yürüt (EU). Her kutu 2 zaman birimi varsayılmıştır.



Boru hattı olmasaydı: önce getir (G), sonra yürüt (Y); BIU ve EU sırayla boş bekler.

Boru Hattı (Pipelining)

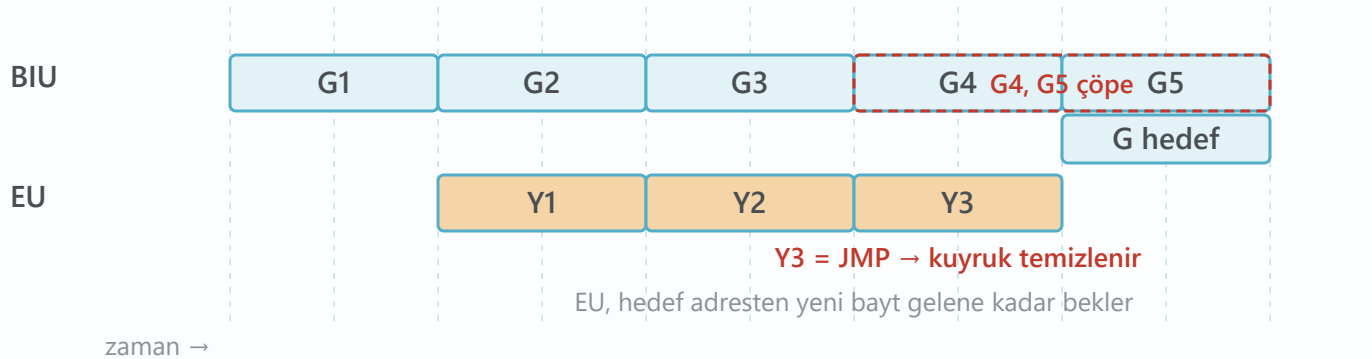
G = getir (fetch, BIU), Y = çöz + yürüt (EU). Her kutu 2 zaman birimi varsayılmıştır.



8086: EU **Y1**'i yürütürken BIU **G2**'yi getirir. İki birim aynı anda çalışır; verim yaklaşık iki katına çıkar.

Boru Hattı (Pipelining)

G = getir (fetch, BIU), Y = çöz + yürüt (EU). Her kutu 2 zaman birimi varsayılmıştır.



Dallanma: Y3 bir **JMP** ise önden getirilen G4, G5 geçersizdir. Kuyruk boşaltılır; hedef adresten yeniden getirilir.

Kuyruk Adım Adım

Bellek (CS:adres → bayt)

B8	34	12	01	D8	41	EB	05	90	90	90	90	90	B0
0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D

Kuyruk (6 bayt, FIFO) — çıkış solda



EU şu an:

– (bekliyor)



IP (sonraki komut) = 0100h

B8 34 12 = MOV AX,1234h · 01 D8 = ADD AX,BX · 41 = INC CX · EB 05 = JMP SHORT 010Dh · 90 = NOP · B0 ... = MOV AL,...

Başlangıç (veya dallanma sonrası): kuyruk boş, EU bekler. CS:IP = CS:0100h.

Kuyruk Adım Adım

Bellek (CS:adres → bayt)

B8	34	12	01	D8	41	EB	05	90	90	90	90	90	B0
0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D

Kuyruk (6 bayt, FIFO) — çıkış solda

B8	34	12	01	D8	41
----	----	----	----	----	----

EU şu an:

MOV AX,1234h çözülüyor

EU

IP (sonraki komut) = 0100h

B8 34 12 = MOV AX,1234h · 01 D8 = ADD AX,BX · 41 = INC CX · EB 05 = JMP SHORT 010Dh · 90 = NOP · B0 ... = MOV AL,...

BIU sözcük sözcük getirir (B8 34, 12 01, D8 41); kuyruk dolar. EU ilk bayt gelir gelmez çözmeye başlar.

Kuyruk Adım Adım

Bellek (CS:adres → bayt)

B8	34	12	01	D8	41	EB	05	90	90	90	90	90	B0
0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D

Kuyruk (6 bayt, FIFO) — çıkış solda

01	D8	41			
----	----	----	--	--	--

EU şu an:

MOV AX,1234h

EU

IP (sonraki komut) = 0103h

B8 34 12 = MOV AX,1234h · 01 D8 = ADD AX,BX · 41 = INC CX · EB 05 = JMP SHORT 010Dh · 90 = NOP · B0 ... = MOV AL,...

EU 3 baytlık **MOV AX,1234h**'yi kuyruktan çekti. Kuyrukta 3 boş yer var (≥ 2).

Kuyruk Adım Adım

Bellek (CS:adres → bayt)

B8	34	12	01	D8	41	EB	05	90	90	90	90	90	B0
0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D

Kuyruk (6 bayt, FIFO) — çıkış solda

01	D8	41	EB	05	
----	----	----	----	----	--

EU şu an:

MOV AX,1234h

EU

IP (sonraki komut) = 0103h

B8 34 12 = MOV AX,1234h · 01 D8 = ADD AX,BX · 41 = INC CX · EB 05 = JMP SHORT 010Dh · 90 = NOP · B0 ... = MOV AL,...

EU yürütürken BIU sonraki sözcüğü (**EB 05**) getirir. Bekleme yok.

Kuyruk Adım Adım

Bellek (CS:adres → bayt)

B8	34	12	01	D8	41	EB	05	90	90	90	90	90	B0
0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D

Kuyruk (6 bayt, FIFO) — çıkış solda

41	EB	05	90	90	
----	----	----	----	----	--

EU şu an:

ADD AX,BX

EU

IP (sonraki komut) = 0105h

B8 34 12 = MOV AX,1234h · 01 D8 = ADD AX,BX · 41 = INC CX · EB 05 = JMP SHORT 010Dh · 90 = NOP · B0 ... = MOV AL,...

EU **01 D8**'i (ADD AX,BX) aldı; BIU **90 90**'ı getirdi.

Adım 5 / 8

Kuyruk Adım Adım

Bellek (CS:adres → bayt)

B8	34	12	01	D8	41	EB	05	90	90	90	90	90	B0
0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D

Kuyruk (6 bayt, FIFO) — çıkış solda

EB	05	90	90	90	90
----	----	----	----	----	----

EU şu an:

INC CX

EU

IP (sonraki komut) = 0106h

B8 34 12 = MOV AX,1234h · 01 D8 = ADD AX,BX · 41 = INC CX · EB 05 = JMP SHORT 010Dh · 90 = NOP · B0 ... = MOV AL,...

EU **41**'i (INC CX) aldı; BIU iki NOP baytı daha getirdi, kuyruk dolu.

Kuyruk Adım Adım

Bellek (CS:adres → bayt)

B8	34	12	01	D8	41	EB	05	90	90	90	90	90	B0
0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D

Kuyruk (6 bayt, FIFO) — çıkış solda



EU şu an:

JMP SHORT 010Dh

EU

IP (sonraki komut) = 010Dh

B8 34 12 = MOV AX,1234h · 01 D8 = ADD AX,BX · 41 = INC CX · EB 05 = JMP SHORT 010Dh · 90 = NOP · B0 ... = MOV AL,...

EU **EB 05**'i yürütür: $IP = 0108h + 5 = \mathbf{010Dh}$. Kuyruktaki 90'lar geçersiz → **kuyruk temizlenir.**

Kuyruk Adım Adım

Bellek (CS:adres → bayt)

B8	34	12	01	D8	41	EB	05	90	90	90	90	90	B0
0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D

Kuyruk (6 bayt, FIFO) — çıkış solda

B0					
----	--	--	--	--	--

EU şu an:

– (bekliyor)

EU

IP (sonraki komut) = 010Dh

B8 34 12 = MOV AX,1234h · 01 D8 = ADD AX,BX · 41 = INC CX · EB 05 = JMP SHORT 010Dh · 90 = NOP · B0 ... = MOV AL,...

BIU 010Dh'den yeniden getirmeye başlar (tek adres olduğundan önce 1 bayt). EU, bayt gelene kadar bekler.

Önden Getirme ve Çözümleme İşlevleri

Önden getirme (prefetch) işlevi

- Komutları bellekten getirip bir **tamponda** (kuyruk) saklar.
- Bir **program sayacı** sonraki getirilecek baytın adresini tutar (8086'da bu iş IP ve BIU içi sayaçla yapılır).
- EU, komutları bellek yerine tampondan alır → bellek erişim süresi gizlenir.
- Birim zamanda daha fazla komut yürütülür.

Çözümleme (decode) işlevi

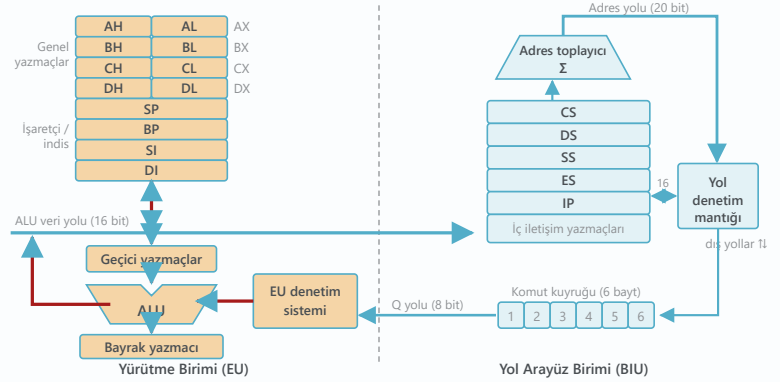
- Makine kodunu, kontrol sinyallerine/mikro işlemlere çevirir.
- Önden getirme ile paralel çalışır.
- Koşullu atlama, CALL, RET gibi karmaşık komutları çözer.
- Yazmaçtan yazmaca, bellekten yazmaca aktarımları yönlendirir.

8086'da bu işlevler ayrı birimler değildir: önden getirme **BIU**'da, çözme **EU**'nun kontrol sisteminde (mikrokod ROM'u ile) yapılır. Ayrı "prefetch unit" ve "decode unit" 80286 ve sonrasında ortaya çıkar.

Yürütme Birimi (EU)

EU'nun başlıca bileşenleri:

- Genel amaçlı yazmaçlar
- Aritmetik-mantık birimi (ALU)
- Özel amaçlı yazmaçlar (SP, BP, SI, DI)
- Komut yazmacı ve komut çözücü
- Bayrak (durum) yazmacı
- Kontrol sistemi



Aritmetik-Mantık Birimi (ALU)

- 8 ve 16 bitlik **aritmetik** ve **mantık** işlemlerini yapar.
- Aritmetik: toplama, çıkarma, çarpma, bölme, artırma/azaltma, karşılaştırma.
- Mantık: AND, OR, XOR, NOT, kaydırma ve döndürme.
- Her işlemden sonra sonuca göre **bayrakları** günceller.

Komut	İşlem	Boyut
ADD AX, BX	$AX \leftarrow AX + BX$	16 bit
SUB AL, 5	$AL \leftarrow AL - 5$	8 bit
MUL BL	$AX \leftarrow AL \times BL$	8 $\rightarrow$ 16
DIV CX	$AX \leftarrow DX:AX \div CX$, $DX \leftarrow \text{kalan}$	32 $\rightarrow$ 16
AND AL, 0Fh	AL'nin üst 4 bitini sıfırla	8 bit
XOR AX, AX	$AX \leftarrow 0$ (ZF = 1)	16 bit

Kod İzleme: ALU Ne Yapar?

