Bölüm 1.1: Giriş

Mikroişlemciler

Mikroişlemci ve mikrodenetleyici | Getir · çöz · yürüt | Mikroişlemci yapısı | Saat hızı

CISC · RISC · EPIC | Özel amaçlı işlemciler | Intel x86 evrimi | Modern işlemci özellikleri

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Temel kavramlar

Mikroişlemciyi, mikrodenetleyiciyi ve bir bilgisayar sisteminin yollarını tanımlamak.

Komut döngüsü

Getir–çöz–yürüt döngüsünü ve program sayacının (IP/PC) rolünü adım adım izlemek.

Performans

Saat frekansı, periyot, CPI ve çalışma süresi arasındaki ilişkiyi hesaplamak.

Mimari türleri

CISC, RISC, EPIC ile vektör, SIMD, DSP, GPU gibi işlemci türlerini karşılaştırmak.

x86 ailesi

4004'ten Core i7'ye Intel işlemcilerinin gelişimini ve 8086'nın yerini açıklamak.

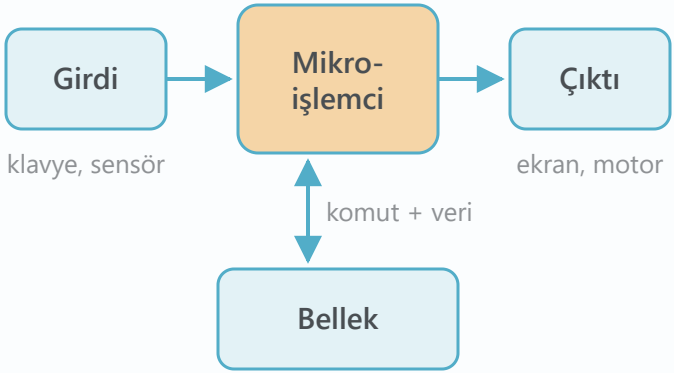
Modern özellikler

Önbellek, çok çekirdek, sanallaştırma ve güç yönetimi gibi kavramları tanımak.

TEMEL KAVRAMLAR

Mikroişlemci Nedir?

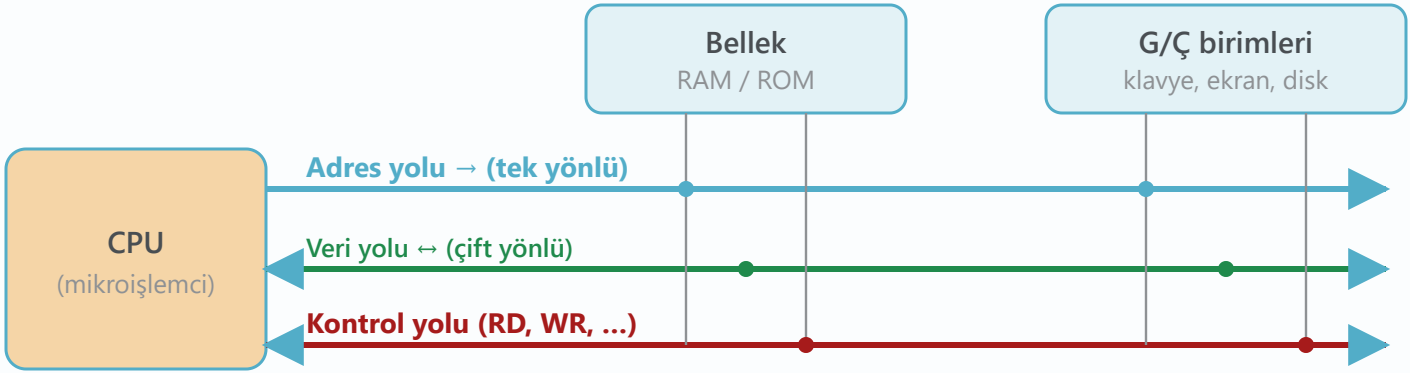
- Bilgisayarın **beyni**: merkezî işlem biriminin (CPU) tek bir yonga üzerinde gerçeklenmiş hâli.
- **Programlanabilir** bir aygıttır; ne yapacağını bellekteki komutlar belirler.
- Temel iş akışı:
  - **girdi** alır,
  - **aritmetik ve mantıksal** işlemler yapar,
  - istenen **çıktıyı** üretir.
- Makine dilindeki komutları okur ve uygular.



Örnek: hesap makinesinde tuşlar **girdi**, toplama işlemi **işlem**, ekrandaki sonuç **çıktıdır**.

TEMEL KAVRAMLAR

Bilgisayar Sistemi ve Yollar (Bus)



Adres yolu

Hangi bellek gözüne / G/Ç kapısına erişileceğini seçer. n hat → 2^n adres. 8086: **20 bit** → **1 MB**.

Veri yolu

Okunan ya da yazılan veriyi taşır. Genişliği bir seferde aktarılan bit sayısıdır. 8086: **16 bit**.

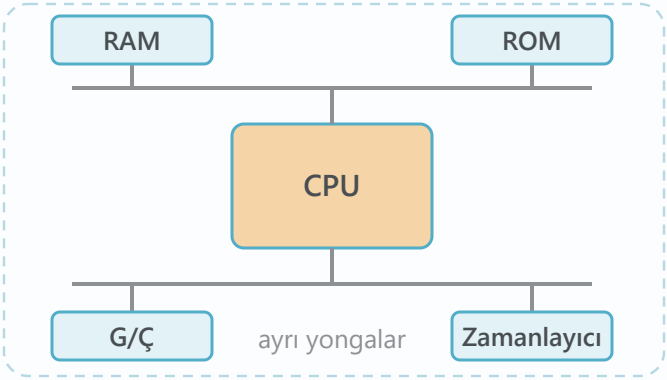
Kontrol yolu

Okuma/yazma, bellek mi G/Ç mı, saat, kesme gibi zamanlama ve kontrol sinyalleri.

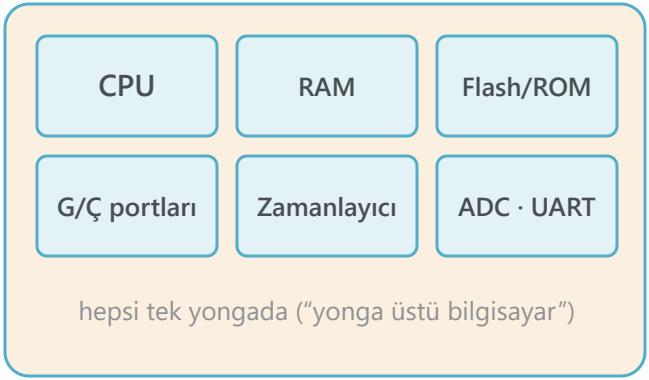
TEMEL KAVRAMLAR

Mikroişlemci ve Mikrodenetleyici

Mikroişlemci (MPU) tabanlı sistem



Mikrodenetleyici (MCU)



Mikroişlemci

Yalnızca CPU; bellek ve çevre birimleri dışarıdan eklenir. Esnek ve güçlü. Ör: **8086**, Core i7, Ryzen.

Mikrodenetleyici

CPU + bellek + çevre birimleri tek yongada. Ucuz, az enerji, tek işe adanmış. Ör: 8051, PIC, AVR (Arduino), STM32.

TEMEL KAVRAMLAR

Mikroişlemci mi, Mikrodenetleyici mi?

Ölçüt	Mikroişlemci (MPU)	Mikrodenetleyici (MCU)
İçerik	CPU (ALU, kontrol birimi, yazmaçlar)	CPU + RAM + Flash + G/Ç + zamanlayıcı + ADC…
Bellek	dışarıda, büyük (GB)	yonga içinde, küçük (KB–MB)
Saat hızı	yüksek (GHz)	düşük–orta (MHz)
Güç	yüksek (W–100 W)	çok düşük (mW, uyku modunda µW)
Maliyet	yüksek; ek yongalar gerekir	düşük; tek yonga yeterli
Yazılım	genelde işletim sistemi + çok sayıda uygulama	tek, sabit bir program (gömülü yazılım)
Kullanım	PC, sunucu, dizüstü	çamaşır makinesi, kumanda, otomobil ECU, IoT

Örnek işlemci **8086**: yolları, yazmaçları ve komutları MCU'larda da aynı mantıkla çalışır.

TEMEL KAVRAMLAR

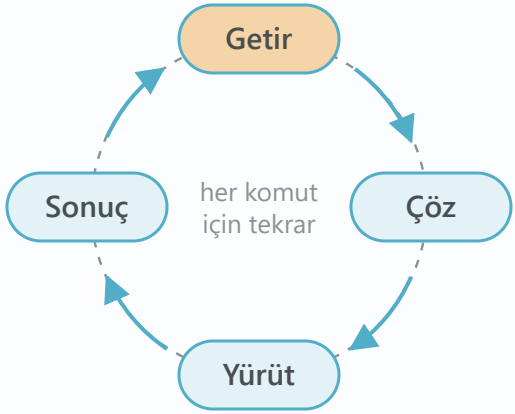
Ana Görevler: Komut Döngüsü

1 · Getir (fetch)
Komutu, PC'nin gösterdiği adresten bellekten çeker.

2 · Çöz (decode)
İşlem kodunu çözer: hangi işlem, hangi işlenenler?

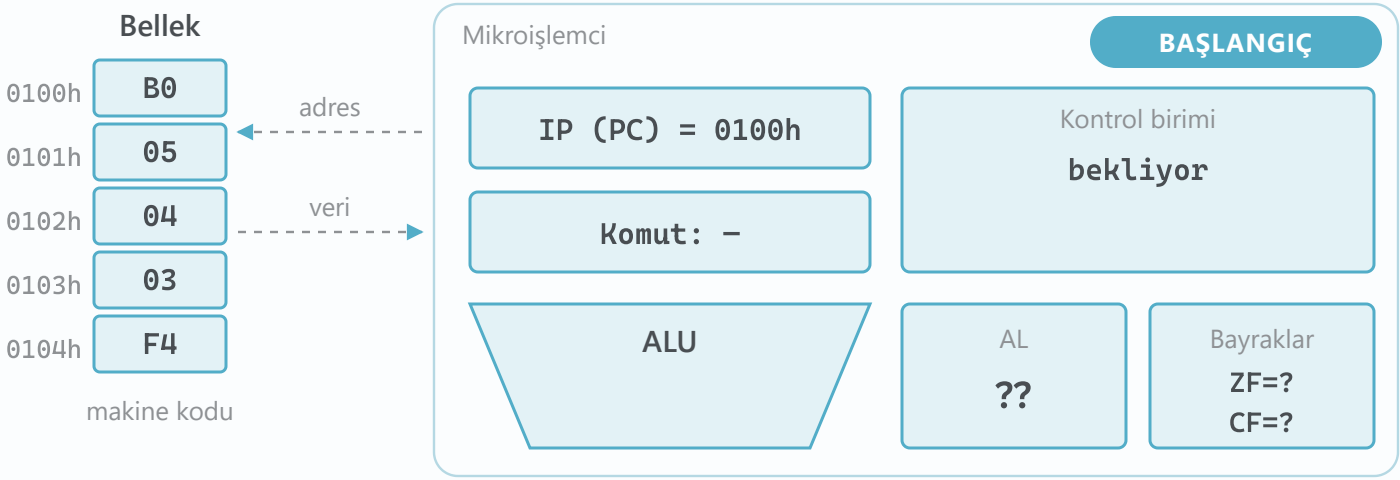
3 · Yürüt (execute)
Çözülen komutu uygular (ALU, veri taşıma, atlama).

4 · Sonuç (result)
Sonucu yazmaca/belleğe yazar, bayrakları günceller.



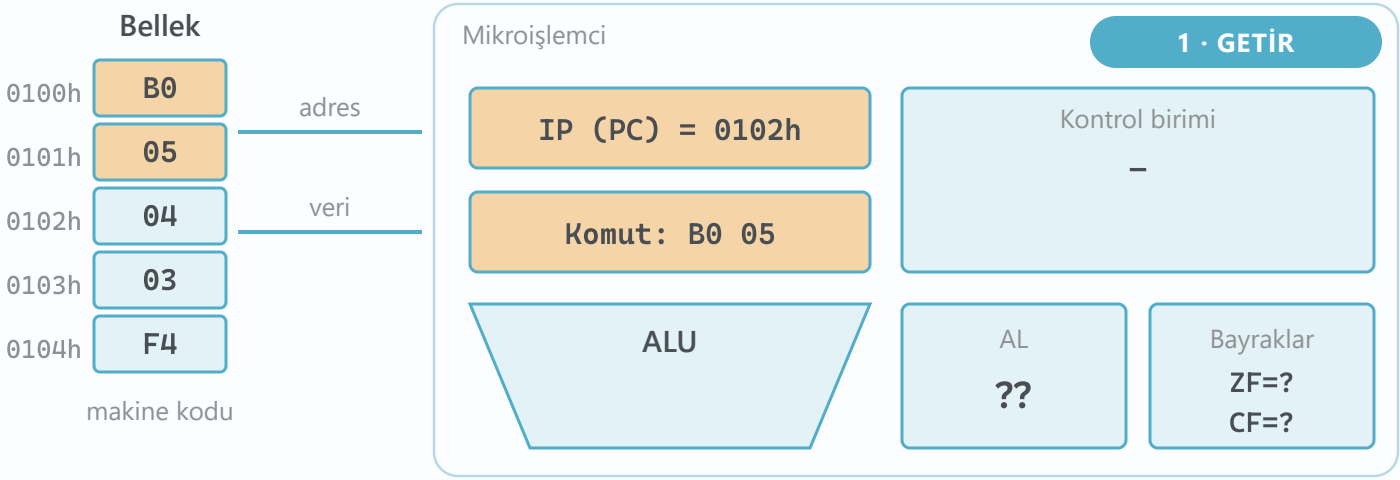
İşlemci açık olduğu sürece bu döngü durmaksızın tekrarlanır. Hızlı işlemciler aşamaları **üst üste bindirir** (boru hattı).