1 org 100h

► mov ax, 0010h

► mov bx, 0030h

4 add ax, bx ; AX ← AX + BX

5 sub al, 5 ; AL ← AL - 5

6 mov bl, 4

7 mul bl ; AX ← AL × BL

8 mov dx, 0

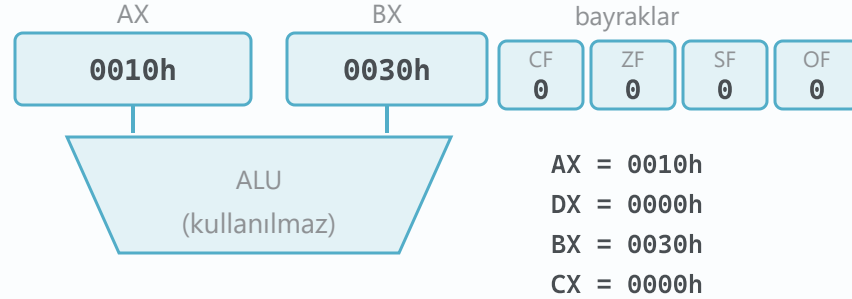
9 mov cx, 3

10 div cx ; AX ← DX:AX ÷ CX

11 and al, 0Fh ; üst 4 biti sil

12 xor ax, ax ; AX ← 0

13 ret



İşlenenler yazmaçlara yüklenir: AX = 0010h, BX = 0030h.
MOV, ALU'yu kullanmaz; bayraklar değişmez.

Adım 1 / 8

DEĞİŞKENLER

AX = 0010h

BX = 0030h

CX = 0000h

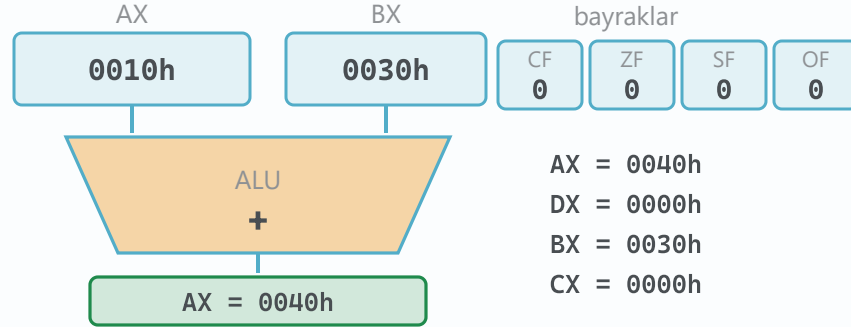
DX = 0000h

CF = 0

ZF = 0

Kod İzleme: ALU Ne Yapar?

```
1 org 100h
2 mov ax, 0010h
3 mov bx, 0030h
4 ► add ax, bx      ; AX ← AX + BX
5 sub al, 5         ; AL ← AL - 5
6 mov bl, 4
7 mul bl           ; AX ← AL × BL
8 mov dx, 0
9 mov cx, 3
10 div cx          ; AX ← DX:AX ÷ CX
11 and al, 0Fh     ; üst 4 biti sil
12 xor ax, ax       ; AX ← 0
13 ret
```



`add ax, bx` : ALU 16 bitlik toplama yapar → AX = **0040h**.
Elde yok (CF = 0), sonuç sıfır değil (ZF = 0).

Adım 2 / 8

DEĞİŞKENLER

AX = 0040h

BX = 0030h

CX = 0000h

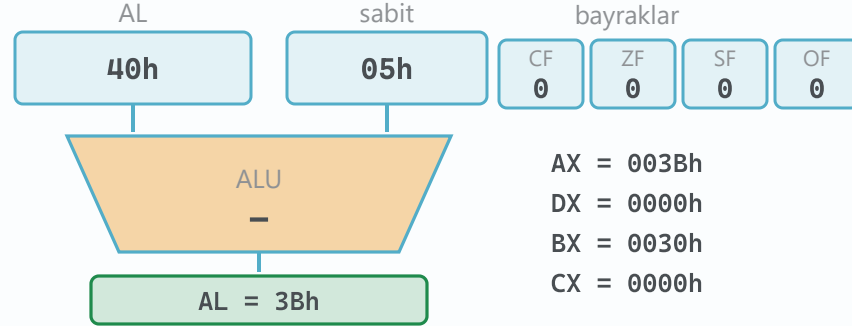
DX = 0000h

CF = 0

ZF = 0

Kod İzleme: ALU Ne Yapar?

```
1 org 100h
2 mov ax, 0010h
3 mov bx, 0030h
4 add ax, bx      ; AX ← AX + BX
5 ► sub al, 5     ; AL ← AL - 5
6 mov bl, 4
7 mul bl          ; AX ← AL × BL
8 mov dx, 0
9 mov cx, 3
10 div cx          ; AX ← DX:AX ÷ CX
11 and al, 0Fh     ; üst 4 biti sil
12 xor ax, ax      ; AX ← 0
13 ret
```



sub al, 5 : 8 bitlik çıkarma; yalnız AL değişir → AL = **3Bh** (AX = 003Bh). Ödünç gerekmedi: CF = 0.

Adım 3 / 8

DEĞİŞKENLER

AX = **003Bh**

BX = 0030h

CX = 0000h

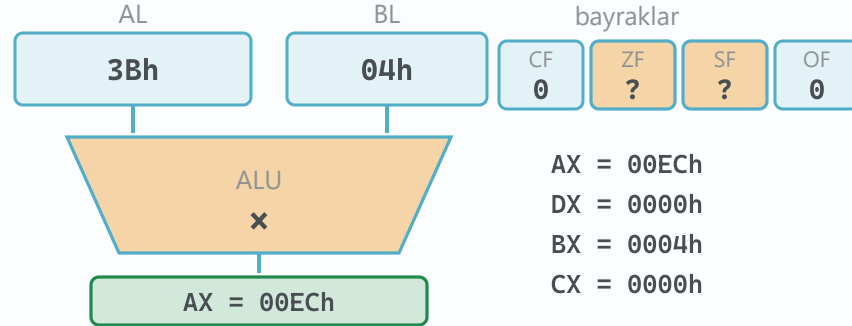
DX = 0000h

CF = 0

ZF = 0

Kod İzleme: ALU Ne Yapar?

```
1 org 100h
2 mov ax, 0010h
3 mov bx, 0030h
4 add ax, bx      ; AX ← AX + BX
5 sub al, 5       ; AL ← AL - 5
▶ mov bl, 4
▶ mul bl          ; AX ← AL × BL
8 mov dx, 0
9 mov cx, 3
10 div cx         ; AX ← DX:AX ÷ CX
11 and al, 0Fh    ; üst 4 biti sil
12 xor ax, ax     ; AX ← 0
13 ret
```



BL = 4. `mul bl` : 8 bitlik işlenenlerin çarpımı 16 bittir: 3Bh × 04h = **00ECh** → AX. AH = 0 olduğundan CF = OF = 0; SF ve ZF tanımsız (?).

Adım 4 / 8

DEĞİŞKENLER

AX = 00ECh

BX = 0004h

CX = 0000h

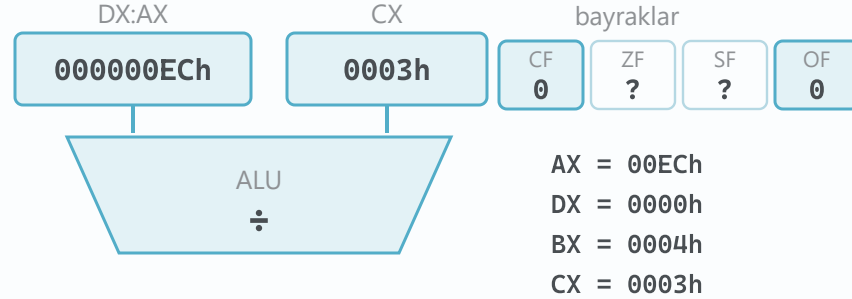
DX = 0000h

CF = 0

ZF = ?

Kod İzleme: ALU Ne Yapar?

```
1 org 100h
2 mov ax, 0010h
3 mov bx, 0030h
4 add ax, bx      ; AX ← AX + BX
5 sub al, 5       ; AL ← AL - 5
6 mov bl, 4
7 mul bl          ; AX ← AL × BL
▶ mov dx, 0
▶ mov cx, 3
10 div cx         ; AX ← DX:AX ÷ CX
11 and al, 0Fh    ; üst 4 biti sil
12 xor ax, ax     ; AX ← 0
13 ret
```



Bölmeye hazırlık: bölünen 32 bitlik **DX:AX** çiftidir, DX = 0 yapılır; bölen CX = 3.

Adım 5 / 8

DEĞİŞKENLER

AX = 00ECh

BX = 0004h

CX = 0003h

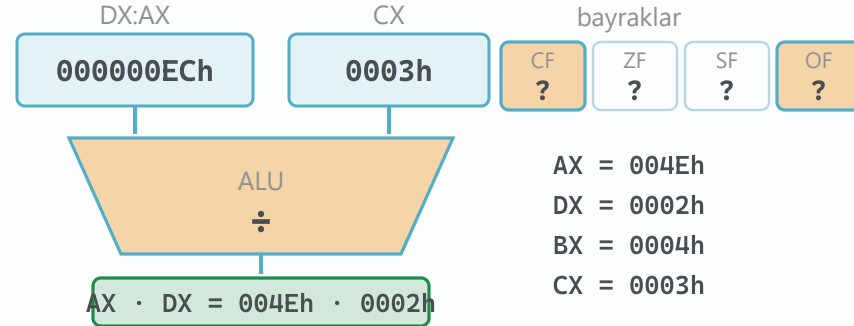
DX = 0000h

CF = 0

ZF = ?

Kod İzleme: ALU Ne Yapar?

```
1 org 100h
2 mov ax, 0010h
3 mov bx, 0030h
4 add ax, bx      ; AX ← AX + BX
5 sub al, 5       ; AL ← AL - 5
6 mov bl, 4
7 mul bl         ; AX ← AL × BL
8 mov dx, 0
9 mov cx, 3
10 ► div cx       ; AX ← DX:AX ÷ CX
11 and al, 0Fh    ; üst 4 biti sil
12 xor ax, ax     ; AX ← 0
13 ret
```



`div cx` : $236 \div 3 = 78$ kalan 2 → bölüm AX = **004Eh**, kalan DX = **0002h**. DIV sonrası bayraklar tanımsızdır.

Adım 6 / 8

DEĞİŞKENLER

AX = 004Eh

BX = 0004h

CX = 0003h

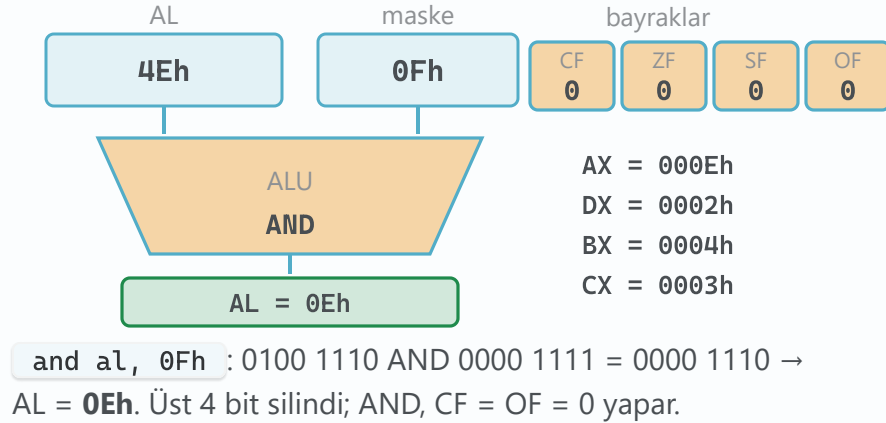
DX = 0002h

CF = ?

ZF = ?

Kod İzleme: ALU Ne Yapar?

```
1 org 100h
2 mov ax, 0010h
3 mov bx, 0030h
4 add ax, bx      ; AX ← AX + BX
5 sub al, 5       ; AL ← AL - 5
6 mov bl, 4
7 mul bl          ; AX ← AL × BL
8 mov dx, 0
9 mov cx, 3
10 div cx          ; AX ← DX:AX ÷ CX
11 ► and al, 0Fh   ; üst 4 biti sil
12 xor ax, ax      ; AX ← 0
13 ret
```



Adım 7 / 8

DEĞİŞKENLER

AX = 000Eh

BX = 0004h

CX = 0003h

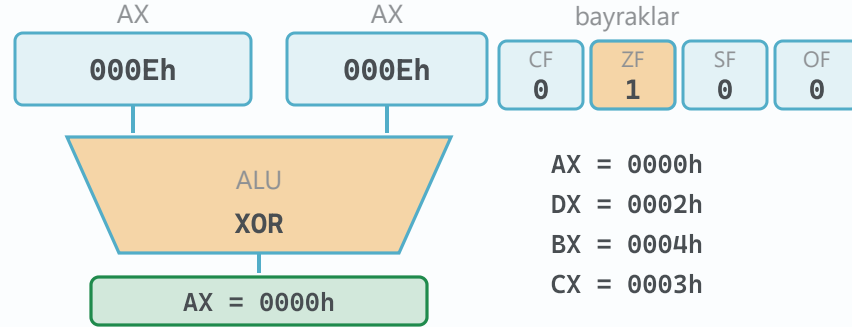
DX = 0002h

CF = 0

ZF = 0

Kod İzleme: ALU Ne Yapar?

```
1 org 100h
2 mov ax, 0010h
3 mov bx, 0030h
4 add ax, bx      ; AX ← AX + BX
5 sub al, 5       ; AL ← AL - 5
6 mov bl, 4
7 mul bl          ; AX ← AL × BL
8 mov dx, 0
9 mov cx, 3
10 div cx          ; AX ← DX:AX ÷ CX
11 and al, 0Fh     ; üst 4 biti sil
12 xor ax, ax      ; AX ← 0
13 ret
```



xor ax, ax : her bit kendisiyle XOR'lanınca 0 olur → AX = 0000h, ZF = 1. **mov ax, 0** 'dan kısa (2 bayt) bir sıfırlama yoludur.

Adım 8 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0004h

CX = 0003h

DX = 0002h

CF = 0

ZF = 1

Komut Yazmacı, Çözücü ve Kontrol

Komut yazmacı

EU, kuyruktan işletilecek komutu alır; işlem kodu (opcode) çözülürken geçici olarak burada tutulur. Programcı bu yazmaca erişemez.

Komut çözücü

İşlem kodunu çözer: hangi işlem, hangi işlenenler, hangi adresleme modu? Gerekli bilgiyi kontrol devresine gönderir.

Kontrol birimi

Komut ve verinin işlemci içindeki akışını yönetir; getir-çöz-yürüt döngüsünü sıralar, diğer bileşenleri koordine eder.

Kontrol biriminin görevleri: komutları **getirmek** (BIU ile), **çözmek**, **yürütmek**, işlemcinin durumunu (yazmaçlar, bayraklar) güncellemek ve her komut sonunda **kesme isteklerini** (INTR, NMI) denetlemek. Hata durumları (ör. sıfıra bölme) özel kesmelerle ele alınır.

Güç yönetimi 8086'da yoktur; bu özellikler çok sonraki işlemcilerle (ör. 386SL) gelmiştir. 8086'nın kontrol birimi mikrokod tabanlıdır.

İÇ YAPI

Komut Uzunlukları

8086 komutları **değişken uzunluktadır**: işlem kodu + (varsa) adresleme baytı + yer değiştirme + anlık veri.

Komut	Makine kodu (hex)	Bayt	Açıklama
INC CX	41	1	yalnızca işlem kodu
MOV AX, BX	8B C3	2	işlem kodu + ModR/M baytı
ADD AL, 7Ch	04 7C	2	AL'ye özel kısa kod + 8 bit anlık
MOV BX, 4050h	BB 50 40	3	işlem kodu + 16 bit anlık (little-endian)
MOV WORD PTR [BX+1234h], 5678h	C7 87 34 12 78 56	6	ModR/M + 16 bit yer değiştirme + 16 bit anlık

Anlık değerler belleğe **little-endian** düzende (düşük bayt önce) yazılır: 4050h → önce 50 , sonra 40 .

Yazmaçtan yazmaca MOV 2 bayttır (8B C3 ya da eşdeğeri 89 D8).

Kod İzleme: Baytlar ve IP

```
1 org 100h
```

```
► inc cx
```

```
3 mov bx, 4050h
```

```
4 mov ax, bx
```

```
5 add al, 7Ch
```

```
6 mov word ptr [bx+1234h], 5678h
```

```
7 ret
```

kod kesimi (CS:0100h–010Eh)



INC CX

1 bayt · işlem kodu

IP: 0100h → 0101h

Program 0100h'de. `inc cx` yalnız işlem kodundan oluşur:
41 (1 bayt) → IP = **0101h**. CX = 1.

Adım 1 / 6

DEĞİŞKENLER

IP = 0101h

CX = 0001h

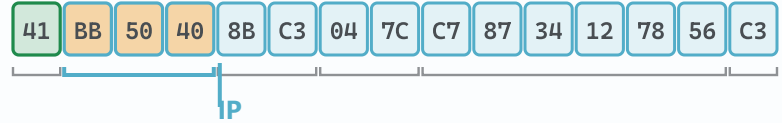
BX = 0000h

AX = 0000h

Kod İzleme: Baytlar ve IP

```
1 org 100h
2 inc cx
3 ► mov bx, 4050h
4 mov ax, bx
5 add al, 7Ch
6 mov word ptr [bx+1234h], 5678h
7 ret
```

kod kesimi (CS:0100h–010Eh)



MOV BX,4050h

3 bayt · işlem kodu + 16 bit anlık

IP: 0101h → 0104h

BB 50 40: BB = “BX'e 16 bitlik sabit”; sabit little-endian yazılır (önce 50, sonra 40). 3 bayt → IP = **0104h**.

Adım 2 / 6

DEĞİŞKENLER

IP = 0104h

CX = 0001h

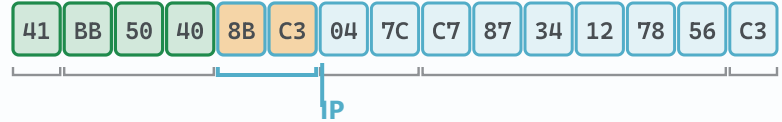
BX = 4050h

AX = 0000h

Kod İzleme: Baytlar ve IP

```
1 org 100h
2 inc cx
3 mov bx, 4050h
4 ► mov ax, bx
5 add al, 7Ch
6 mov word ptr [bx+1234h], 5678h
7 ret
```

kod kesimi (CS:0100h–010Eh)



MOV AX, BX

2 bayt · işlem kodu + ModR/M

IP: 0104h → 0106h

8B C3: 8B = MOV r16, r/m16; ModR/M C3 = 11 000 011 → mod 11 (yazmaç), reg AX, r/m BX. 2 bayt → IP = **0106h**.

Adım 3 / 6

DEĞİŞKENLER

IP = 0106h

CX = 0001h

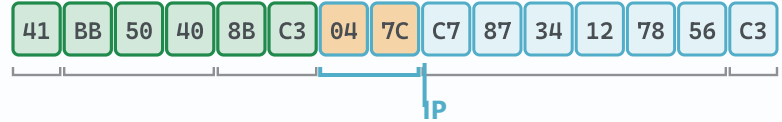
BX = 4050h

AX = 4050h

Kod İzleme: Baytlar ve IP

```
1 org 100h
2 inc cx
3 mov bx, 4050h
4 mov ax, bx
5 ► add al, 7Ch
6 mov word ptr [bx+1234h], 5678h
7 ret
```

kod kesimi (CS:0100h–010Eh)



ADD AL, 7Ch

2 bayt · AL kısa kodu + 8 bit anlık

IP: 0106h → 0108h

04 7C: AL'ye özel kısa kod + 8 bitlik sabit. $AL = 50h + 7Ch = CCh$, $AX = 40CCh$. IP = **0108h**.

Adım 4 / 6

DEĞİŞKENLER

IP = 0108h

CX = 0001h

BX = 4050h

AX = 40CCh

Kod İzleme: Baytlar ve IP

```

1 org 100h
2 inc cx
3 mov bx, 4050h
4 mov ax, bx
5 add al, 7Ch
▶ mov word ptr [bx+1234h], 5678h
7 ret

```

kod kesimi (CS:0100h–010Eh)



MOV WORD PTR [BX+1234h], 5678h

6 bayt · kod + ModR/M + yer değ. + anlık

DS:5284h

78 56

veri: 5678h

IP: 0108h → 010Eh

6 bayt: C7 + ModR/M 87 (mod 10: 16 bit yer değiştirme, r/m BX) + **34 12** + **78 56**. EA = 4050h + 1234h = 5284h; oraya 5678h yazılır. IP = **010Eh**.

Adım 5 / 6

DEĞİŞKENLER

IP = **010Eh**

CX = **0001h**

BX = **4050h**

AX = **40CCh**

[5284h] = **5678h**

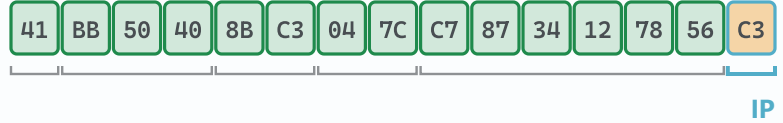
Kod İzleme: Baytlar ve IP

```

1 org 100h
2 inc cx
3 mov bx, 4050h
4 mov ax, bx
5 add al, 7Ch
6 mov word ptr [bx+1234h], 5678h
▶ ret

```

kod kesimi (CS:0100h–010Eh)



RET

1 bayt · işlem kodu

DS:5284h

78 56

veri: 5678h

IP: 010Eh → 0000h

C3 = RET (1 bayt); IP yığından alınan 0000h olur (COM programı DOS'a döner). Altı komut 15 bayt tuttu: uzunluk 1 ile 6 bayt arasında.

Adım 6 / 6

DEĞİŞKENLER

IP = 0000h

CX = 0001h

BX = 4050h

AX = 40CCh

[5284h] = 5678h

YAZMAÇLAR

Yazmaç Kümesi

Genel amaçlı (veri) yazmaçları

AX	AH	AL	akümülatör
BX	BH	BL	taban
CX	CH	CL	sayaç
DX	DH	DL	veri

15 8 7 0

İşaretçi ve indis

SP	yığın işaretçisi
BP	taban işaretçisi
SI	kaynak indis
DI	hedef indis

Özel

IP	komut işaretçisi
----	------------------

Kesim (segment)

CS	kod
DS	veri
SS	yığın
ES	ek

FLAGS	bayraklar
-------	-----------

$$4 + 4 + 4 + 2 = 14 \text{ adet 16 bitlik yazmaç}$$

- **Genel amaçlı:** veri saklar, aritmetik/mantık işlemlerine katılır; her biri 8 bitlik iki yarıya bölünebilir.
- **Kesim yazmaçları:** bellekteki dört etkin kesimin başlangıcını gösterir. **Özel:** IP sonraki komutu, FLAGS son işlemin sonucunu gösterir.

Dahili Yazmaçlar ve Sözcük Boyu

- 8086, 16 bitlik bir **tamsayı** (skaler, vektörel olmayan) işlemcidir; kayan noktalı işlemler için ayrı bir yardımcı işlemci (8087) kullanılır.
- 40 pinli, **çift sıralı paket** (DIP, dual in-line package) olarak üretilmiştir.
- Yonga içindeki yazmaçların boyutu, işlemcinin **aynı anda** üzerinde çalışabileceği veri miktarını (sözcük boyu) gösterir → 16 bit.
- Programcıya **14 adet 16 bitlik** yazmaç sunar.
- Yazmaçlar, işlemcinin içindeki en hızlı depolama alanıdır: bellek erişimi için yol döngüsü (≥ 4 saat) gerekir, yazmaç erişimi gerekmez.