Komut Döngüsü: `MOV AL,5` · `ADD AL,3` · `HLT`



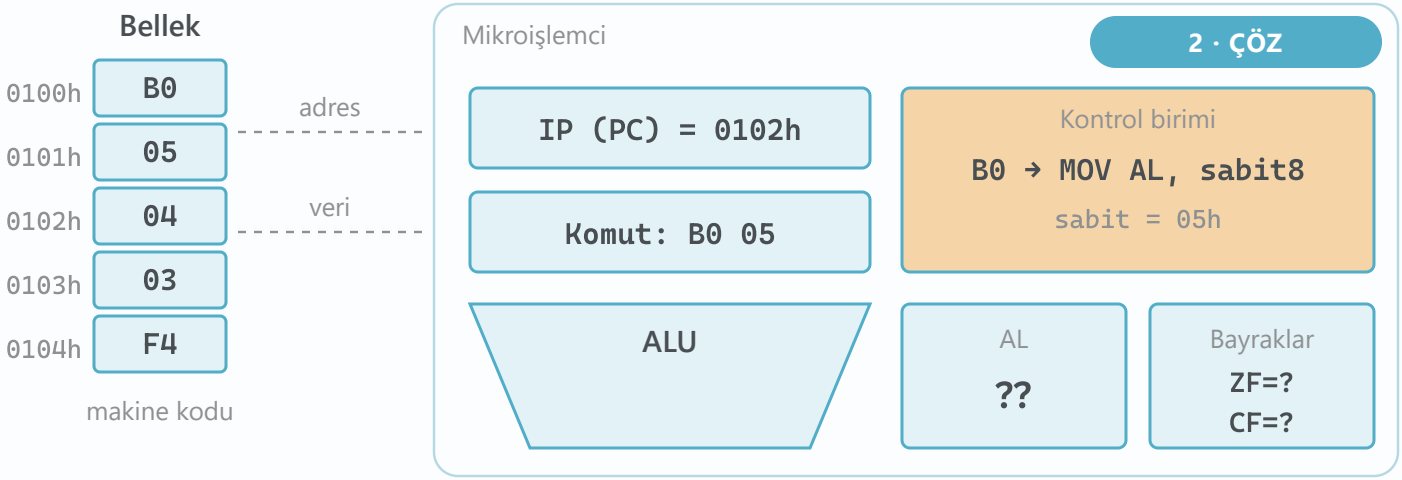
Program belleğe yüklenmiş. **IP** (8086'da program sayacının adı) ilk komutun adresini, **0100h**'yi gösteriyor.

Komut Döngüsü: `MOV AL,5` · `ADD AL,3` · `HLT`



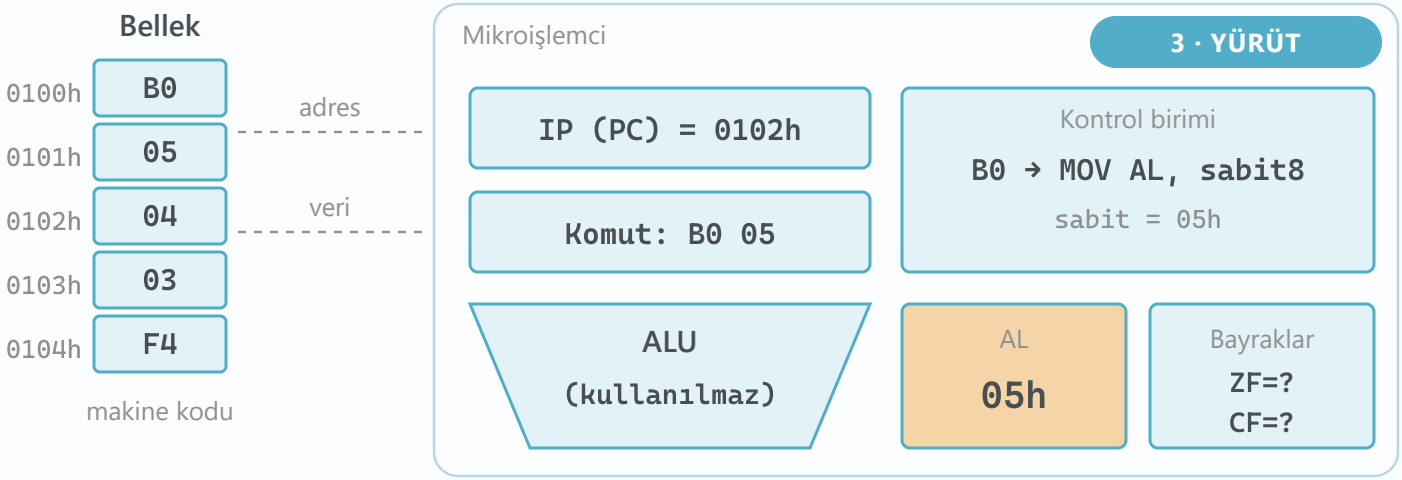
Getir: IP adres yoluna konur, bellekten **B0 05** okunur. IP komut uzunluğu (2 bayt) kadar artar → **0102h**.

Komut Döngüsü: `MOV AL,5` · `ADD AL,3` · `HLT`



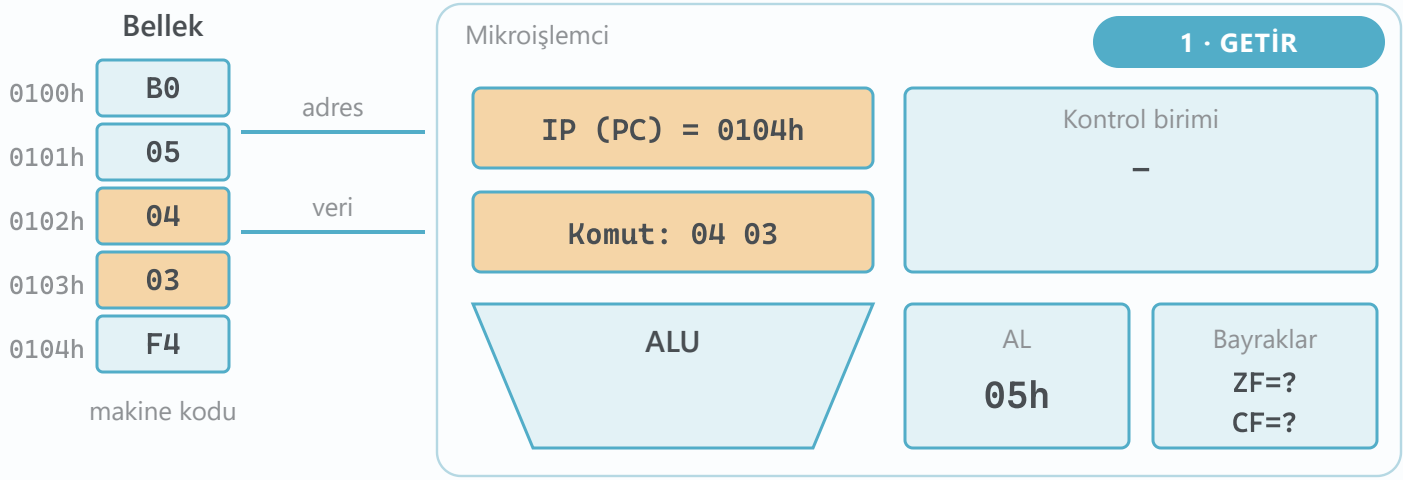
Çöz: kontrol birimi **B0** işlem kodunu “AL'ye 8 bitlik sabit yükle” olarak çözer; sabit 05h.

Komut Döngüsü: `MOV AL,5` · `ADD AL,3` · `HLT`



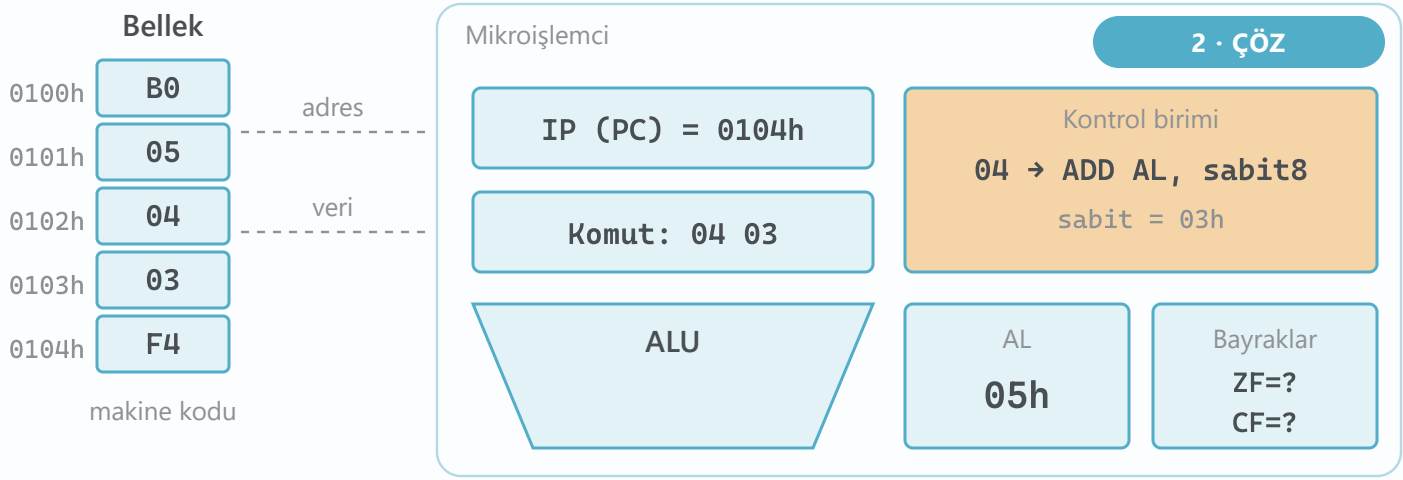
Yürüt / sonuç: 05h, AL yazmacına yazılır. MOV bayrakları **etkilemez**.

Komut Döngüsü: `MOV AL,5` · `ADD AL,3` · `HLT`



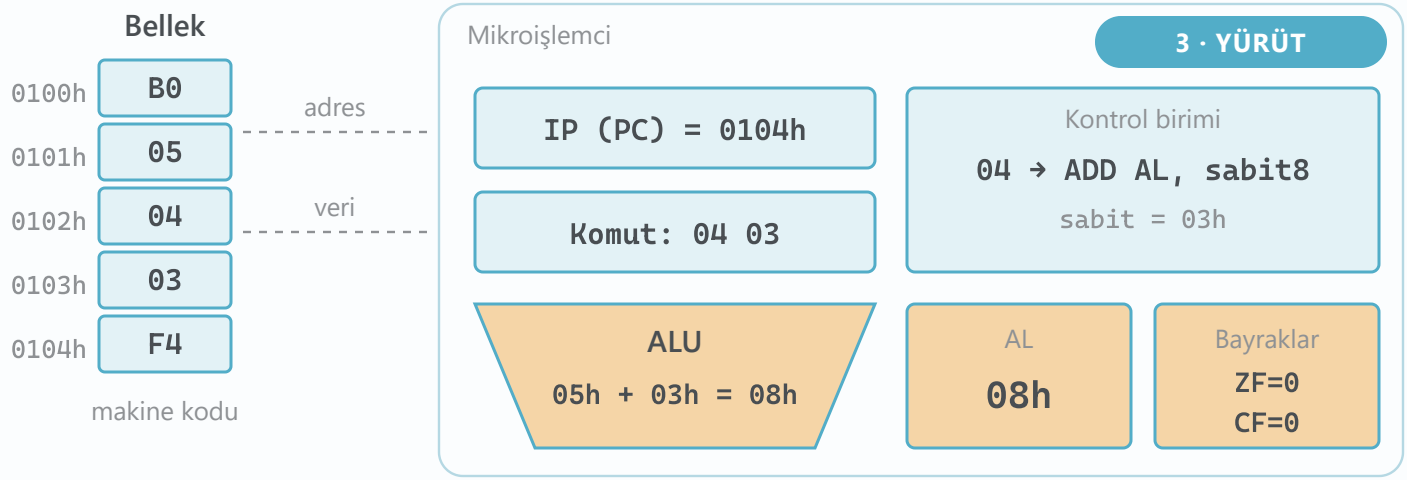
Döngü yeniden başlar: **04 03** getirilir, IP → **0104h**.