8 bit

0 ... 255 (FFh)

16 bit

0 ... 65 535 (FFFFh)

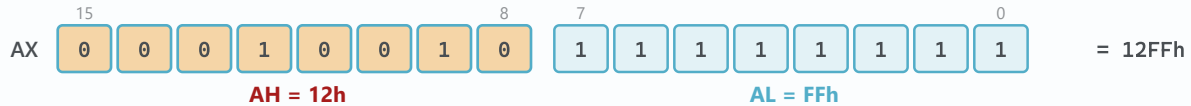
32 bit (DX:AX)

0 ... 4 294 967 295

YAZMAÇLAR

Genel Amaçlı Yazmaçlar

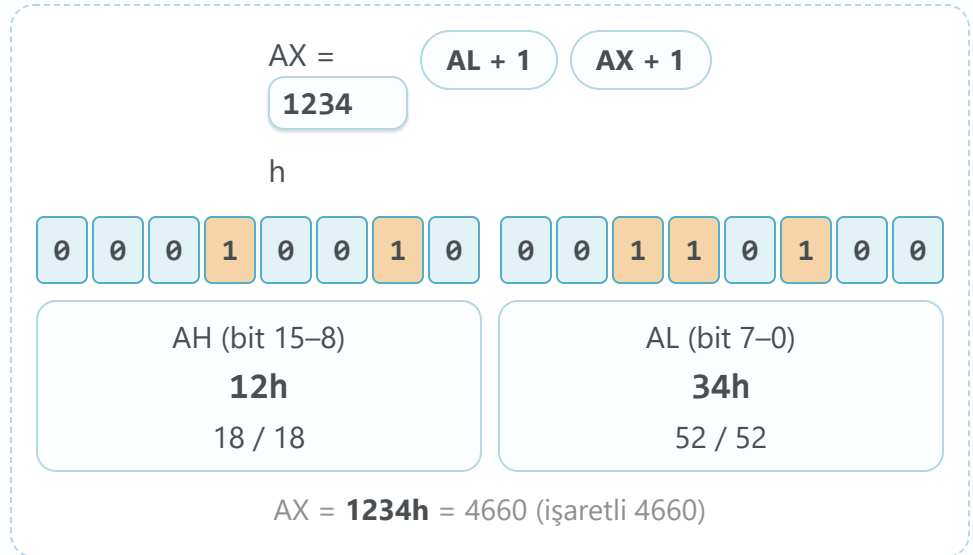
Yazmaç	Yarılar	Özel kullanımı
AX akümülatör (accumulator)	AH · AL	Aritmetik işlenen ve sonuç; MUL/DIV, IN/OUT, dizgi komutları AX/AL'yi kullanır. Birçok komutun AX için daha kısa kodu vardır.
BX taban (base)	BH · BL	Dolaylı adreslemede bağıl konumu (offset) tutar: <code>MOV AL, [BX]</code> .
CX sayaç (count)	CH · CL	LOOP ve REP için döngü sayacı; kaydırma/döndürmede sayı CL 'de.
DX veri (data)	DH · DL	AX ile birlikte 32 bitlik değer (<code>DX:AX</code>) – MUL sonucu, DIV böleni; IN/OUT'ta port adresi.



Yarılar bağımsız 8 bitlik yazmaçlardır ama aynı bitleri paylaşır: AL 'yi değiştirmek AX 'i de değiştirir. AH = bit 15–8 (high), AL = bit 7–0 (low).

Demo: AX = AH : AL

- 16 bitlik bir değer gir (hex).
Yüksek bayt **AH**, düşük bayt **AL** olur.
- Bitlere tıklayarak tek tek değiştir.
- **AL** 'ye 1 ekle düğmesi yalnız düşük baytı değiştirir:
AL = FFh iken taşan elde AH'ye **geçmez** (8 bitlik işlem).



YAZMAÇLAR

İşaretçi ve İndis Yazmaçları

Belirli bellek konumlarını gösteren **bağlı konum** (offset) yazmaçlarıdır; bir kesim yazmacıyla birlikte adres oluştururlar.

Yazmaç	Görev	Varsayılan kesim
SP yığın işaretçisi	Yığının üstünü (top) gösterir; PUSH, POP, CALL, RET, INT, IRET otomatik günceller.	SS
BP taban işaretçisi	Yığının içindeki rastgele konumlara erişir (ör. prosedür parametreleri <code>[BP+4]</code>).	SS
SI kaynak indis	Dizi/dizgi işlemlerinde kaynak konumu.	DS
DI hedef indis	Dizi/dizgi işlemlerinde hedef konumu; dizgi komutlarında (MOVSB, STOSB) <code>ES:DI</code> .	DS (dizgide ES)

PUSH, SP'yi 2 azaltıp SS:SP 'ye yazar; POP okuyup SP'yi 2 artırır. Yığın bellekte **aşağı doğru** büyür.

Kesim Yazmaçları

- Kesimler **bellekte**, kesim yazmaçları **işlemcinin içinde** bulunur.
- Her kesim yazmacı bir kesimin **başlangıç adresini / 16** tutar (kesim tabanı = yazmaç $\times$ 10h).

CS Code Segment	Programın kodu; CS:IP ile sonraki komut
DS Data Segment	Değişkenler ve veriler
SS Stack Segment	Yığın; SS:SP , SS:BP
ES Extra Segment	Ek veri; dizgi hedefi ES:DI

Kurallar

- Kesim yazmacına anlık değer doğrudan yüklenemez: **MOV DS, 2000h** **geçersiz**; önce **MOV AX, 2000h** , sonra **MOV DS, AX** .
- **CS** , MOV ile değiştirilemez; JMP FAR, CALL FAR, RET FAR, INT, IRET değiştirir.
- Kesimden kesime doğrudan MOV yoktur (**MOV ES, DS** geçersiz).

Komut İşaretçisi (IP)

- 16 bitlik yazmaç (Instruction Pointer).
- Kod kesiminde **bir sonraki komutun** bağlı konumunu tutar.
- Her komut baytı getirildikçe ilerler; komut uzunluğu kadar artmış olur.
- Dallanma komutları (JMP, Jcc, CALL, RET, LOOP, INT) IP'ye **yeni bir değer** yükler.
- Programcı IP'yi doğrudan okuyamaz/yazamaz (`MOV IP, ...` yoktur).
- Sonraki komutun fiziksel adresi: **$CS \times 10h + IP$** .

Örnek

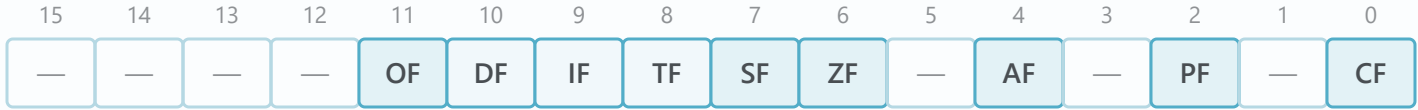
CS = `4321h` , IP = `1000h`

```
43210h    ; CS × 10h
+ 01000h   ; IP
-----
44210h
```

Sonraki komut **44210h** fiziksel adresindedir.

YAZMAÇLAR

Bayrak Yazmacı (FLAGS)



☐ durum bayrağı (6)

☐ kontrol bayrağı (3)

☐ kullanılmaz (7 bit)

Durum bayrakları

Her aritmetik/mantık işleminden sonra **sonuca göre** otomatik güncellenir: CF, PF, AF, ZF, SF, OF. Koşullu atlamalar (JZ, JC, JS ...) bunlara bakar.

Kontrol bayrakları

İşlemcinin **davranışını** yönlendirmek için programcı tarafından ayarlanır: TF, IF, DF.

16 bitlik yazmacın yalnızca 9 biti kullanılır. MOV komutları bayrakları değiştirmez.

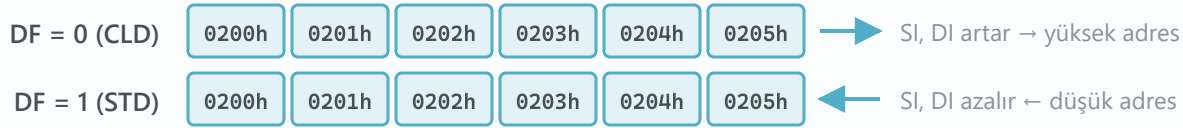
Durum Bayrakları

Bayrak	Bit	1 olur, eğer...	Örnek (8 bit)
CF elde (carry)	0	işaretsiz işlemde en üst bitten elde/ödünç çıktıysa	$\text{FFh} + \text{01h} = \text{00h} \rightarrow \text{CF}=1$
PF eşlik (parity)	2	sonucun düşük baytındaki 1 sayısı çiftse	03h (2 adet 1) $\rightarrow \text{PF}=1$
AF yardımcı (auxiliary)	4	bit 3'ten bit 4'e elde/ödünç olduysa (BCD için)	$\text{0Fh} + \text{01h} \rightarrow \text{AF}=1$
ZF sıfır (zero)	6	sonuç sıfırsa	$\text{05h} - \text{05h} \rightarrow \text{ZF}=1$
SF işaret (sign)	7	sonucun en üst biti 1 ise (negatif)	$\text{80h} \rightarrow \text{SF}=1$
OF taşma (overflow)	11	işaretili sonuç sığmadıysa	$\text{7Fh} + \text{01h} = \text{80h} \rightarrow \text{OF}=1$

CF işaretsiz, **OF işaretili** sayılar için taşmayı bildirir. $\text{7Fh} + \text{01h}$: işaretsiz $127+1=128$ doğru ($\text{CF}=0$), işaretili $+127+1 = -128$ yanlış ($\text{OF}=1$).

Kontrol Bayrakları

Bayrak	Görev	Ayarlama
TF tuzak (trap)	1 iken her komuttan sonra tip 1 kesme → adım adım yürütme (hata ayıklayıcılar).	Doğrudan komutu yok; PUSHF/POPF ile yığın üzerinden
IF kesme (interrupt)	1 iken INTR pininden gelen maskelenebilir kesmeler kabul edilir. NMI'yi etkilemez.	STI (1), CLI (0)
DF yön (direction)	Dizgi komutlarında SI/DI'nin yönü: 0 → artar, 1 → azalır.	CLD (0), STD (1)



CLC , STC , CMC komutları da vardır, ancak bunlar bir kontrol bayrağını değil **CF durum bayrağını** temizler/kurar/tersler (ör. çok baytlı toplama öncesi).

Kod İzleme: Bayraklar ve IP

COM programı **CS:0100h**'den başlar. Varsayım: yazmaçlar ve bayraklar 0.

Adım 1 / 9

► org 100h

```
2 mov ax, 1234h    ; B8 34 12
3 mov bx, 4050h    ; BB 50 40
4 add ax, bx        ; 01 D8
5 mov cx, 0FFFFh   ; B9 FF FF
6 inc cx           ; 41
7 add al, 7Ch       ; 04 7C
8 sub ah, 53h       ; 80 EC 53
9 ret              ; C3
```

DEĞİŞKENLER

IP = **0100h**

AX = **0000h**

BX = **0000h**

CX = **0000h**

CF = **0**

ZF = **0**

SF = **0**

OF = **0**

PF = **0**

AF = **0**

Kod İzleme: Bayraklar ve IP

AX ← 1234h (AH=12h, AL=34h). Komut 3 bayt → IP 3 artar. MOV bayrakları **değiştirmez**.

Adım 2 / 9

```
1 org 100h
```

```
▶ mov ax, 1234h ; B8 34 12
```

```
3 mov bx, 4050h ; BB 50 40
```

```
4 add ax, bx ; 01 D8
```

```
5 mov cx, 0FFFFh ; B9 FF FF
```

```
6 inc cx ; 41
```

```
7 add al, 7Ch ; 04 7C
```

```
8 sub ah, 53h ; 80 EC 53
```

```
9 ret ; C3
```

DEĞİŞKENLER

IP = 0103h

AX = 1234h

BX = 0000h

CX = 0000h

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 0

AF = 0

Kod İzleme: Bayraklar ve IP

BX ← 4050h. Bellekte BB 50 40 : anlık değer little-endian.