Komut Döngüsü: `MOV AL,5` · `ADD AL,3` · `HLT`



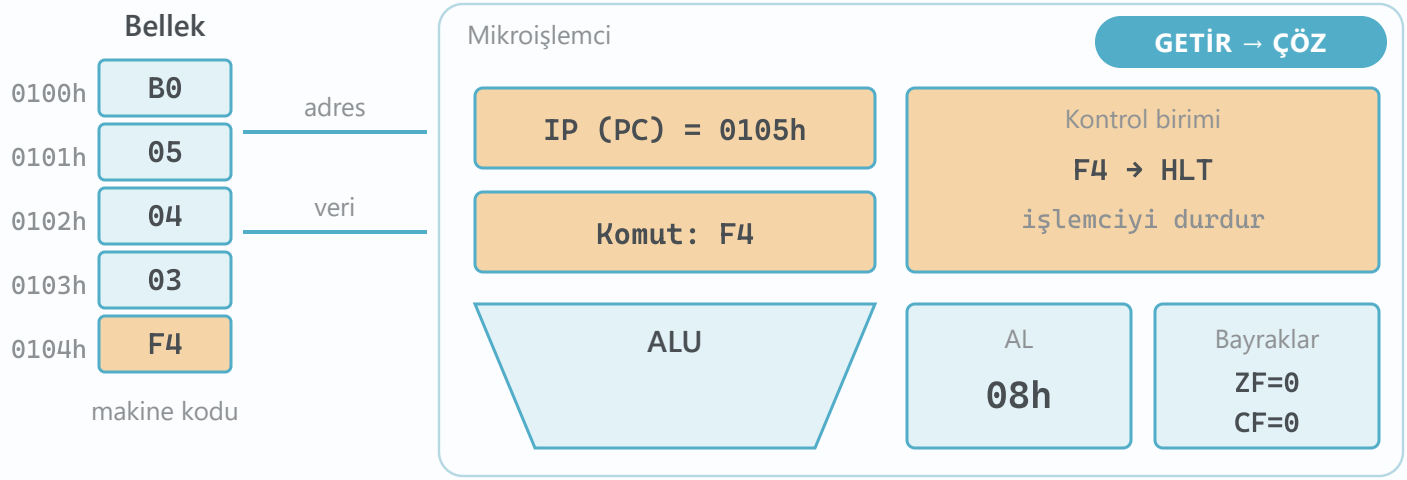
04 = "AL'ye 8 bitlik sabit ekle". İşlenenler: AL (05h) ve 03h.

Komut Döngüsü: `MOV AL,5` · `ADD AL,3` · `HLT`



ALU toplar: 05h + 03h = **08h** → AL. Sonuç sıfır değil (**ZF=0**), elde yok (**CF=0**).

Komut Döngüsü: `MOV AL,5` · `ADD AL,3` · `HLT`



F4 = HLT getirilip çözülür; işlemci bir kesme gelene kadar durur. Her komut için aynı döngü tekrarlanır.

TEMEL KAVRAMLAR

Temel İşlevler

Aritmetik ve mantıksal işlemler

Toplama, çıkarma, karşılaştırma, AND/OR/XOR, kaydırma. **ALU** (Arithmetic Logic Unit) tarafından yapılır. Yeni işlemciler kayan noktalı sayılarla da çalışır (FPU).

```
ADD AX, BX
CMP AL, 10
AND AL, 0Fh
```

Veri taşıma

Veri yazmaçlar, bellek ve G/Ç arasında bir yerden başka bir yere kopyalanır. Programlardaki en sık işlemdir.

```
MOV AX, [1234h]
MOV [BX], AL
IN AL, 60h
```

Program sayacı (PC)

Sıradaki komutun adresini tutar. PC'ye yeni değer yazılarak **atlanır**; koşullu atlamalarla **karar** verilir. 8086'da adı **IP**.

```
JMP etiket
JZ sifirsa
CALL alt_program
```

Her program, bu üç işlevin (hesapla · taşı · sırayı değiştir) birleşiminden oluşur.

Kod İzleme: IP Nasıl Değişir?

```
▶ 0100  mov  cx, 3      ; sayaç = 3
  0103  mov  ax, 0      ; toplam = 0
  0106  dongu: add ax, cx
  0108  loop dongu      ; CX--, CX≠0 ise atla
  010A  hlt             ; dur
```

`mov cx, 3` 3 bayttır; çalıştıktan sonra IP sıradaki komutu (0103h) gösterir.

Adım 1 / 9

DEĞİŞKENLER

`IP = 0103h` `CX = 0003h` `AX = ????`

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Kod İzleme: IP Nasıl Değişir?

```
1 0100  mov  cx, 3      ; sayaç = 3
▶ 0103  mov  ax, 0      ; toplam = 0
3 0106  dongu: add ax, cx
4 0108  loop dongu      ; CX--, CX≠0 ise atla
5 010A  hlt             ; dur
```

Toplam sıfırlanır. IP yine komut uzunluğu kadar ilerler → 0106h.

Adım 2 / 9

DEĞİŞKENLER

IP = 0106h | CX = 0003h | AX = 0000h

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Kod İzleme: IP Nasıl Değişir?

```
1 0100  mov  cx, 3      ; sayaç = 3
2 0103  mov  ax, 0      ; toplam = 0
▶ 0106  dongu: add ax, cx
4 0108  loop dongu      ; CX--, CX≠0 ise atla
5 010A  hlt             ; dur
```

AX = 0 + 3 = 3. `add ax, cx` 2 bayttır → IP = 0108h.

Adım 3 / 9

DEĞİŞKENLER

IP = 0108h CX = 0003h AX = 0003h

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Kod İzleme: IP Nasıl Değişir?

```
1 0100  mov  cx, 3      ; sayaç = 3
2 0103  mov  ax, 0      ; toplam = 0
3 0106  dongu: add ax, cx
▶ 0108  loop dongu      ; CX--, CX≠0 ise atla
5 010A  hlt             ; dur
```

`loop` : CX = CX − 1 = 2 ≠ 0 → **atla**. IP'ye 0106h yazılır; sıralı akış bozulur.

Adım 4 / 9

DEĞİŞKENLER

`IP = 0106h` `CX = 0002h` `AX = 0003h`

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Kod İzleme: IP Nasıl Değişir?

```
1 0100  mov  cx, 3      ; sayaç = 3
2 0103  mov  ax, 0      ; toplam = 0
▶ 0106  dongu: add ax, cx
4 0108  loop dongu      ; CX--, CX≠0 ise atla
5 010A  hlt             ; dur
```

AX = 3 + 2 = 5.

Adım 5 / 9

DEĞİŞKENLER

IP = 0108h CX = 0002h AX = 0005h

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Kod İzleme: IP Nasıl Değişir?

```
1 0100  mov  cx, 3      ; sayaç = 3
2 0103  mov  ax, 0      ; toplam = 0
3 0106  dongu: add ax, cx
▶ 0108  loop dongu      ; CX--, CX≠0 ise atla
5 010A  hlt             ; dur
```

CX = 1 ≠ 0 → yine 0106h'ye atlanır.

Adım 6 / 9

DEĞİŞKENLER

IP = 0106h CX = 0001h AX = 0005h

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Kod İzleme: IP Nasıl Değişir?

```
1 0100  mov  cx, 3      ; sayaç = 3
2 0103  mov  ax, 0      ; toplam = 0
▶ 0106  dongu: add ax, cx
4 0108  loop dongu      ; CX--, CX≠0 ise atla
5 010A  hlt             ; dur
```

AX = 5 + 1 = 6.

Adım 7 / 9

DEĞİŞKENLER

IP = 0108h | CX = 0001h | AX = 0006h

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Kod İzleme: IP Nasıl Değişir?

```
1 0100  mov  cx, 3      ; sayaç = 3
2 0103  mov  ax, 0      ; toplam = 0
3 0106  dongu: add ax, cx
▶ 0108  loop dongu      ; CX--, CX≠0 ise atla
5 010A  hlt             ; dur
```

CX = 0 → atlama **yok**; IP sıradaki komuta (010Ah) geçer.

Adım 8 / 9

DEĞİŞKENLER

IP = **010Ah** CX = **0000h** AX = **0006h**

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Kod İzleme: IP Nasıl Değişir?

```
1 0100  mov  cx, 3      ; sayaç = 3
2 0103  mov  ax, 0      ; toplam = 0
3 0106  dongu: add ax, cx
4 0108  loop dongu      ; CX--, CX≠0 ise atla
▶ 010A  hlt             ; dur
```

`hlt` : işlemci durur. Sonuç AX = 3 + 2 + 1 = **6**.

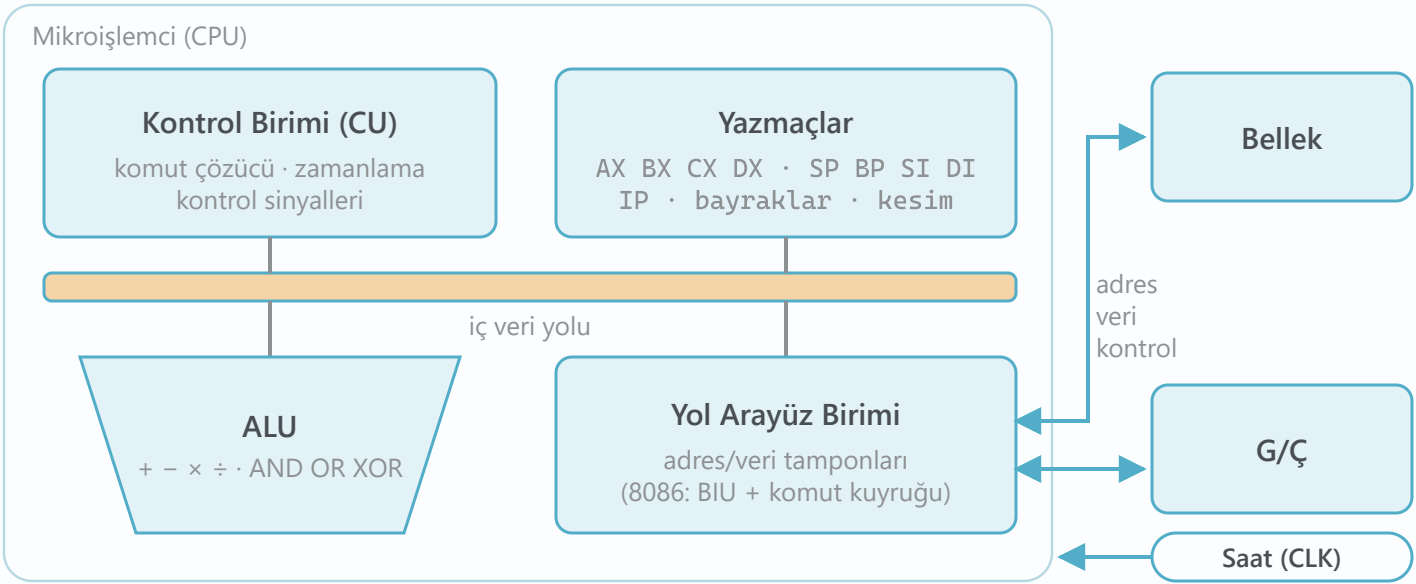
Adım 9 / 9

DEĞİŞKENLER

`IP = 010Bh` `CX = 0000h` `AX = 0006h`

Soldaki sayılar komutların bellek adresleridir (bağıl konum, hex). `loop` bayrakları değiştirmez.

Mikroişlemci Yapısı



Kontrol birimi komutları çözer ve diğer birimlere **ne zaman ne yapacaklarını** söyler; ALU hesaplar; yazmaçlar en hızlı geçici depodur; yol arayüzü dış dünyayla konuşur.

Yapı Bileşenleri

ALU

İkili sayılar üzerinde aritmetik (+, −) ve mantıksal (AND, OR, NOT, XOR) işlemler ile karşılaştırma ve kaydırma yapar. Sonuca göre **bayrakları** (sıfır, elde, işaret, taşma) ayarlar.

Kontrol birimi

Komutu çözer, hangi yazmacın hangi yola bağlanacağını, ALU'nun hangi işlemi yapacağını ve bellek okuma/yazma sinyallerini zamanlar.

Yazmaçlar (register)

İşlemci içindeki küçük ve çok hızlı bellek gözleri. Genel amaçlı (veri), adres/işaretçi, durum (bayrak) ve program sayacı yazmaçları vardır.

Yol arayüzü

Adres, veri ve kontrol yollarını sürer. 8086'da **BIU** komutları önden getirip 6 baytlık bir **kuyrukta** bekletir.

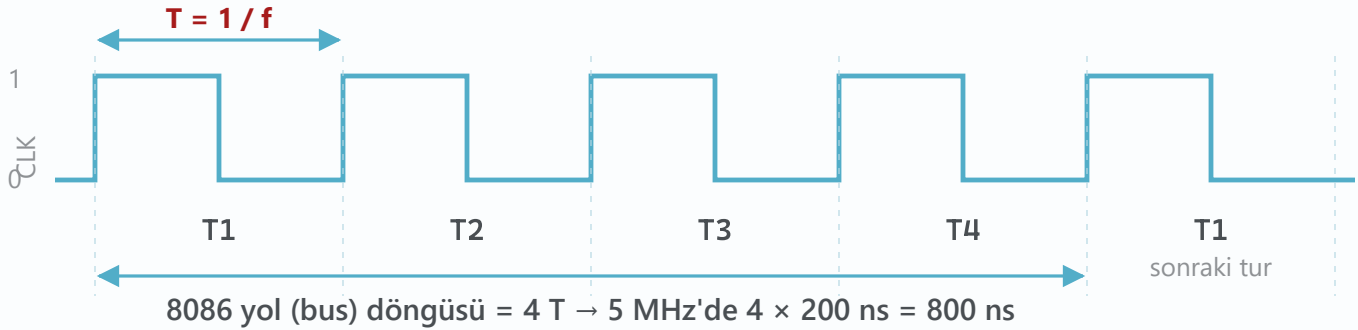
Saat ve zamanlama

Tüm işlemler saat sinyalinin vuruşlarıyla eşzamanlı ilerler. Bir vuruşun süresi **T = 1/f**.

Önbellek, FPU, çekirdekler

Modern işlemcilerde aynı yongada önbellek, kayan nokta birimi, birden çok çekirdek ve grafik birimi de bulunur.

Saat Sinyali ve Periyot



- **Saat frekansı** f: saniyedeki vuruş (tur) sayısı, Hz.
- **Periyot** T = 1 / f: bir vuruşun süresi.
- Bir komut, birkaç saat turu sürebilir; 8086'da bir bellek erişimi en az **4 T** (T1–T4) sürer.

f	T = 1/f	4 T
5 MHz (8086)	200 ns	800 ns
10 MHz (8086-1)	100 ns	400 ns
3 GHz	0,333 ns	1,33 ns

Hesaplayıcı: Saat Hızı, Periyot ve Çalışma Süresi

8086 · 5 MHz | 80386 · 33 MHz | Pentium · 66 MHz | Core i7 · 3,33 GHz

Saat frekansı f (MHz)	Komut başına tur (CPI)	Komut sayısı N (milyon)
5	10	1

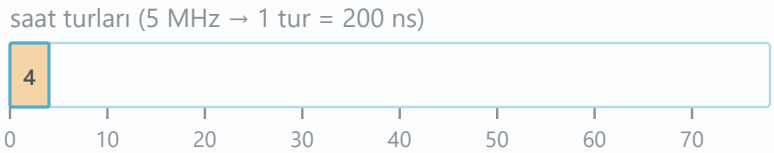
Periyot T	8086 yol döngüsü (4 T)	Hız (MIPS = f / CPI)	Çalışma süresi
200 ns	800 ns	0,5	2 s

T = 1 / 5 MHz = **200 ns** · süre = N × CPI / f = $1 \cdot 10^6 \times 10 / 5 \cdot 10^6$ Hz = **2 s**

CPI değerleri kabaca örnektir: 8086 komutları ortalama ~10 tur sürerken modern işlemciler tur başına birden çok komut bitirebilir (CPI < 1).

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
▶ mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
4 dongu: add ax, cx  ; 3 tur
5 loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **`4 tur × 200 ns = 0,8 µs`**

bu komut: MOV sabit → 4 tur **`CX = 4 · AX = 0`**

Her komut belirli sayıda **saat turu** sürer. 5 MHz'lik 8086'da bir tur T = 1/f = **200 ns**. `mov cx, 4` : 4 tur.

Adım 1 / 11

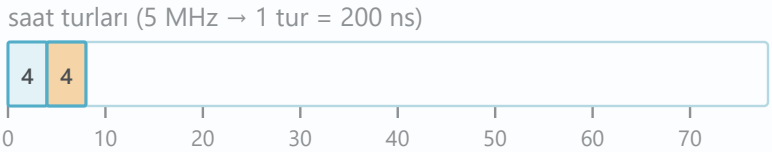
DEĞİŞKENLER

`CX = 4` `AX = 0` `tur = 4` `süre = 0,8 µs`

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
▶ mov ax, 0          ; 4 tur
4 dongu: add ax, cx  ; 3 tur
5 loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **8 tur × 200 ns = 1,6 µs**

bu komut: MOV sabit → 4 tur — CX = 4 · AX = 0

`mov ax, 0` : yine 4 tur. Toplam 8 tur = 1,6 µs.

Adım 2 / 11

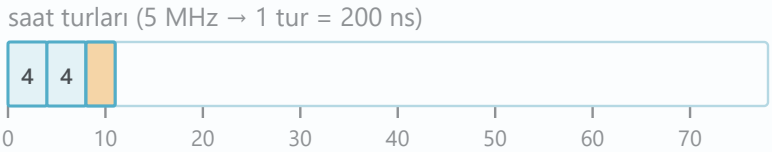
DEĞİŞKENLER

CX = 4 | AX = 0 | tur = 8 | süre = 1,6 µs

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
▶ dongu: add ax, cx  ; 3 tur
5 loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **11 tur × 200 ns = 2,2 µs**

bu komut: ADD yazmaç → 3 tur `CX = 4 · AX = 4`

`add ax, cx` : yazmaçtan yazmaca toplama 3 tur; bellek erişimi olmadığı için kısa. AX = 4.

Adım 3 / 11

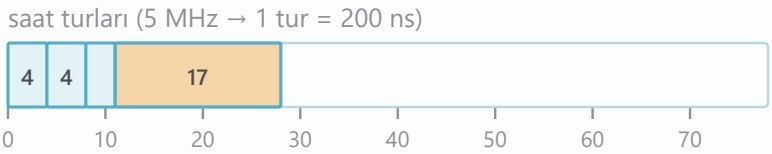
DEĞİŞKENLER

`CX = 4` `AX = 4` `tur = 11` `süre = 2,2 µs`

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
4 dongu: add ax, cx  ; 3 tur
▶ loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **28 tur × 200 ns = 5,6 µs**

bu komut: LOOP → 17 tur CX = 3 · AX = 4

`loop` : CX = 3 ≠ 0 → atlar. Atlayan LOOP **17 tur** sürer: kuyruk boşaltılır, hedeften yeniden getirilir.

Adım 4 / 11

DEĞİŞKENLER

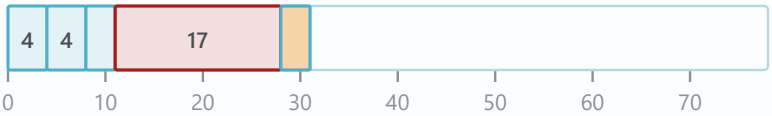
CX = 3 | AX = 4 | tur = 28 | süre = 5,6 µs

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
▶ dongu: add ax, cx  ; 3 tur
5 loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```

saat turları (5 MHz → 1 tur = 200 ns)



şimdiye kadar: **31 tur × 200 ns = 6,2 µs**

bu komut: ADD yazmaç → 3 tur CX = 3 · AX = 7

`add ax, cx` : yazmaçtan yazmaca toplama 3 tur; bellek erişimi olmadığı için kısa. AX = 7.

Adım 5 / 11

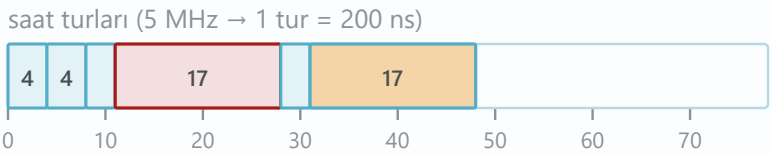
DEĞİŞKENLER

CX = 3 | AX = 7 | tur = 31 | süre = 6,2 µs

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
4 dongu: add ax, cx  ; 3 tur
▶ loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **48 tur × 200 ns = 9,6 µs**

bu komut: LOOP → 17 tur `CX = 2 · AX = 7`

`loop` : CX = 2 ≠ 0 → atlar. Atlayan LOOP **17 tur** sürer: kuyruk boşaltılır, hedeften yeniden getirilir.

Adım 6 / 11

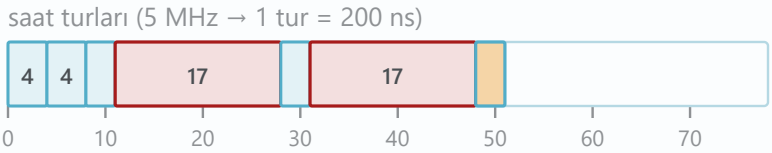
DEĞİŞKENLER

`CX = 2` `AX = 7` `tur = 48` `süre = 9,6 µs`

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
▶ dongu: add ax, cx  ; 3 tur
5 loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **51 tur × 200 ns = 10,2 µs**

bu komut: ADD yazmaç → 3 tur

CX = 2 · AX = 9

`add ax, cx` : yazmaçtan yazmaca toplama 3 tur; bellek erişimi olmadığı için kısa. AX = 9.

Adım 7 / 11

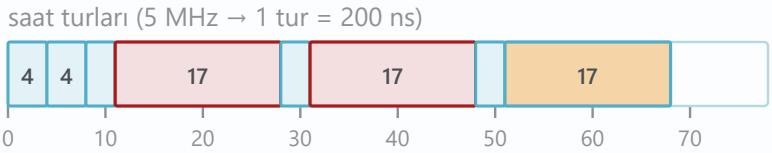
DEĞİŞKENLER

CX = 2 | AX = 9 | tur = 51 | süre = 10,2 µs

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
4 dongu: add ax, cx  ; 3 tur
▶ loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **68 tur × 200 ns = 13,6 µs**

bu komut: LOOP → 17 tur CX = 1 · AX = 9

`loop` : CX = 1 ≠ 0 → atlar. Atlayan LOOP **17 tur** sürer: kuyruk boşaltılır, hedeften yeniden getirilir.

Adım 8 / 11

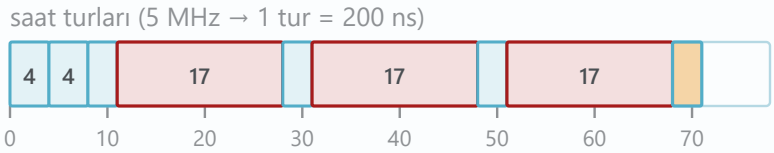
DEĞİŞKENLER

CX = 1 | AX = 9 | tur = 68 | süre = 13,6 µs

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
▶ dongu: add ax, cx  ; 3 tur
5 loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **71 tur × 200 ns = 14,2 µs**

bu komut: ADD yazmaç → 3 tur **CX = 1 · AX = 10**

`add ax, cx` : yazmaçtan yazmaca toplama 3 tur; bellek erişimi olmadığı için kısa. AX = 10.

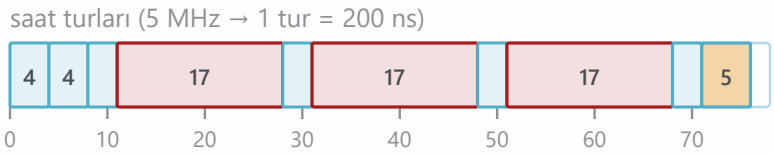
DEĞİŞKENLER

CX = **1** | AX = **10** | tur = **71** | süre = **14,2 µs**

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
4 dongu: add ax, cx  ; 3 tur
▶ loop dongu         ; 17 / 5 tur
6 hlt                ; 2 tur
```



şimdiye kadar: **76 tur × 200 ns = 15,2 µs**

bu komut: LOOP → 5 tur

CX = 0 · AX = 10

CX = 0 → atlamaz; atlamayan LOOP yalnızca **5 tur**.

Adım 10 / 11

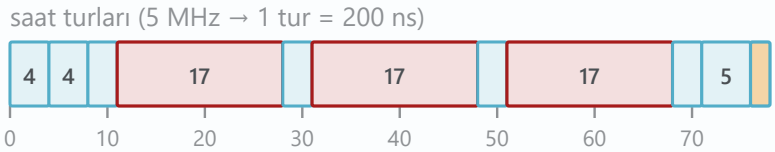
DEĞİŞKENLER

CX = 0 | AX = 10 | tur = 76 | süre = 15,2 µs

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

Kod İzleme: Bir Döngü Kaç Saat Turu Sürer?