Adım 3 / 9

```
1 org 100h
2 mov ax, 1234h ; B8 34 12
▶ 3 mov bx, 4050h ; BB 50 40
4 add ax, bx ; 01 D8
5 mov cx, 0FFFFh ; B9 FF FF
6 inc cx ; 41
7 add al, 7Ch ; 04 7C
8 sub ah, 53h ; 80 EC 53
9 ret ; C3
```

DEĞİŞKENLER

IP = 0106h

AX = 1234h

BX = 4050h

CX = 0000h

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 0

AF = 0

Kod İzleme: Bayraklar ve IP

1234h + 4050h = **5284h**. Elde yok (CF=0), bit 15 = 0 (SF=0). Düşük bayt 84h = 1000 0100b → iki adet 1 → **PF=1**.

Adım 4 / 9

```
1 org 100h
2 mov ax, 1234h      ; B8 34 12
3 mov bx, 4050h      ; BB 50 40
4 ► add ax, bx        ; 01 D8
5 mov cx, 0FFFFh     ; B9 FF FF
6 inc cx              ; 41
7 add al, 7Ch         ; 04 7C
8 sub ah, 53h         ; 80 EC 53
9 ret                 ; C3
```

DEĞİŞKENLER

IP = 0108h

AX = 5284h

BX = 4050h

CX = 0000h

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 1

AF = 0

Kod İzleme: Bayraklar ve IP

CX ← FFFFh. Bayraklar aynı kalır.

Adım 5 / 9

```
1 org 100h
2 mov ax, 1234h      ; B8 34 12
3 mov bx, 4050h      ; BB 50 40
4 add ax, bx         ; 01 D8
5 ▶ mov cx, 0FFFFh   ; B9 FF FF
6 inc cx             ; 41
7 add al, 7Ch         ; 04 7C
8 sub ah, 53h         ; 80 EC 53
9 ret                ; C3
```

DEĞİŞKENLER

IP = 010Bh

AX = 5284h

BX = 4050h

CX = FFFFh

CF = 0

ZF = 0

SF = 0

OF = 0

PF = 1

AF = 0

Kod İzleme: Bayraklar ve IP

$FFFFh + 1 = 0000h \rightarrow ZF=1$, $AF=1$. Dikkat: **INC**, **CF**'yi **değiştirmez** ($CF=0$ kaldı).

Adım 6 / 9

```
1 org 100h
2 mov ax, 1234h      ; B8 34 12
3 mov bx, 4050h      ; BB 50 40
4 add ax, bx          ; 01 D8
5 mov cx, 0FFFFh     ; B9 FF FF
▶ inc cx              ; 41
7 add al, 7Ch         ; 04 7C
8 sub ah, 53h         ; 80 EC 53
9 ret                 ; C3
```

DEĞİŞKENLER

IP = 010Ch

AX = 5284h

BX = 4050h

CX = 0000h

CF = 0

ZF = 1

SF = 0

OF = 0

PF = 1

AF = 1

Kod İzleme: Bayraklar ve IP

AL: $84h + 7Ch = 100h \rightarrow AL=00h$, **CF=1**, ZF=1. 8 bitlik işlem: elde AH'ye geçmez, AX=5200h. İşaretli:
 $-124 + 124 = 0 \rightarrow OF=0$.

Adım 7 / 9

```
1 org 100h
2 mov ax, 1234h      ; B8 34 12
3 mov bx, 4050h      ; BB 50 40
4 add ax, bx         ; 01 D8
5 mov cx, 0FFFFh     ; B9 FF FF
6 inc cx             ; 41
7 ► add al, 7Ch       ; 04 7C
8 sub ah, 53h        ; 80 EC 53
9 ret                ; C3
```

DEĞİŞKENLER

IP = 010Eh

AX = 5200h

BX = 4050h

CX = 0000h

CF = 1

ZF = 1

SF = 0

OF = 0

PF = 1

AF = 1

Kod İzleme: Bayraklar ve IP

AH: $52h - 53h = FFh \rightarrow$ ödünç alındı (**CF=1**), SF=1. İşaretili: $82 - 83 = -1$ doğru $\rightarrow$ OF=0. FFh'de sekiz adet 1 $\rightarrow$ PF=1.

Adım 8 / 9

```
1 org 100h
2 mov ax, 1234h      ; B8 34 12
3 mov bx, 4050h      ; BB 50 40
4 add ax, bx         ; 01 D8
5 mov cx, 0FFFFh     ; B9 FF FF
6 inc cx             ; 41
7 add al, 7Ch         ; 04 7C
8 ► sub ah, 53h       ; 80 EC 53
9 ret                ; C3
```

DEĞİŞKENLER

IP = 0111h

AX = FF00h

BX = 4050h

CX = 0000h

CF = 1

ZF = 0

SF = 1

OF = 0

PF = 1

AF = 1

Kod İzleme: Bayraklar ve IP

RET: COM programı DOS'a döner. IP'nin **komut uzunluklarına göre** 0100h → 0111h ilerlediğine dikkat.

Adım 9 / 9

```
1 org 100h
2 mov ax, 1234h      ; B8 34 12
3 mov bx, 4050h      ; BB 50 40
4 add ax, bx         ; 01 D8
5 mov cx, 0FFFFh     ; B9 FF FF
6 inc cx             ; 41
7 add al, 7Ch         ; 04 7C
8 sub ah, 53h        ; 80 EC 53
▶ ret                ; C3
```

DEĞİŞKENLER

IP = **0111h**

AX = **FF00h**

BX = **4050h**

CX = **0000h**

CF = **1**

ZF = **0**

SF = **1**

OF = **0**

PF = **1**

AF = **1**

Kesimli Bellek Mimarisi

- 8086 belleği **kesimli** (segmented) olarak adresler.
- Her bellek adresi iki 16 bitlik parçayla verilir:
kesim : bağıl konum (segment : offset), ör.
`2000h:0010h`.
- Kesim yazarı** kesimin başlangıcını gösterir.
- Bağıl konum**, istenen baytın kesimin başından itibaren uzaklığıdır (0000h–FFFFh).
- İkisi BIU'daki adres toplayıcıda birleştirilerek **20 bitlik fiziksel adres** elde edilir.

Mantıksal ve fiziksel adres

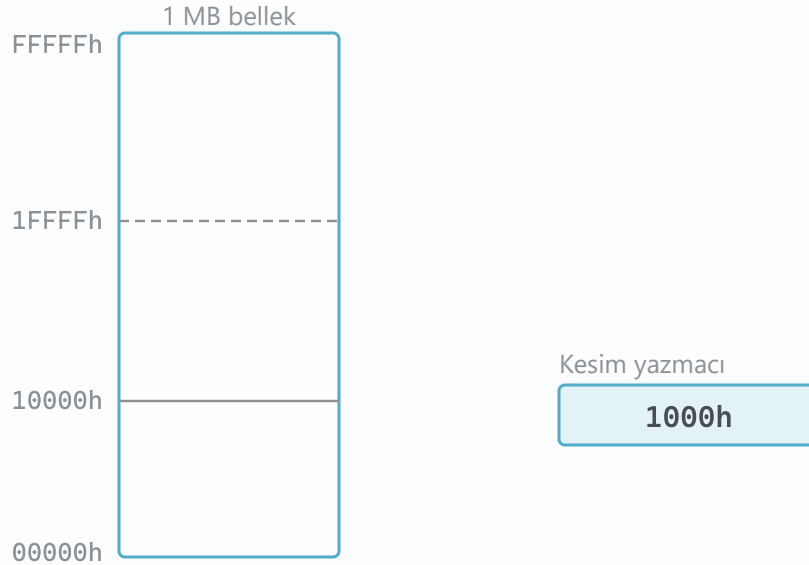
Mantıksal adres: programın kullandığı
`kesim:offset` çifti.

Fiziksel adres: adres yoluna çıkan gerçek 20 bitlik adres.

$$FA = \text{kesim} \times 10h + \text{bağıl konum}$$

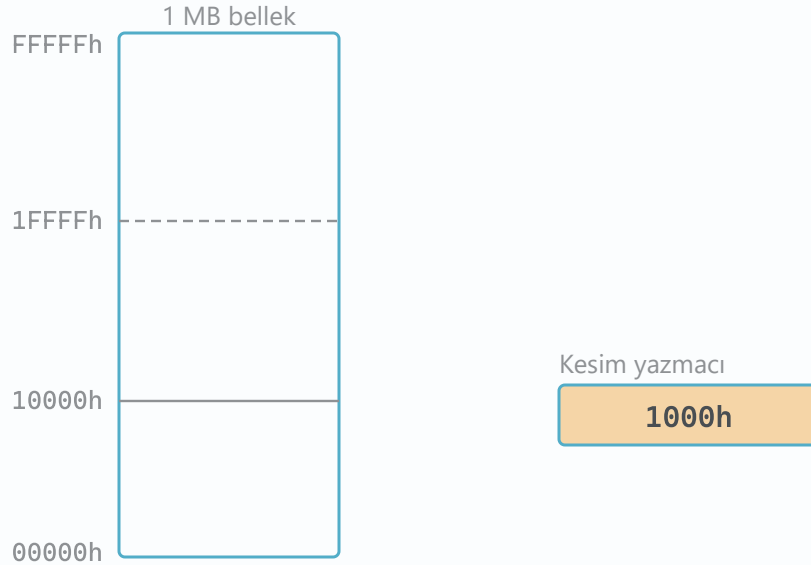


Kesim ve Bağlı Konum



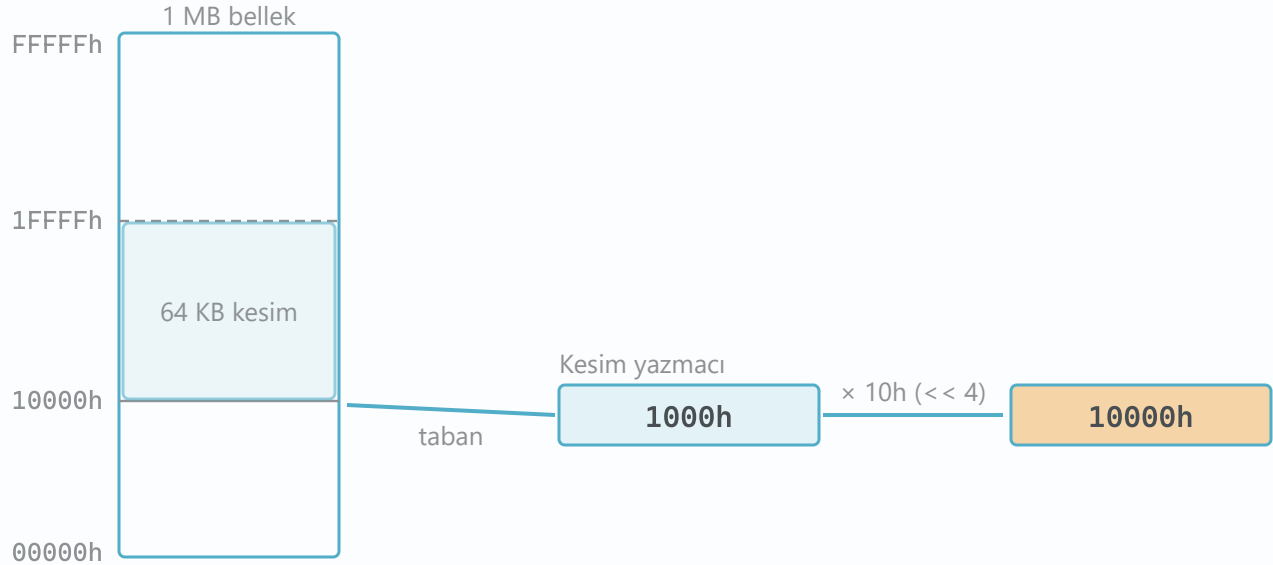
Bellek 1 MB: 00000h–FFFFFh. Bir kesim yazmacı ve bir bağlı konumdan 20 bitlik adres üreteceğiz.