```
1 org 100h
2 mov cx, 4          ; 4 tur
3 mov ax, 0          ; 4 tur
4 dongu: add ax, cx  ; 3 tur
5 loop dongu         ; 17 / 5 tur
▶ hlt                ; 2 tur
```



şimdiye kadar: **78 tur × 200 ns = 15,6 µs**

bu komut: HLT → 2 tur

CX = 0 · AX = 10

`hlt` : 2 tur. Toplam **78 tur × 200 ns = 15,6 µs**; turların yarıdan fazlası (51) atlayan LOOP'larda harcandı.

Adım 11 / 11

DEĞİŞKENLER

CX = 0 | AX = 10 | tur = 78 | süre = 15,6 µs

Tur sayıları 8086 veri sayfasındaki en iyi durum değerleridir (kuyruk dolu, bekleme durumu yok). Süre = tur sayısı × T.

MİKROİŞLEMCİ YAPISI

16, 32 ve 64 Bitlik İşlemcilerin Saat Hızları

16-bit

- 8086: 5 · 8 · 10 MHz
- 8088: 5 · 8 MHz (IBM PC'de 4,77 MHz)
- 80186/80188: 6 MHz'den başlayarak
- 80286: 6 · 8 · 10 · 12,5 MHz

32-bit

- 80386: 16 – 33 MHz
- 80486: 16 – 100 MHz (DX2/DX4 saat çarpanı)
- Pentium: 60 · 66 MHz ile başladı

64-bit

- Core 2: 1,2 – 3 GHz
- Core i7 (1. nesil): 2,66 – 3,33 GHz
- Core i5 (1. nesil): 2,4 – 3,6 GHz
- Core i3 (1. nesil): 2,93 – 3,33 GHz

30 yılda saat hızı yaklaşık **600 kat** arttı (5 MHz → ~3 GHz). 2005 civarında güç ve ısı sınırına dayanıldı; o tarihten sonra performans artışı daha çok **çok çekirdek** ve tur başına daha fazla komutla sağlandı.

4,77 MHz, IBM PC'deki 8088'in saat hızıdır.

“n-bit İşlemci” Ne Demek?

İşlemcinin bir seferde işleyebildiği sözcük (word) genişliği: yazmaç ve ALU boyutu, çoğunlukla veri yolu genişliği. n bitle 0 … 2^n − 1 aralığı temsil edilir.

Sözcük	İşaretsiz aralık
8 bit	0 … 255
16 bit	0 … 65.535
32 bit	0 … 4.294.967.295
64 bit	0 … ≈ 1,8 × 10^{19}

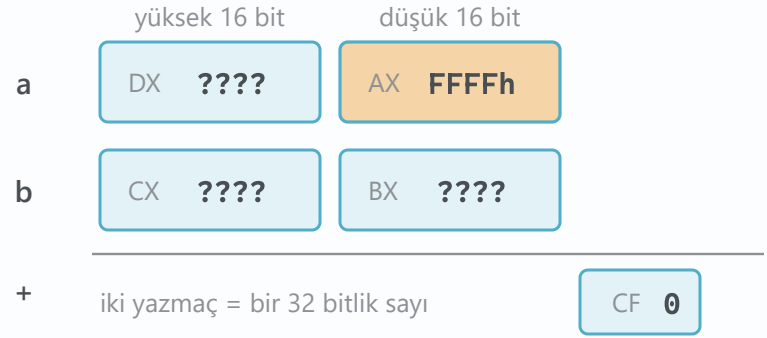
Adres yolu genişliği ise erişilebilen bellek miktarını belirler: n hat → 2^n bayt.

Adres yolu	Bellek	Örnek
16 bit	64 KB	8080, 8085
20 bit	1 MB	8086, 8088
24 bit	16 MB	80286
32 bit	4 GB	80386, 80486
36 bit	64 GB	Pentium Pro (PAE)

8086 **16 bitlik** bir işlemcidir ama **20 bitlik** adres yoluna sahiptir. 20 bitlik adres, iki 16 bitlik değerin (kesim ve bağıl konum) birleştirilmesiyle üretilir.

16 Bitlik İşlemcide 32 Bitlik Toplama

```
1 org 100h
▶ mov ax, 0FFFFh ; a: düşük yarı
3 mov dx, 0001h  ; a: yüksek yarı
4 mov bx, 0002h  ; b: düşük yarı
5 mov cx, 0003h  ; b: yüksek yarı
6 add ax, bx     ; düşük yarılar
7 adc dx, cx     ; yüksek + CF
8 ret
```



a = 0001FFFFh (131.071) 16 bitlik bir yazmaca sığmaz; iki yarıya bölünür. Düşük yarı **AX = FFFFh**.

Adım 1 / 7

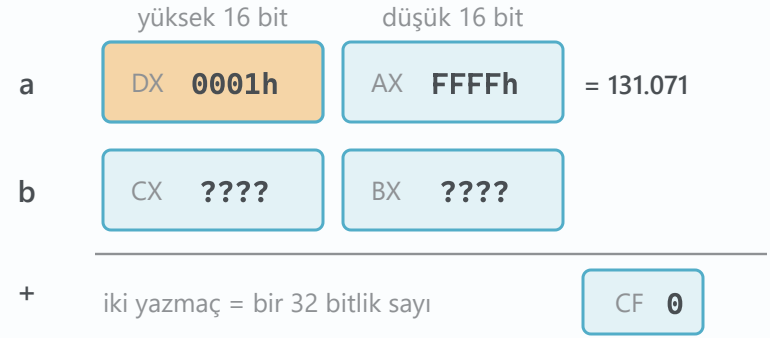
DEĞİŞKENLER

AX = FFFFh | DX = 0000h | BX = 0000h | CX = 0000h | CF = 0

`adc` (add with carry), önceki toplamanın eldesini (CF) de ekler.

16 Bitlik İşlemcide 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0FFFFh ; a: düşük yarı
▶ mov dx, 0001h  ; a: yüksek yarı
4 mov bx, 0002h  ; b: düşük yarı
5 mov cx, 0003h  ; b: yüksek yarı
6 add ax, bx     ; düşük yarılar
7 adc dx, cx     ; yüksek + CF
8 ret
```



Yüksek yarı **DX = 0001h**. 32 bitlik a artık **DX:AX** çiftinde.

Adım 2 / 7

DEĞİŞKENLER

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

`adc` (add with carry), önceki toplamanın eldesini (CF) de ekler.

16 Bitlik İşlemcide 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0FFFFh ; a: düşük yarı
3 mov dx, 0001h  ; a: yüksek yarı
▶ mov bx, 0002h  ; b: düşük yarı
5 mov cx, 0003h  ; b: yüksek yarı
6 add ax, bx     ; düşük yarılar
7 adc dx, cx     ; yüksek + CF
8 ret
```



b = 00030002h (196.610): düşük yarı **BX = 0002h**.

Adım 3 / 7

DEĞİŞKENLER

AX = FFFFh | DX = 0001h | BX = 0002h | CX = 0000h | CF = 0

`adc` (add with carry), önceki toplamanın eldesini (CF) de ekler.

16 Bitlik İşlemcide 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0FFFFh ; a: düşük yarı
3 mov dx, 0001h  ; a: yüksek yarı
4 mov bx, 0002h  ; b: düşük yarı
▶ mov cx, 0003h  ; b: yüksek yarı
6 add ax, bx     ; düşük yarılar
7 adc dx, cx     ; yüksek + CF
8 ret
```



Yüksek yarı **CX = 0003h**; b, **CX:BX** çiftinde.

Adım 4 / 7

DEĞİŞKENLER

AX = FFFFh | DX = 0001h | BX = 0002h | CX = 0003h | CF = 0

`adc` (add with carry), önceki toplamanın eldesini (CF) de ekler.

16 Bitlik İşlemcide 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0FFFFh ; a: düşük yarı
3 mov dx, 0001h  ; a: yüksek yarı
4 mov bx, 0002h  ; b: düşük yarı
5 mov cx, 0003h  ; b: yüksek yarı
▶ add ax, bx     ; düşük yarılar
7 adc dx, cx     ; yüksek + CF
8 ret
```



`add ax, bx` : FFFFh + 0002h = 1 0001h. 17. bit 16 bitlik AX'e sığmaz: AX = **0001h**, taşan elde **CF = 1**.

Adım 5 / 7

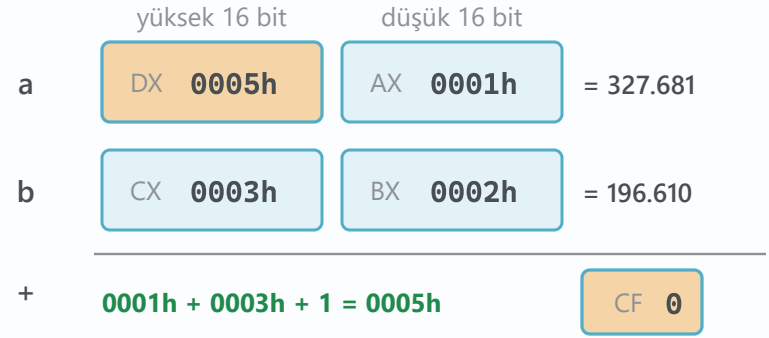
DEĞİŞKENLER

AX = 0001h | DX = 0001h | BX = 0002h | CX = 0003h | CF = 1

`adc` (add with carry), önceki toplamanın eldesini (CF) de ekler.

16 Bitlik İşlemcide 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0FFFFh ; a: düşük yarı
3 mov dx, 0001h  ; a: yüksek yarı
4 mov bx, 0002h  ; b: düşük yarı
5 mov cx, 0003h  ; b: yüksek yarı
6 add ax, bx     ; düşük yarılar
▶ adc dx, cx     ; yüksek + CF
8 ret
```



`adc dx, cx` : 0001h + 0003h + CF(1) = **0005h**. Sonuç DX:AX = 00050001h = **327.681**.

Adım 6 / 7

DEĞİŞKENLER

AX = 0001h | DX = 0005h | BX = 0002h | CX = 0003h | CF = 0

`adc` (add with carry), önceki toplamanın eldesini (CF) de ekler.

16 Bitlik İşlemcide 32 Bitlik Toplama

```
1 org 100h
2 mov ax, 0FFFFh ; a: düşük yarı
3 mov dx, 0001h  ; a: yüksek yarı
4 mov bx, 0002h  ; b: düşük yarı
5 mov cx, 0003h  ; b: yüksek yarı
6 add ax, bx     ; düşük yarılar
7 adc dx, cx     ; yüksek + CF
▶ ret
```



16 bitlik işlemci 32 bitlik toplamayı **iki komutta** yaptı; 32 bitlik bir işlemci (80386) aynı işi tek `add` ile yapar.

Adım 7 / 7

DEĞİŞKENLER

AX = 0001h | DX = 0005h | BX = 0002h | CX = 0003h | CF = 0

`adc` (add with carry), önceki toplamanın eldesini (CF) de ekler.

İşlemci Türleri: Genel Bakış



CISC
Complex Instruction Set Computer · Karmaşık Komut Kümeli Bilgisayar

RISC
Reduced Instruction Set Computer · Azaltılmış Komut Kümeli Bilgisayar

EPIC
Explicitly Parallel Instruction Computing · Açıkça Paralel Komut İşleme

Karmaşık Komut Kümeli Bilgisayar (CISC)

- Karmaşık işlemleri **tek bir komutla** yapabilir.
- Bir komut; bellekten yükleme, aritmetik işlem ve belleğe kaydetme gibi **birden çok düşük düzeyli işlem** içerebilir.
- Komutlarda birden çok **adresleme modu** ve doğrudan bellek işlenenleri bulunur.
- Komut uzunlukları **değişkendir** (8086'da 1–6 bayt); çözme karmaşıktır.
- Görece **az sayıda yazmaç** kullanır.
- Çoğunlukla **mikroprogramlı** kontrol birimi.

Örnekler

- Intel 8086, 80386, 80486
- Pentium, Pentium Pro, II, III
- Motorola 68000, 68020, 68040
- DEC VAX, IBM System/360

```
ADD [BX], AX    ; bellek = bellek + AX
REP MOVSB       ; CX bayt kopyala
```

Tek komut: oku + topla + yaz

Azaltılmış Komut Kümeli Bilgisayar (RISC)

- Az sayıda, **basit ve sabit uzunluklu** komut (ör. hepsi 32 bit).
- Optimize komutlar ve **boru hattı** (pipeline) ile hızlı yürütme.
- Hedef: ideal durumda **her saat turunda bir komut tamamlamak** (CPI ≈ 1).
- Belleğe yalnızca **yükle/sakla** (load/store) komutları erişir; işlemler yazmaçlar arasında yapılır.
- Bellek erişimini azaltmak için **çok sayıda yazmaç** (32 ve üzeri).
- Donanımsal (hardwired) kontrol birimi; karmaşıklık derleyiciye kayar.

Örnekler

- IBM RS/6000 (POWER)
- Motorola MC88100
- DEC Alpha 21064, 21164, 21264
- MIPS, SPARC
- **ARM** (telefonlar, Apple M serisi)
- **RISC-V** (açık kaynak komut kümesi)

Bir komut tek turda *tamamlanmaz*; birkaç aşamadan geçer. Boru hattı sayesinde her turda *bir komut biter* (verim/throughput).

Boru Hattı (Pipeline): Aşamaların Üst Üste Binmesi



Tur 1: K1 **getirilir** (IF). Diğer aşamalar boş.

Adım 1 / 8

Fabrikadaki montaj bandı gibi: her aşama ayrı bir donanım birimidir ve her turda bir sonraki komuta geçer. Bağımlılıklar ve dallanmalar boru hattında **bekleme (stall)** doğurabilir.

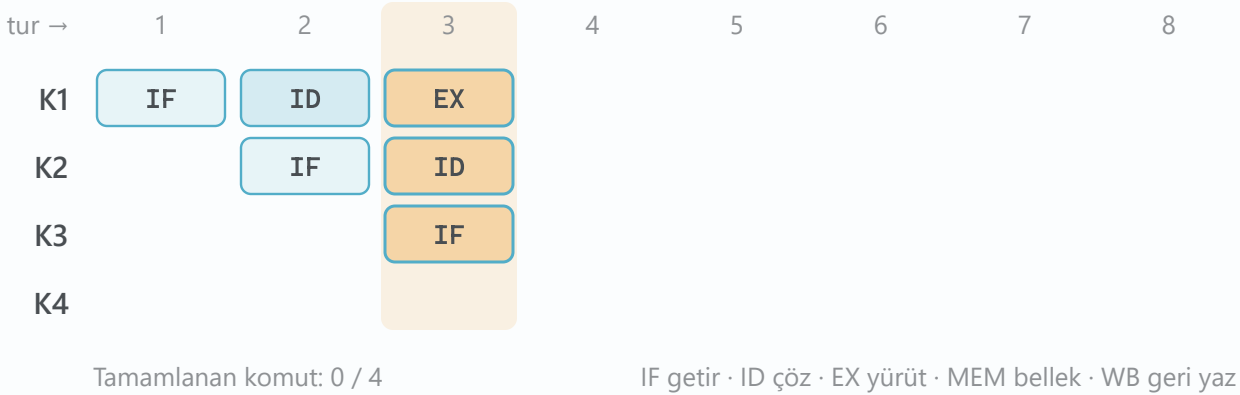
Boru Hattı (Pipeline): Aşamaların Üst Üste Binmesi



Tur 2: K1 **çözülürken** (ID) K2 getirilir. İki komut aynı anda farklı aşamalarda.

Fabrikadaki montaj bandı gibi: her aşama ayrı bir donanım birimidir ve her turda bir sonraki komuta geçer. Bağımlılıklar ve dallanmalar boru hattında **bekleme (stall)** doğurabilir.

Boru Hattı (Pipeline): Aşamaların Üst Üste Binmesi

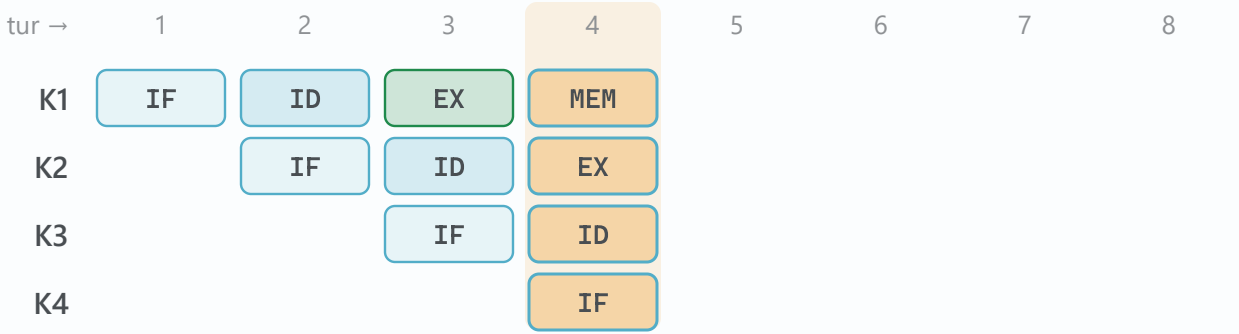


Tur 3: üç komut üç farklı aşamada çalışıyor.

Adım 3 / 8

Fabrikadaki montaj bandı gibi: her aşama ayrı bir donanım birimidir ve her turda bir sonraki komuta geçer. Bağımlılıklar ve dallanmalar boru hattında **bekleme (stall)** doğurabilir.

Boru Hattı (Pipeline): Aşamaların Üst Üste Binmesi

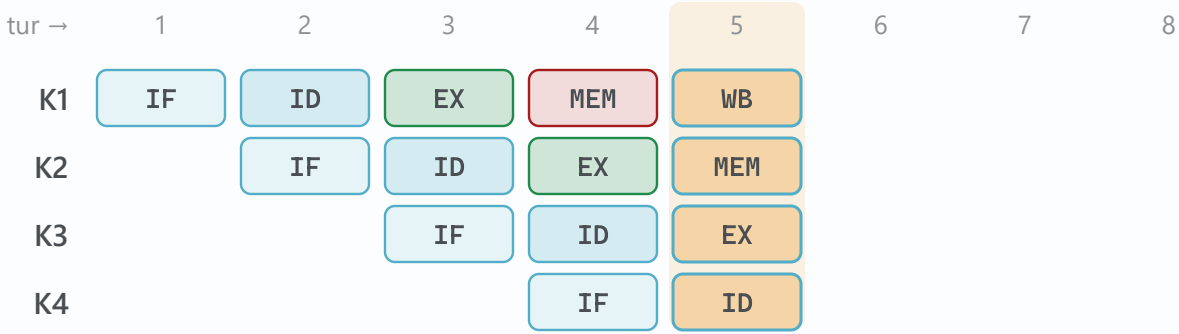


Tur 4: dört komut birden boru hattında.

Adım 4 / 8

Fabrikadaki montaj bandı gibi: her aşama ayrı bir donanım birimidir ve her turda bir sonraki komuta geçer. Bağımlılıklar ve dallanmalar boru hattında **bekleme (stall)** doğurabilir.

Boru Hattı (Pipeline): Aşamaların Üst Üste Binmesi



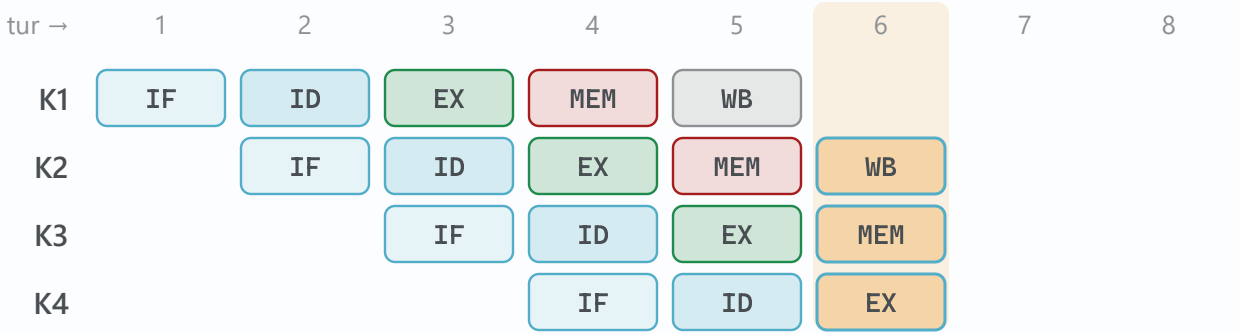
Tamamlanan komut: 1 / 4

IF getir · ID çöz · EX yürüt · MEM bellek · WB geri yaz

Tur 5: boru hattı **dolu** sayılır; K1 tamamlanır (WB). Bundan sonra her turda bir komut biter.

Fabrikadaki montaj bandı gibi: her aşama ayrı bir donanım birimidir ve her turda bir sonraki komuta geçer. Bağımlılıklar ve dallanmalar boru hattında **bekleme (stall)** doğurabilir.

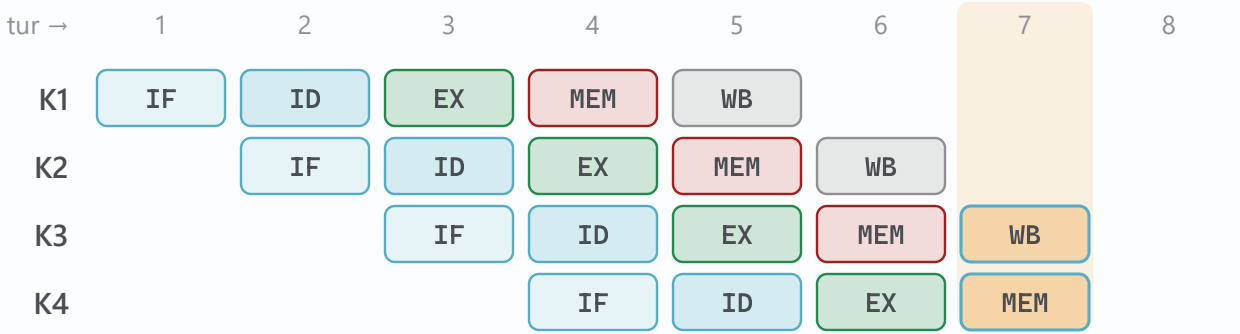
Boru Hattı (Pipeline): Aşamaların Üst Üste Binmesi



Tur 6: K2 tamamlanır.

Fabrikadaki montaj bandı gibi: her aşama ayrı bir donanım birimidir ve her turda bir sonraki komuta geçer. Bağımlılıklar ve dallanmalar boru hattında **bekleme (stall)** doğurabilir.

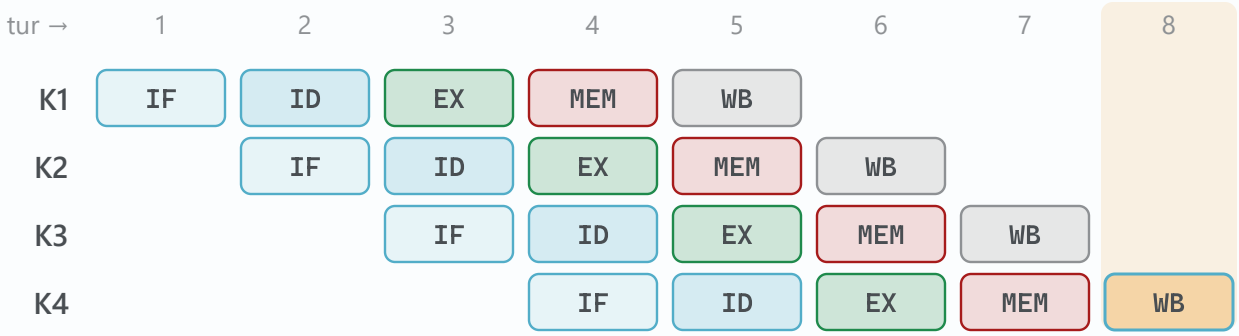
Boru Hattı (Pipeline): Aşamaların Üst Üste Binmesi



Tur 7: K3 tamamlanır.

Fabrikadaki montaj bandı gibi: her aşama ayrı bir donanım birimidir ve her turda bir sonraki komuta geçer. Bağımlılıklar ve dallanmalar boru hattında **bekleme (stall)** doğurabilir.

Boru Hattı (Pipeline): Aşamaların Üst Üste Binmesi



Tamamlanan komut: 4 / 4

IF getir · ID çöz · EX yürüt · MEM bellek · WB geri yaz

Tur 8: K4 tamamlanır. 4 komut **k + n − 1 = 5 + 4 − 1 = 8** turda biter; boru hattısız **4 × 5 = 20** tur gerekirdi.

Fabrikadaki montaj bandı gibi: her aşama ayrı bir donanım birimidir ve her turda bir sonraki komuta geçer. Bağımlılıklar ve dallanmalar boru hattında **bekleme (stall)** doğurabilir.

İŞLEMCİ TÜRLERİ

Aynı İş, İki Yaklaşım

Görev: bellekteki bir sayıya bir yazmacın değerini eklemek (`bellek[adr] += x`).

CISC (8086)

```
ADD [BX], AX      ; 1 komut
                  ; oku, topla, yaz
```

- Kod kısa, bellekte az yer kaplar.
- Komut birçok tur sürer.

RISC (ARM)

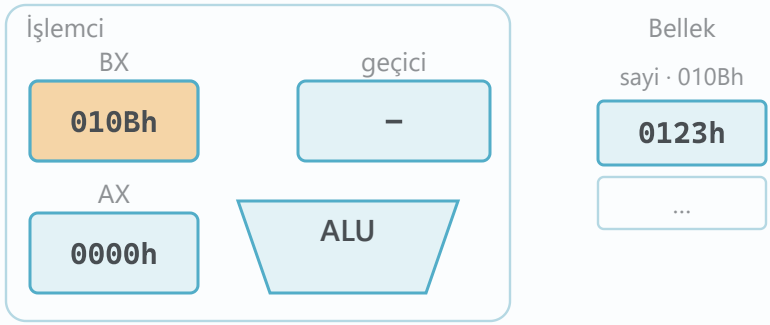
```
LDR r2, [r1]      @ yükle
ADD r2, r2, r0    @ topla
STR r2, [r1]      @ sakla
```

- 3 basit komut; her biri boru hattında akıcı.
- Değer yazmaçta kalırsa sonraki işlemler belleğe gitmez.

Modern x86 işlemciler (Pentium Pro'dan beri) karmaşık komutları içeride RISC benzeri **mikro-işlemlere (µop)** çevirir: dışı CISC, içi RISC.