Kesim ve Bağlı Konum



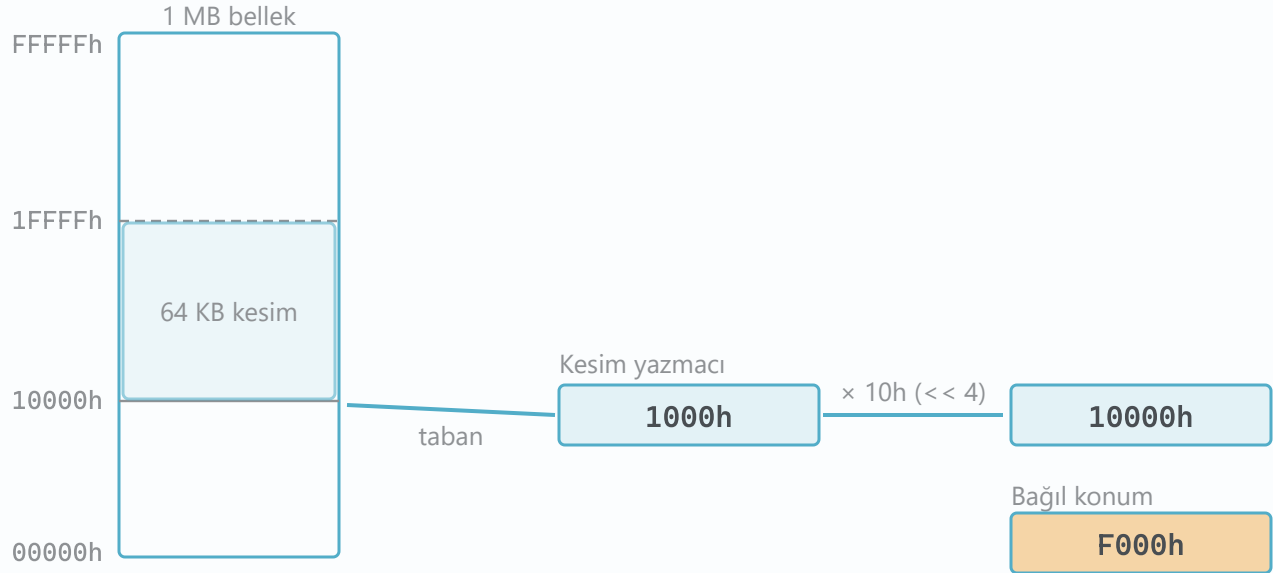
Kesim yazmacı **1000h** (16 bit).

Kesim ve Bağlı Konum



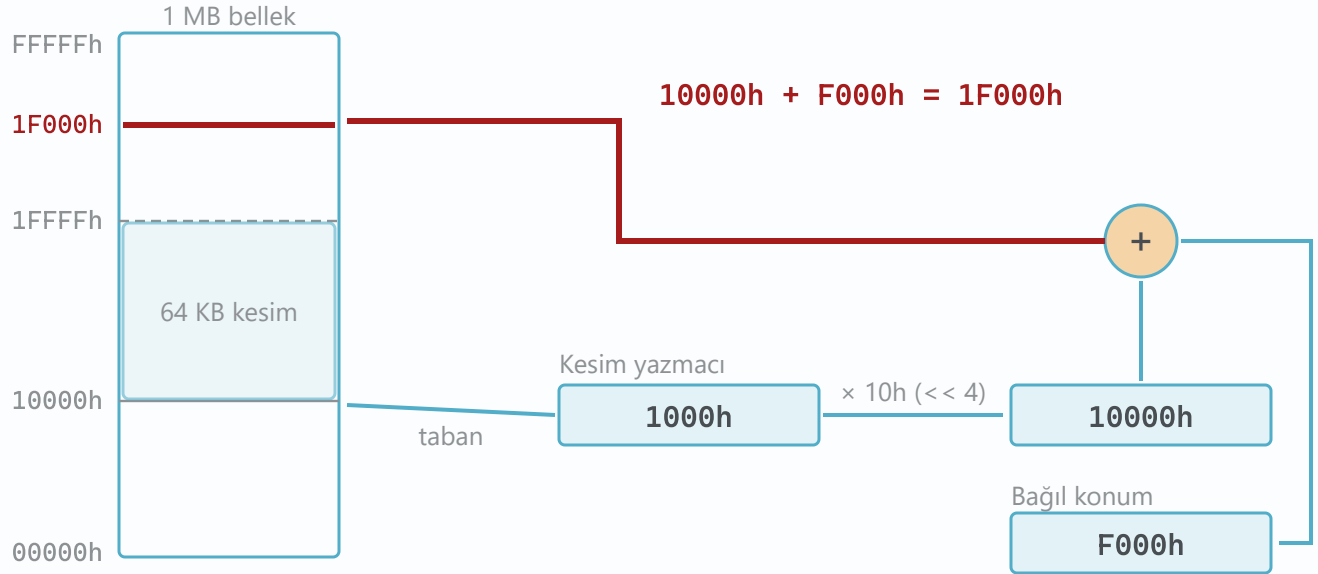
Kesim değeri 4 bit **sola kaydırılır** ($\times 10h$): $1000h \rightarrow \mathbf{10000h}$. Kesim tabanı her zaman 16'nın katıdır (paragraf sınırı).

Kesim ve Bağlı Konum



Bağlı konum **F000h**. En büyük bağlı konum FFFFh olduğundan kesim 10000h–1FFFFh aralığını (64 KB) kapsar.

Kesim ve Bağıl Konum



Toplayıcı: **10000h + F000h = 1F000h**. Bu 20 bitlik değer adres yoluna çıkar.

Adım Adım: CS:IP → Fiziksel Adres

Örnek 1: CS = 4321h, IP = 1000h

CS (16 bit)



CS = **4321h** (4 onaltılık basamak = 16 bit).

Adım 1 / 8

Adım Adım: CS:IP → Fiziksel Adres

Örnek 1: CS = 4321h, IP = 1000h

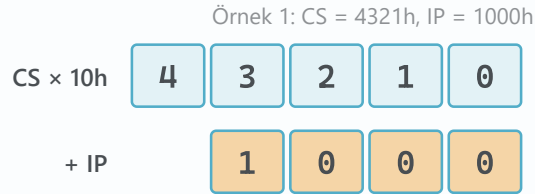
CS × 10h



← sağa bir 0 eklendi

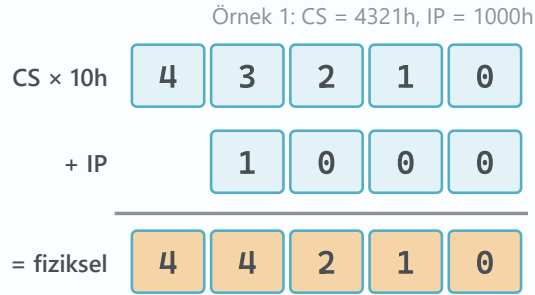
× 10h = onaltılık olarak sağa bir **0** eklemek (4 bit sola kaydırma): **43210h**.

Adım Adım: CS:IP → Fiziksel Adres



IP = **1000h** sağa hizalanarak altına yazılır (en soldaki basamak boş = 0).

Adım Adım: CS:IP → Fiziksel Adres



$$43210h + 1000h = 44210h$$

Basamakları sağdan toplarız: 0+0, 1+0, 2+0, 3+1, 4+0 → **44210h**. Elde oluşmadı.

Adım Adım: CS:IP → Fiziksel Adres

Örnek 2: CS = F000h, IP = FFF0h

CS (16 bit)



CS = **F000h**, IP = **FFF0h** (8086'nin **sıfırlama sonrası** ilk komutunun adresi BIOS'ta).

Adım Adım: CS:IP → Fiziksel Adres

Örnek 2: CS = F000h, IP = FFF0h

CS × 10h



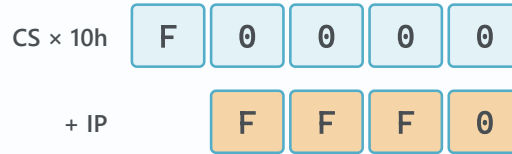
← sağa bir 0 eklendi

$$\text{F000h} \times 10\text{h} = \mathbf{\text{F0000h}}.$$

Adım 6 / 8

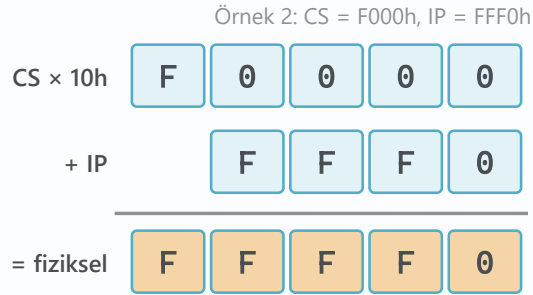
Adım Adım: CS:IP → Fiziksel Adres

Örnek 2: CS = F000h, IP = FFF0h



IP = FFF0h sağa hizalanır.

Adım Adım: CS:IP → Fiziksel Adres



$$\text{F0000h} + \text{FFF0h} = \text{FFFF0h}$$

F0000h + FFF0h = **FFFF0h**: 1 MB'ın son 16 baytı. Reset sonrası işlemci ilk komutu buradan alır.

Demo: Fiziksel Adres Hesaplayıcı

Kesim : Bağlı konum rastgele

4321

1000

h

h

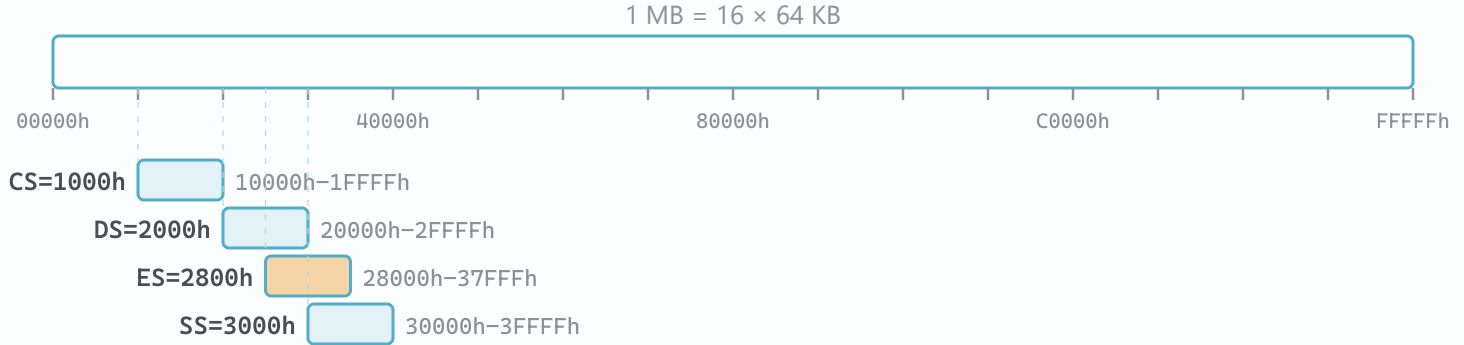
```
4321h      ; kesim
43210h     ; kesim × 10h (sola 4 bit)
+ 1000h    ; bağlı konum
-----
44210h     ; fiziksel adres
```



Kesim aralığı **43210h–5320Fh** · normalize edilmiş yazım **4421h:0000h**

Kesim tabanı ve 64 KB'lık kesim aralığı haritada gösterilir. Toplam 1 MB'ı aşarsa 8086 adresi 20 bite kırpar (sarma).