Kod İzleme: Tek CISC Komutu, Üç İş

```
1 org 100h
▶ mov bx, offset sayi ; adres
3 mov ax, 7           ; değer
4 add [bx], ax        ; oku, topla, yaz
5 add [bx], ax        ; tekrar
6 ret
7 sayi dw 0123h
```



BX = 010Bh: `sayi` değişkeninin adresi. Bellekte sayi = **0123h**.

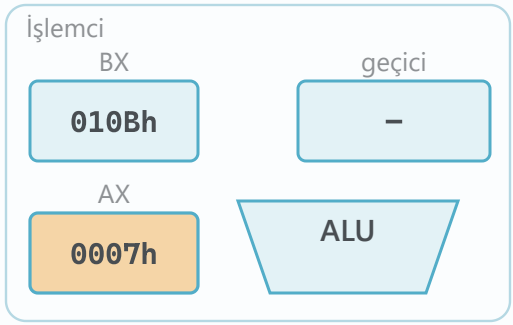
Adım 1 / 7

DEĞİŞKENLER

`BX = 010Bh` `AX = 0000h` `geçici = –` `sayi = 0123h`

Kod İzleme: Tek CISC Komutu, Üç İş

```
1 org 100h
2 mov bx, offset sayi ; adres
▶ mov ax, 7           ; değer
4 add [bx], ax        ; oku, topla,
  yaz
5 add [bx], ax        ; tekrar
6 ret
7 sayi dw 0123h
```



AX = **0007h**, eklenecek değer.

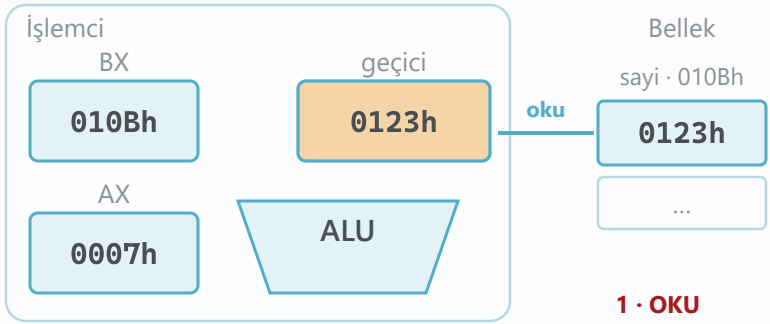
Adım 2 / 7

DEĞİŞKENLER

`BX = 010Bh` `AX = 0007h` `geçici = –` `sayi = 0123h`

Kod İzleme: Tek CISC Komutu, Üç İş

```
1 org 100h
2 mov bx, offset sayi ; adres
3 mov ax, 7           ; değer
▶ add [bx], ax        ; oku, topla,
  yaz
5 add [bx], ax        ; tekrar
6 ret
7 sayi dw 0123h
```



`add [bx], ax` tek komut ama içeride üç iş yapar. **1 · Oku:** BIU, 010Bh adresindeki sözcüğü (0123h) bir bellek okuma döngüsüyle geçici yazmaca getirir.

Adım 3 / 7

DEĞİŞKENLER

BX = 010Bh | AX = 0007h | geçici = 0123h | sayi = 0123h

Kod İzleme: Tek CISC Komutu, Üç İş

```
1 org 100h
2 mov bx, offset sayi ; adres
3 mov ax, 7           ; değer
▶ add [bx], ax        ; oku, topla,
  yaz
5 add [bx], ax        ; tekrar
6 ret
7 sayi dw 0123h
```



2 · Topla: ALU, 0123h + 0007h = **012Ah** hesaplar ve bayrakları günceller.

Adım 4 / 7

DEĞİŞKENLER

`BX = 010Bh` `AX = 0007h` `geçici = 012Ah` `sayi = 0123h`

Kod İzleme: Tek CISC Komutu, Üç İş

```
1 org 100h
2 mov bx, offset sayi ; adres
3 mov ax, 7           ; değer
▶ add [bx], ax        ; oku, topla,
  yaz
5 add [bx], ax        ; tekrar
6 ret
7 sayi dw 0123h
```



3 · Yaz: sonuç ikinci bir yol döngüsüyle belleğe geri yazılır: sayi = **012Ah**. Bir komut = 2 bellek erişimi.

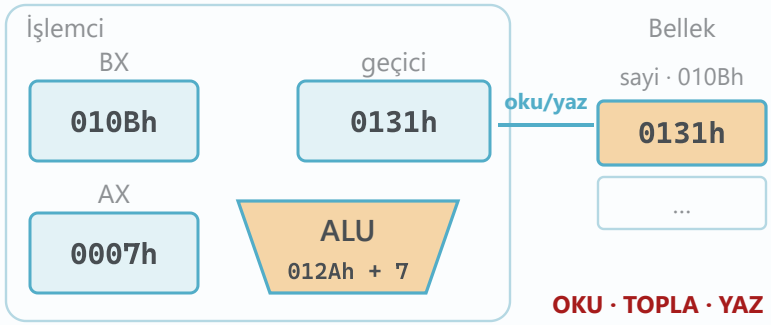
Adım 5 / 7

DEĞİŞKENLER

`BX = 010Bh` `AX = 0007h` `geçici = 012Ah` `sayi = 012Ah`

Kod İzleme: Tek CISC Komutu, Üç İş

```
1 org 100h
2 mov bx, offset sayi ; adres
3 mov ax, 7           ; değer
4 add [bx], ax        ; oku, topla, yaz
▶ add [bx], ax        ; tekrar
6 ret
7 sayi dw 0123h
```



Aynı komut yeniden: oku (012Ah), topla (+7), yaz → sayi = **0131h**.

Adım 6 / 7

DEĞİŞKENLER

`BX = 010Bh` `AX = 0007h` `geçici = 0131h` `sayi = 0131h`

Kod İzleme: Tek CISC Komutu, Üç İş

```
1 org 100h
2 mov bx, offset sayi ; adres
3 mov ax, 7           ; değer
4 add [bx], ax        ; oku, topla,
  yaz
5 add [bx], ax        ; tekrar
▶ ret
7 sayi dw 0123h
```



CISC kodu kısa: tek komut. RISC işlemcide aynı iş üç komuttur (`LDR` , `ADD` , `STR`); ara değer yazmaçta görünür olur.

Adım 7 / 7

DEĞİŞKENLER

`BX = 010Bh` `AX = 0007h` `geçici = 0131h` `sayi = 0131h`

İŞLEMCİ TÜRLERİ

RISC ve CISC İşlemciler

Ölçüt	RISC	CISC
Komut kümesi	az ve basit	geniş ve karmaşık
Komut uzunluğu	sabit	değişken
Komut başına tur	az (≈1, boru hattıyla)	çok (değişken)
Bellek erişimi	yalnızca load/store	çoğu komut belleğe erişebilir
Yazmaç sayısı	çok	az
Program boyutu	daha uzun	daha kısa
Güç tüketimi	düşük	görece yüksek
Esneklik	yük derleyicide	geniş komut yelpazesi, esnek kullanım

RISC: **az çeşit komutu hızlı** çalıştırır, yüksek performans/düşük güç. CISC: **çok çeşit komut** sunar ama tek tek komutlar daha yavaştır.

Açıkça Paralel Komut İşleme (EPIC)

- Hangi komutların **paralel** çalışabileceğine donanım değil **derleyici** karar verir ve bunu koda açıkça yazar.
- Yüksek saat frekansı gerekmeden çok sayıda komut aynı anda yürütülebilir.
- Komutlar **128 bitlik paketler** (bundle) hâlinde kodlanır: **41 bitlik üç komut** + **5 bitlik şablon**.



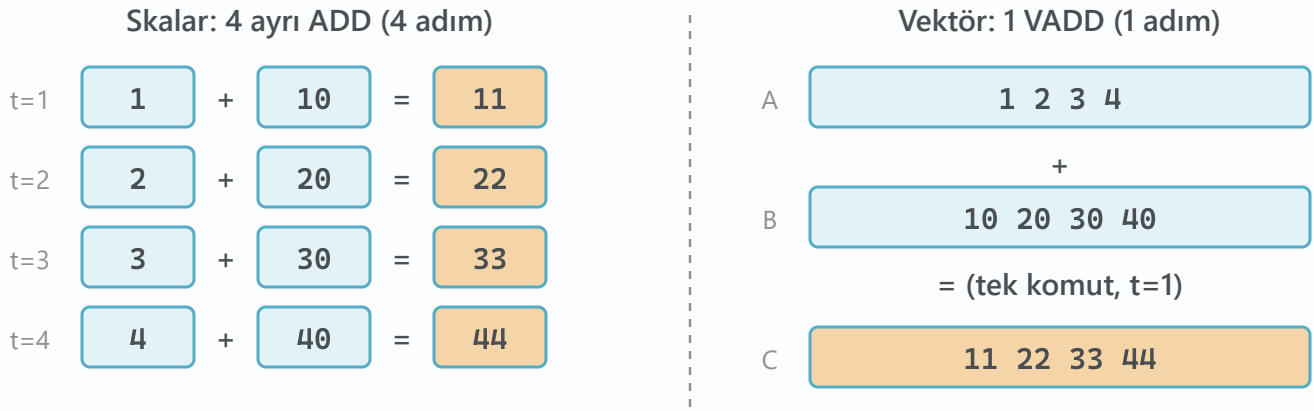
Örnek

Intel–HP ortak mimarisi **IA-64**; Itanium işlemcileri (2001–2021).

Sonuç

Derleyiciye çok yük bindirdi ve x86 uyumluluğu zayıf kaldı; piyasada x86-64'e yenildi.

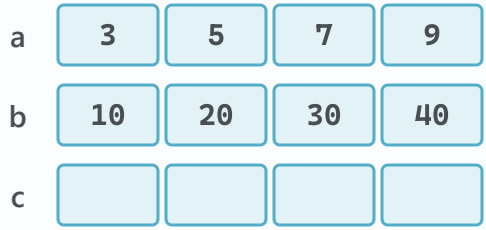
Skalar ve Vektör İşleme



Skalar işlemci her komutta **tek bir veri öğesini** işler; vektör işlemci tek bir komutla **tüm bir veri dizisini** (vektörü) işler. Komut getirme/çözme ve döngü yükü bir kez ödenir.

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
▶  mov si, 0
▶  mov cx, 4
4  dongu: mov al, a[si]   ; bir öğe oku
5  add al, b[si]          ; bir toplama
6  mov c[si], al          ; bir öğe yaz
7  inc si
8  loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

2 komut

vektör (4 öğe)

1 toplama

SI = 0 (indis), CX = 4 (öğe sayısı). Skalar işlemci her komutta **tek** bir veri öğesiyle çalışır.

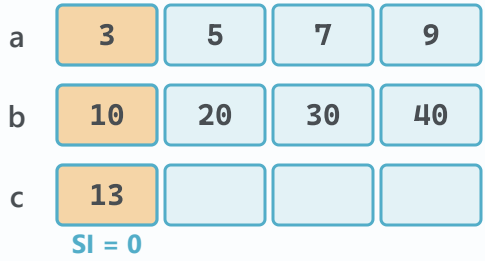
Adım 1 / 9

DEĞİŞKENLER

`SI = 0` `CX = 4` `AL = 0` `komut = 2`

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
2  mov si, 0
3  mov cx, 4
▶  dongu: mov al, a[si]   ; bir öğe oku
▶  add al, b[si]          ; bir toplama
▶  mov c[si], al          ; bir öğe yaz
7  inc si
8  loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

5 komut

vektör (4 öğe)

1 toplama

Öğe 0: a[0] + b[0] = 3 + 10 = **13** → c[0]. Bir öğe için üç komut getirilip çözüldü.

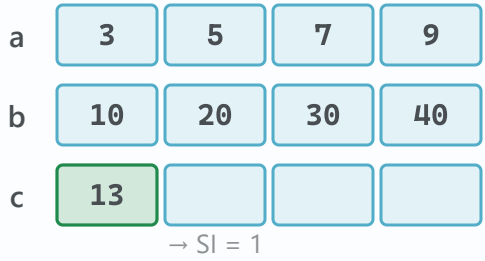
Adım 2 / 9

DEĞİŞKENLER

`SI = 0` `CX = 4` `AL = 13` `komut = 5`

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
2  mov si, 0
3  mov cx, 4
4  dongu: mov al, a[si]   ; bir öğe oku
5  add al, b[si]          ; bir toplama
6  mov c[si], al          ; bir öğe yaz
   inc si
   loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

7 komut

vektör (4 öğe)

1 toplama

SI = 1, CX = 3 ≠ 0 → döngü başına. Döngü yükü (INC, LOOP) her öğe için yeniden ödenir.

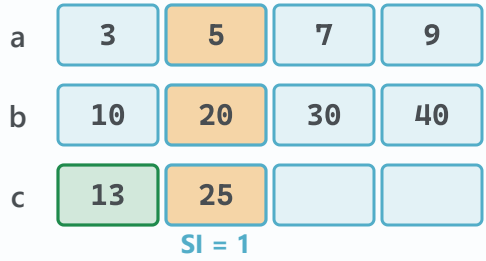
Adım 3 / 9

DEĞİŞKENLER

`SI = 1` `CX = 3` `AL = 13` `komut = 7`

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
2  mov si, 0
3  mov cx, 4
▶ dongu: mov al, a[si]   ; bir öğe oku
▶ add al, b[si]          ; bir toplama
▶ mov c[si], al          ; bir öğe yaz
7  inc si
8  loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

10 komut

vektör (4 öğe)

1 toplama

Öğe 1: a[1] + b[1] = 5 + 20 = **25** → c[1]. Bir öğe için üç komut getirilip çözüldü.