Bellek Kesimleme



ES kesimi DS ve SS ile çakışıyor: kesimler üst üste binebilir.

- 20 bitlik adres yolu 1 MB'ı adresler; bir kesim en fazla **64 KB**'tır (16 bitlik offset).
- Herhangi bir anda yalnızca **4 kesim etkindir** (CS, DS, SS, ES) → aynı anda en çok 256 KB doğrudan erişilir. Başka bölgeye erişmek için kesim yazmacı değiştirilir.
- Kesimler sabit 16 parça değildir: her **paragraf** (16 bayt) sınırında başlayabilir, çakışabilir veya ayrık olabilir.

1 MB / 64 KB = 16 yalnızca bir hesaptır; 8086'da kesim başlangıçları sabit değildir (65 536 olası başlangıç).

Neden Kesimleme?

Artıları

- + 16 bitlik yazmaçlarla 20 bitlik (1 MB) bellek adreslenir.
- + Kod, veri ve yığın birbirinden ayrılır; program düzeni netleşir.
- + **Yeniden konumlanabilirlik:** yalnızca kesim yazmaçları değiştirilerek program bellekte başka yere yüklenebilir.
- + Komutlar kısa kalır: çoğu adres 16 bitlik bağlı konum ile yazılır.

Eksileri

- Tek bir dizi/yapı 64 KB'ı aşamaz (aşarsa kesim değiştirmek gerekir).
- Aynı fiziksel adresin birçok `kesim:offset` yazımı vardır → işaretçi karşılaştırması zorlaşır.
- Kesimler arasında koruma yoktur: hatalı bir yazma başka kesimi bozabilir.
- Adres hesabı programcı için ek zihinsel yük.

Aynı Adres, Farklı Kesim:Bağıl Konum

Mantıksal adres	Hesap	Fiziksel adres
1234h:0005h	12340h + 0005h	12345h
1000h:2345h	10000h + 2345h	12345h
1200h:0345h	12000h + 0345h	12345h
0234h:FFF5h	02340h + FFF5h	12335h (dikkat: farklı!)
FFFFh:0010h	FFFF0h + 0010h = 100000h	00000h (8086'da 20 bite sarar)

Normalleştirme: bağıl konumu 0–Fh arasına indirmek için kesim = FA'nın ilk 4 basamağı, bağıl konum = son basamak. 12345h → 1234h:0005h . Bir fiziksel adresin en fazla 4096 farklı yazımı vardır.

Sarma (wrap-around): 8086'da A20 hattı yoktur; 1 MB'ı aşan toplam 00000h'ye döner. 80286+ işlemcilerde bu davranış "A20 kapısı" ile taklit edilir.

Varsayılan Kesimler

Erişim türü	Kesim	Bağlı konum kaynağı	Değiştirilebilir mi?
Komut getirme	CS	IP	hayır
Yığın işlemleri (PUSH, POP, CALL, RET)	SS	SP	hayır
BP ile adresleme [BP+...]	SS	etkin adres	evet: DS : [BP]
Genel veri [BX] , [SI] , [DI] , [1234h]	DS	etkin adres	evet: ES : [BX]
Dizgi kaynağı (MOVSB, LODSB)	DS	SI	evet
Dizgi hedefi (MOVSB, STOSB)	ES	DI	hayır

Varsayılan kesimi değiştirmek için **kesim öneki** (segment override) kullanılır: `MOV AL, ES : [BX]` → fiziksel adres = $ES \times 10h + BX$. Önek komuta 1 bayt ekler (26h).

Kod İzleme: Aynı Bağlı Konum, Üç Kesim

```
1 org 100h
▶ mov ax, 2000h
▶ mov ds, ax
4 mov ax, 3000h
5 mov es, ax
6 mov bx, 0010h
7 mov bp, 0010h
8 mov byte ptr [bx], 11h ; DS
9 mov byte ptr [bp], 22h ; SS
10 mov byte ptr es:[bx], 33h ; ES
11 mov al, [bx] ; DS
12 mov ah, [bp] ; SS
13 ret
```

kesim yazmaçları

DS = 2000h

ES = 0700h

SS = 0700h

BX = 0000h · BP = 0000h

ES kesimi (henüz yok)

30010h

??

DS kesimi

20010h

??

SS kesimi

07010h

??

Kesim yazmacına sabit yüklenemez; AX üzerinden **DS = 2000h** yapılır. COM programında SS = CS = **0700h** kalır.

Adım 1 / 8

DEĞİŞKENLER

DS = 2000h

ES = 0700h

SS = 0700h

BX = 0000h

BP = 0000h

AX = 2000h

Kod İzleme: Aynı Bağlı Konum, Üç Kesim

```
1 org 100h
2 mov ax, 2000h
3 mov ds, ax
▶ mov ax, 3000h
▶ mov es, ax
6 mov bx, 0010h
7 mov bp, 0010h
8 mov byte ptr [bx], 11h ; DS
9 mov byte ptr [bp], 22h ; SS
10 mov byte ptr es:[bx], 33h ; ES
11 mov al, [bx] ; DS
12 mov ah, [bp] ; SS
13 ret
```

kesim yazmaçları

DS = 2000h

ES = 3000h

SS = 0700h

BX = 0000h · BP = 0000h

ES kesimi

30010h

??

DS kesimi

20010h

??

SS kesimi

07010h

??

Aynı yolla **ES = 3000h**. Artık üç farklı kesim tabanı var:
07000h (SS), 20000h (DS), 30000h (ES).

Adım 2 / 8

DEĞİŞKENLER

DS = 2000h

ES = 3000h

SS = 0700h

BX = 0000h

BP = 0000h

AX = 3000h

Kod İzleme: Aynı Bağlı Konum, Üç Kesim

```
1 org 100h
2 mov ax, 2000h
3 mov ds, ax
4 mov ax, 3000h
5 mov es, ax
▶ mov bx, 0010h
▶ mov bp, 0010h
8 mov byte ptr [bx], 11h ; DS
9 mov byte ptr [bp], 22h ; SS
10 mov byte ptr es:[bx], 33h ; ES
11 mov al, [bx] ; DS
12 mov ah, [bp] ; SS
13 ret
```

kesim yazmaçları

DS = 2000h

ES = 3000h

SS = 0700h

BX = 0010h · BP = 0010h

ES kesimi

30010h

??

DS kesimi

20010h

??

SS kesimi

07010h

??

BX ve BP aynı bağlı konumu tutar: **0010h**. Hangi bayta erişileceğini bağlı konum değil, **kullanılan kesim** belirleyecek.

Adım 3 / 8

DEĞİŞKENLER

DS = 2000h

ES = 3000h

SS = 0700h

BX = 0010h

BP = 0010h

AX = 3000h

Kod İzleme: Aynı Bağlı Konum, Üç Kesim

```
1 org 100h
2 mov ax, 2000h
3 mov ds, ax
4 mov ax, 3000h
5 mov es, ax
6 mov bx, 0010h
7 mov bp, 0010h
▶ mov byte ptr [bx], 11h ; DS
9 mov byte ptr [bp], 22h ; SS
10 mov byte ptr es:[bx], 33h ; ES
11 mov al, [bx] ; DS
12 mov ah, [bp] ; SS
13 ret
```

$$DS \times 10h + BX$$
$$\begin{array}{r} 20000h \\ + 0010h \\ \hline \end{array}$$
20010h**yaz 11h**

ES kesimi

30010h

??

DS kesimi

20010h**11h**

SS kesimi

07010h

??

[bx] varsayılan olarak **DS** kullanır: $2000h \times 10h + 0010h$
= **20010h** ← 11h.

Adım 4 / 8

DEĞİŞKENLER

DS = **2000h**ES = **3000h**SS = **0700h**BX = **0010h**BP = **0010h**AX = **3000h**

Kod İzleme: Aynı Bağlı Konum, Üç Kesim

```
1 org 100h
2 mov ax, 2000h
3 mov ds, ax
4 mov ax, 3000h
5 mov es, ax
6 mov bx, 0010h
7 mov bp, 0010h
8 mov byte ptr [bx], 11h ; DS
9 ► mov byte ptr [bp], 22h ; SS
10 mov byte ptr es:[bx], 33h ; ES
11 mov al, [bx] ; DS
12 mov ah, [bp] ; SS
13 ret
```

$$SS \times 10h + BP$$
$$\begin{array}{r} 07000h \\ + 0010h \\ \hline \end{array}$$
07010h**yaz 22h**

ES kesimi

30010h

??

DS kesimi

20010h**11h**

SS kesimi

07010h**22h**

[bp] varsayılan olarak **SS** kullanır: $0700h \times 10h + 0010h =$
07010h ← 22h. Aynı bağlı konum, başka bayt.

Adım 5 / 8

DEĞİŞKENLER

DS = **2000h**ES = **3000h**SS = **0700h**BX = **0010h**BP = **0010h**AX = **3000h**

Kod İzleme: Aynı Bağlı Konum, Üç Kesim

```
1 org 100h
2 mov ax, 2000h
3 mov ds, ax
4 mov ax, 3000h
5 mov es, ax
6 mov bx, 0010h
7 mov bp, 0010h
8 mov byte ptr [bx], 11h ; DS
9 mov byte ptr [bp], 22h ; SS
▶ mov byte ptr es:[bx], 33h ; ES
11 mov al, [bx] ; DS
12 mov ah, [bp] ; SS
13 ret
```

$$ES \times 10h + BX$$
$$\begin{array}{r} 30000h \\ + 0010h \\ \hline \end{array}$$
30010h**yaz 33h**

ES kesimi

30010h**33h**

DS kesimi

20010h**11h**

SS kesimi

07010h**22h**

es: kesim öneki DS yerine **ES**'yi seçtirir (+1 bayt, 26h):
 $30000h + 0010h = \mathbf{30010h} \leftarrow 33h.$

Adım 6 / 8

DEĞİŞKENLER

DS = **2000h**ES = **3000h**SS = **0700h**BX = **0010h**BP = **0010h**AX = **3000h**

Kod İzleme: Aynı Bağlı Konum, Üç Kesim

```
1 org 100h
2 mov ax, 2000h
3 mov ds, ax
4 mov ax, 3000h
5 mov es, ax
6 mov bx, 0010h
7 mov bp, 0010h
8 mov byte ptr [bx], 11h ; DS
9 mov byte ptr [bp], 22h ; SS
10 mov byte ptr es:[bx], 33h ; ES
▶ mov al, [bx] ; DS
12 mov ah, [bp] ; SS
13 ret
```

$$DS \times 10h + BX$$

20000h
+ 0010h

20010h

oku → AL

ES kesimi

30010h

33h

DS kesimi

20010h

11h

SS kesimi

07010h

22h

`mov al, [bx]` yine DS:0010h'yi okur → AL = **11h**
(ES:0010h'deki 33h değil).

Adım 7 / 8

DEĞİŞKENLER

DS = 2000h

ES = 3000h

SS = 0700h

BX = 0010h

BP = 0010h

AX = 3011h

Kod İzleme: Aynı Bağlı Konum, Üç Kesim

```
1 org 100h
2 mov ax, 2000h
3 mov ds, ax
4 mov ax, 3000h
5 mov es, ax
6 mov bx, 0010h
7 mov bp, 0010h
8 mov byte ptr [bx], 11h ; DS
9 mov byte ptr [bp], 22h ; SS
10 mov byte ptr es:[bx], 33h ; ES
11 mov al, [bx] ; DS
12 ► mov ah, [bp] ; SS
13 ret
```

$$SS \times 10h + BP$$

07000h
+ 0010h

07010h

oku → AH

ES kesimi

30010h

33h

DS kesimi

20010h

11h

SS kesimi

07010h

22h

`mov ah, [bp]` SS:0010h'yi okur → AH = **22h**. Sonuç AX = **2211h**: üç erişim, aynı bağlı konum, üç farklı fiziksel adres.

Adım 8 / 8

DEĞİŞKENLER

DS = 2000h

ES = 3000h

SS = 0700h

BX = 0010h

BP = 0010h

AX = 2211h

Kod İzleme: Kesim:Bağlı Konum ile Erişim

AX ← 2000h. Kesim yazmacına anlık değer yüklenemediği için önce AX'e yüklenir.

Adım 1 / 10

► **mov ax, 2000h**

2 mov ds, ax ; DS = 2000h

3 mov bx, 0010h

4 mov byte ptr [bx], 41h

5 mov ax, 2001h

6 mov ds, ax ; +16 bayt

7 mov bx, 0000h

8 mov al, [bx] ; aynı bayt

9 mov si, 0FFFFh

10 mov ah, [si] ; son bayt

DEĞİŞKENLER

AX = **2000h**

DS = ?

BX = **0000h**

SI = **0000h**

FA = -

Kod İzleme: Kesim:Bağlı Konum ile Erişim

DS ← AX = 2000h. Veri kesimi artık **20000h–2FFFFh**.

Adım 2 / 10

```
1 mov ax, 2000h
▶ mov ds, ax      ; DS = 2000h
3 mov bx, 0010h
4 mov byte ptr [bx], 41h
5 mov ax, 2001h
6 mov ds, ax      ; +16 bayt
7 mov bx, 0000h
8 mov al, [bx]     ; aynı bayt
9 mov si, 0FFFFh
10 mov ah, [si]    ; son bayt
```

DEĞİŞKENLER

AX = 2000h

DS = 2000h

BX = 0000h

SI = 0000h

FA = -

Kod İzleme: Kesim:Bağlı Konum ile Erişim

BX ← 0010h (bağlı konum).

Adım 3 / 10

```
1 mov ax, 2000h
2 mov ds, ax          ; DS = 2000h
▶ 3 mov bx, 0010h
4 mov byte ptr [bx], 41h
5 mov ax, 2001h
6 mov ds, ax          ; +16 bayt
7 mov bx, 0000h
8 mov al, [bx]         ; aynı bayt
9 mov si, 0FFFFh
10 mov ah, [si]        ; son bayt
```

DEĞİŞKENLER

AX = 2000h

DS = 2000h

BX = 0010h

SI = 0000h

FA = -

Kod İzleme: Kesim:Bağlı Konum ile Erişim

FA = $2000h \times 10h + 0010h$ = **20010h**. Oraya 41h ('A') yazılır.

Adım 4 / 10

```
1 mov ax, 2000h
2 mov ds, ax          ; DS = 2000h
3 mov bx, 0010h
▶ mov byte ptr [bx], 41h
5 mov ax, 2001h
6 mov ds, ax          ; +16 bayt
7 mov bx, 0000h
8 mov al, [bx]         ; aynı bayt
9 mov si, 0FFFFh
10 mov ah, [si]        ; son bayt
```

DEĞİŞKENLER

AX = 2000h

DS = 2000h

BX = 0010h

SI = 0000h

FA = 20010h

[20010h] = 41h

Kod İzleme: Kesim:Bağlı Konum ile Erişim

AX ← 2001h.

Adım 5 / 10

```
1 mov ax, 2000h
2 mov ds, ax          ; DS = 2000h
3 mov bx, 0010h
4 mov byte ptr [bx], 41h
▶ mov ax, 2001h
6 mov ds, ax          ; +16 bayt
7 mov bx, 0000h
8 mov al, [bx]        ; aynı bayt
9 mov si, 0FFFFh
10 mov ah, [si]       ; son bayt
```

DEĞİŞKENLER

AX = 2001h

DS = 2000h

BX = 0010h

SI = 0000h

FA = 20010h

[20010h] = 41h

Kod İzleme: Kesim:Bağlı Konum ile Erişim

DS ← 2001h: kesim tabanı 16 bayt (bir paragraf) ileri kaydı → **20010h–3000Fh**.

Adım 6 / 10

```
1 mov ax, 2000h
2 mov ds, ax          ; DS = 2000h
3 mov bx, 0010h
4 mov byte ptr [bx], 41h
5 mov ax, 2001h
▶ mov ds, ax          ; +16 bayt
7 mov bx, 0000h
8 mov al, [bx]         ; aynı bayt
9 mov si, 0FFFFh
10 mov ah, [si]        ; son bayt
```

DEĞİŞKENLER

AX = 2001h

DS = 2001h

BX = 0010h

SI = 0000h

FA = 20010h

[20010h] = 41h

Kod İzleme: Kesim:Bağlı Konum ile Erişim

BX ← 0000h.

Adım 7 / 10

```
1 mov ax, 2000h
2 mov ds, ax          ; DS = 2000h
3 mov bx, 0010h
4 mov byte ptr [bx], 41h
5 mov ax, 2001h
6 mov ds, ax          ; +16 bayt
▶ mov bx, 0000h
8 mov al, [bx]         ; aynı bayt
9 mov si, 0FFFFh
10 mov ah, [si]        ; son bayt
```

DEĞİŞKENLER

AX = 2001h

DS = 2001h

BX = 0000h

SI = 0000h

FA = 20010h

[20010h] = 41h

Kod İzleme: Kesim:Bağlı Konum ile Erişim

FA = 20010h + 0000h = **20010h** → AL = 41h. 2000h:0010h ile 2001h:0000h **aynı bayt!**

Adım 8 / 10

```
1 mov ax, 2000h
2 mov ds, ax          ; DS = 2000h
3 mov bx, 0010h
4 mov byte ptr [bx], 41h
5 mov ax, 2001h
6 mov ds, ax          ; +16 bayt
7 mov bx, 0000h
▶ mov al, [bx]         ; aynı bayt
9 mov si, 0FFFFh
10 mov ah, [si]        ; son bayt
```

DEĞİŞKENLER

AX = **2041h**

DS = **2001h**

BX = **0000h**

SI = **0000h**

FA = **20010h**

[20010h] = **41h**

Kod İzleme: Kesim:Bağlı Konum ile Erişim

SI ← FFFFh: en büyük bağlı konum.

Adım 9 / 10

```
1 mov ax, 2000h
2 mov ds, ax          ; DS = 2000h
3 mov bx, 0010h
4 mov byte ptr [bx], 41h
5 mov ax, 2001h
6 mov ds, ax          ; +16 bayt
7 mov bx, 0000h
8 mov al, [bx]         ; aynı bayt
▶ mov si, 0FFFFh
10 mov ah, [si]         ; son bayt
```

DEĞİŞKENLER

AX = 2041h

DS = 2001h

BX = 0000h

SI = FFFFh

FA = 20010h

[20010h] = 41h

Kod İzleme: Kesim:Bağlı Konum ile Erişim

FA = 20010h + FFFFh = **3000Fh**: bu kesimin son baytı. AH bu bayttaki değeri alır.

Adım 10 / 10

```
1 mov ax, 2000h
2 mov ds, ax          ; DS = 2000h
3 mov bx, 0010h
4 mov byte ptr [bx], 41h
5 mov ax, 2001h
6 mov ds, ax          ; +16 bayt
7 mov bx, 0000h
8 mov al, [bx]         ; aynı bayt
9 mov si, 0FFFFh
▶ mov ah, [si]         ; son bayt
```

DEĞİŞKENLER

AX = ??41h

DS = 2001h

BX = 0000h

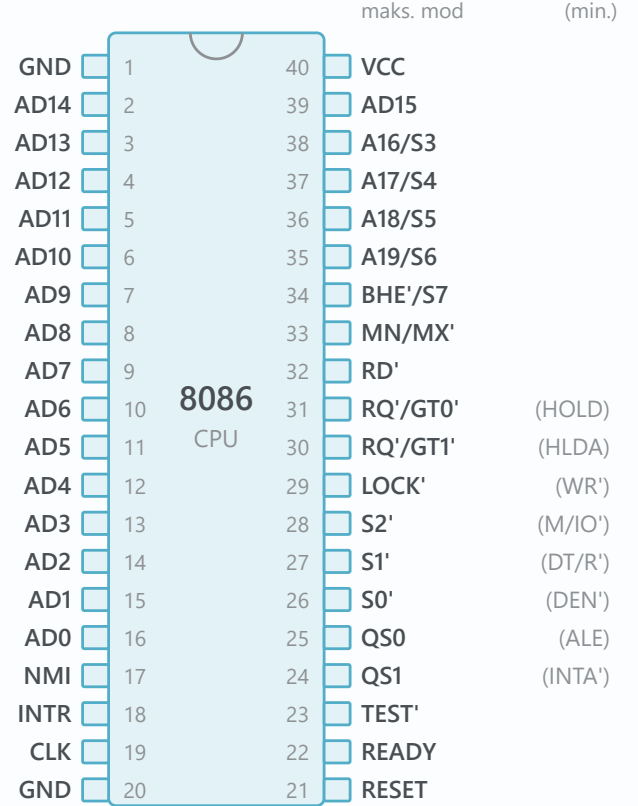
SI = FFFFh

FA = 3000Fh

[20010h] = 41h

8086 Pin Diyagramı

- 40 pinli DIP paket; pin 1 ve 20 toprak (GND), pin 40 besleme (+5 V).
- **AD0–AD15**: adres ve veri aynı pinlerden zaman paylaşımli (çoklanmıř) geer.
- **A16–A19**: adresin st 4 biti, durum bitleriyle oklanır → toplam 20 bit adres.
- Pin 24–31'in iřlevi **MN/MX'** pinine gre deėiřir (parantez iindekiler minimum mod).
- ' iřareti (diyagramda st izgi) sinyalin **aktif dřk** olduėunu gsterir.
- Ayrıntılar: **Blm 2 – 8086 Pinler**.



ÖZET

Özet

Genel

16 bit veri yolu ve yazmaçlar, 20 bit adres yolu
→ 1 MB. CISC, değişken uzunlukta komutlar.

14 yazmaç

AX, BX, CX, DX · SP, BP, SI, DI · CS, DS, SS, ES ·
IP, FLAGS.

Adres hesabı

$FA = \text{kesim} \times 10h + \text{bağlı konum}$. Kesim ≤ 64
KB, paragraf sınırında başlar, kesimler
çakışabilir.

BIU + EU

BIU getirir ve adres üretir; EU çözer ve yürütür.
6 baytlık FIFO kuyruk ile ikisi paralel çalışır.

Bayraklar

Durum: CF, PF, AF, ZF, SF, OF. Kontrol: TF, IF,
DF.

Dallanma

IP'ye yeni değer yükler ve önden getirme
kuyruğunu temizler.

ÖZET

Alıştırmalar

1. CS = `2F00h` , IP = `01A4h` ise sonraki komutun fiziksel adresi nedir?
2. `3A2C5h` fiziksel adresini normalleştirilmiş `kesim:offset` biçiminde yazın; DS = `3000h` ise bağlı konum ne olmalıdır?
3. AX = `80FFh` iken `ADD AL, 1` sonrası AX, CF, ZF, SF, OF ne olur?
4. `MOV DS, 1000h` neden derlenmez? Doğru yazımı verin.
5. Kuyrukta 6 bayt varken EU bir `JMP` yürütürse kuyruğa ne olur? Neden?
6. 8086 bir komutu getirirken aynı anda başka komutu yürütebilir mi? Hangi birimler sayesinde?

► İpuçları / cevaplar

Kaynaklar

- Intel, *8086 16-Bit HMOS Microprocessor* veri sayfası (231455).
- Intel, *The 8086 Family User's Manual*, 1979.
- B. B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009.
- M. A. Mazidi, J. G. Mazidi, D. Causey, *The x86 PC: Assembly Language, Design, and Interfacing*, 5. baskı, Pearson, 2010.
- emu8086 belgeleri ve komut kümesi başvurusu.

Son

Bölüm 1: 8086 Mimarisi

Sonraki bölüm: Bölüm 1: Programlama Dilleri · [Terim sözlüğü](#)