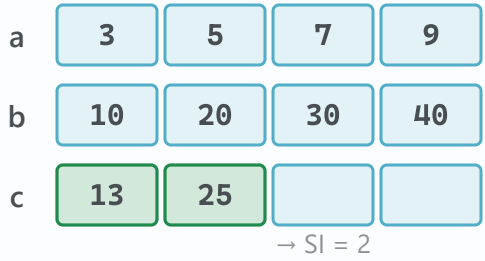
Adım 4 / 9

DEĞİŞKENLER

`SI = 1` `CX = 3` `AL = 25` `komut = 10`

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
2  mov si, 0
3  mov cx, 4
4  dongu: mov al, a[si]   ; bir öğe oku
5  add al, b[si]          ; bir toplama
6  mov c[si], al          ; bir öğe yaz
   inc si
   loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

12 komut

vektör (4 öğe)

1 toplama

SI = 2, CX = 2 ≠ 0 → döngü başına. Döngü yükü (INC, LOOP) her öğe için yeniden ödenir.

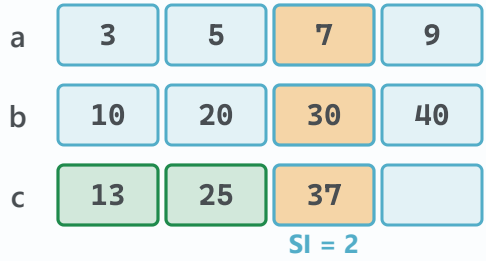
Adım 5 / 9

DEĞİŞKENLER

`SI = 2` `CX = 2` `AL = 25` `komut = 12`

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
2  mov si, 0
3  mov cx, 4
▶  dongu: mov al, a[si]   ; bir öğe oku
▶  add al, b[si]          ; bir toplama
▶  mov c[si], al          ; bir öğe yaz
7  inc si
8  loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

15 komut

vektör (4 öğe)

1 toplama

Öğe 2: a[2] + b[2] = 7 + 30 = **37** → c[2]. Bir öğe için üç komut getirilip çözüldü.

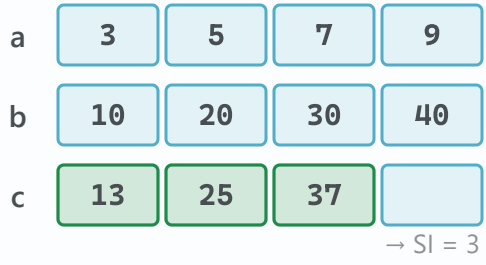
Adım 6 / 9

DEĞİŞKENLER

`SI = 2` `CX = 2` `AL = 37` `komut = 15`

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
2  mov si, 0
3  mov cx, 4
4  dongu: mov al, a[si]   ; bir öğe oku
5  add al, b[si]          ; bir toplama
6  mov c[si], al          ; bir öğe yaz
▶  inc si
▶  loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

17 komut

vektör (4 öğe)

1 toplama

SI = 3, CX = 1 ≠ 0 → döngü başına. Döngü yükü (INC, LOOP) her öğe için yeniden ödenir.

Adım 7 / 9

DEĞİŞKENLER

SI = 3 | CX = 1 | AL = 37 | komut = 17

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
2  mov si, 0
3  mov cx, 4
▶  dongu: mov al, a[si]   ; bir öğe oku
▶  add al, b[si]          ; bir toplama
▶  mov c[si], al          ; bir öğe yaz
7  inc si
8  loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

20 komut

vektör (4 öğe)

1 toplama

Öğe 3: a[3] + b[3] = 9 + 40 = **49** → c[3]. Bir öğe için üç komut getirilip çözüldü.

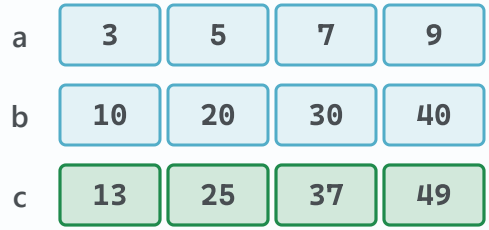
Adım 8 / 9

DEĞİŞKENLER

`SI = 3` `CX = 1` `AL = 49` `komut = 20`

Kod İzleme: Skalar İşlemcide c = a + b

```
1  org 100h
2  mov si, 0
3  mov cx, 4
4  dongu: mov al, a[si]   ; bir öğe oku
5  add al, b[si]          ; bir toplama
6  mov c[si], al          ; bir öğe yaz
   inc si
   loop dongu
9  ret
10 a db 3, 5, 7, 9
11 b db 10, 20, 30, 40
12 c db 4 dup(0)
```



skalar

22 komut

vektör (4 öğe)

1 toplama

CX = 0: döngü bitti. 4 toplama için **22 komut** yürütüldü; vektör işlemci tek bir vektör toplama komutuyla dört öğeyi birden işler.

Adım 9 / 9

DEĞİŞKENLER

`SI = 4` `CX = 0` `AL = 49` `komut = 22`

Vektör İşlemcisi (Vector Processor)

- **Vektör** adı verilen veri dizileri üzerinde matematiksel işlemler yapar.
- Tek bir veri öğesi üzerinde çalışan **skalar** işlemcilere göre bu tür işlerde çok daha hızlıdır.
- Birden çok veri öğesini aynı anda (ya da derin boru hattında art arda) işler.
- Uzun **vektör yazmaçları** ve vektör yükle/sakla birimleri vardır.
- Süper bilgisayar, sunucu ve GPU gibi yüksek performanslı sistemlerde kullanılır.



Tarihî
Cray-1 (1976) süper bilgisayarı

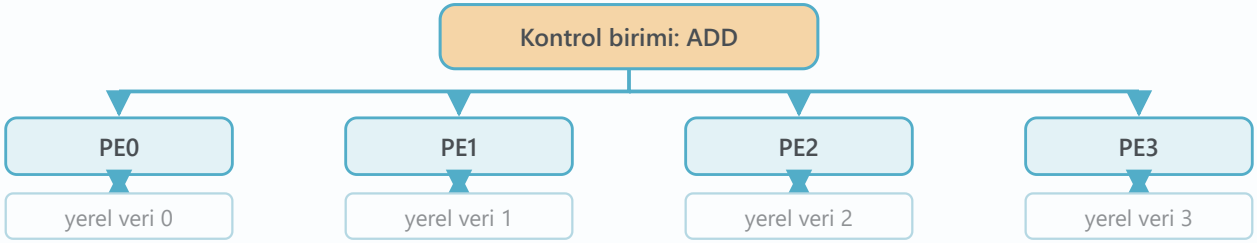
Bugün
x86 SSE/AVX, ARM NEON/SVE, RISC-V V uzantısı

Uygulama
hava tahmini, fizik benzetimi, yapay zekâ

Dizi İşlemcisi (Array Processor)

- **SIMD** (Single Instruction, Multiple Data · tek komut, çok veri) olarak da adlandırılır.
- Tek kontrol birimi, aynı komutu çok sayıda **işlem elemanına** (PE) yayınlar; her PE kendi verisini işler.
- Paralel hesaplamalar ve vektör işlemleri için optimize edilmiştir.
- Tek veri öğesiyle çalışan skalar işlemcilerden farklıdır.
- Süper bilgisayar, sunucu ve GPU'larda kullanılır; genel amaçlı hesaplamada nadirdir.

Flynn sınıflandırması	
SISD	tek komut, tek veri (klasik 8086)
SIMD	tek komut, çok veri (dizi/vektör, GPU)
MISD	çok komut, tek veri (nadir; hata toleransı)
MIMD	çok komut, çok veri (çok çekirdek, kümeler)

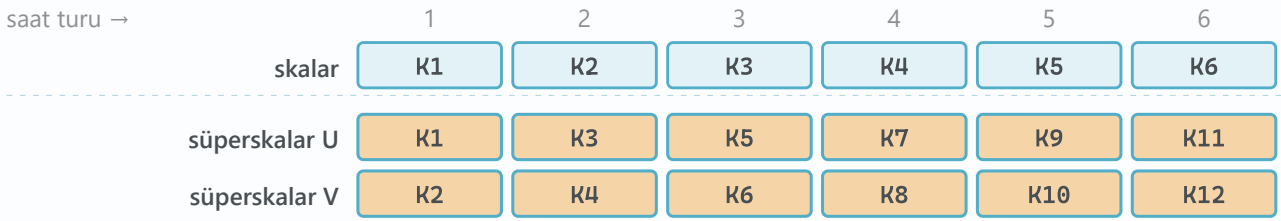


Sayıl (Skalar) İşlemci (Scalar Processor)

- Sayıl (vektör olmayan) veriyi, komut başına **bir veri öğesi** olarak işler.
- Basit sayıl işlemciler yalnızca tamsayılarla çalışır.
- Güçlü olanlar kayan noktalı sayıları da işler: tamsayı birimi (**ALU**) ve kayan nokta birimi (**FPU**) aynı yongadadır.
- CISC ya da RISC mimarisinde olabilir.
- Genel amaçlı hesaplamada en yaygın türdür.

Süperskalar (superscalar)

Birden çok **boru hattı**/yürütme birimiyle **bir saat turunda birden fazla komut** başlatabilen skalar işlemci. İlk x86 örneği: **Pentium** (U ve V boru hatları).



Dijital Sinyal İşlemcisi (DSP)

- Dijital sinyaller üzerinde işlem yapmak için optimize edilmiştir.
- Dijital sinyal: ses, görüntü gibi duyusal bilgileri temsil eden **örnek (sample) dizileri**.
- Sinyal işlemeye özel komut kümesi ve donanım: tek turda **çarp-topla (MAC)**, dairesel tampon adresleme, sıfır yüklü döngü.
- Uygulamalar:
  - ses işleme ve filtreleme
  - görüntü işleme ve işaretleme
  - telekomünikasyon altyapısı (modem, baz istasyonu)

FIR süzgeci

$$y[n] = \Sigma\, h[k] \cdot x[n - k]$$

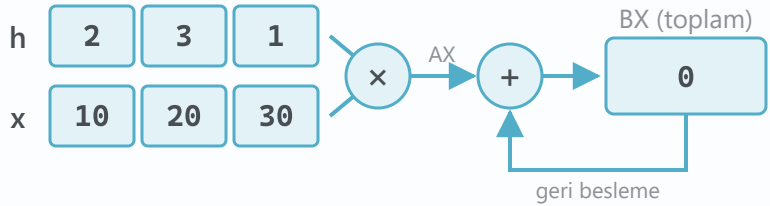
Her çıktı örneği için N çarpma + N toplama. DSP her terimi **tek MAC komutuyla** yapar:

```
A ← A + h[k] × x[n−k]
```

Ör: TI TMS320, Analog Devices SHARC

Kod İzleme: FIR Toplamı 8086 ile

```
1  org 100h
2  mov si, 0
3  mov bx, 0          ; toplam
4  mov cx, 3          ; terim sayısı
5  dongu: mov al, h[si]
6  mul x[si]          ; AX = h × x
7  add bx, ax         ; toplama ekle
8  inc si
9  loop dongu
10 ret
11 h db 2, 3, 1
12 x db 10, 20, 30
```



AX = 0 y = 2·10 + 3·20 + 1·30 = 110

BX = 0 akümülatör, CX = 3 terim. Süzgeç çıktısı y = h[0]·x[0] + h[1]·x[1] + h[2]·x[2].

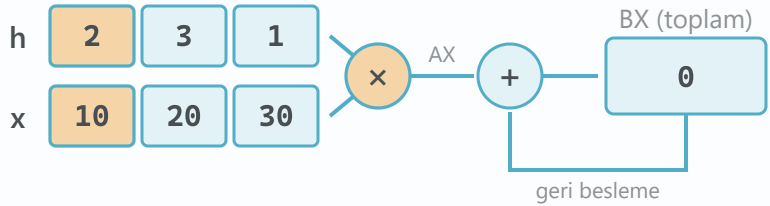
Adım 1 / 7

DEĞİŞKENLER

SI = 0 | CX = 3 | AX = 0 | BX = 0

Kod İzleme: FIR Toplamı 8086 ile

```
1  org 100h
2  mov si, 0
3  mov bx, 0          ; toplam
4  mov cx, 3          ; terim sayısı
▶  dongu: mov al, h[si]
▶  mul x[si]          ; AX = h × x
7  add bx, ax         ; toplama ekle
8  inc si
9  loop dongu
10 ret
11 h db 2, 3, 1
12 x db 10, 20, 30
```



AX = 20 y = 2·10 + 3·20 + 1·30 = 110

Terim 0: AL = h[0] = 2, `mul` → AX = 2 × 10 = **20**. 8086'da MUL tek başına 70 turdan uzun sürer.

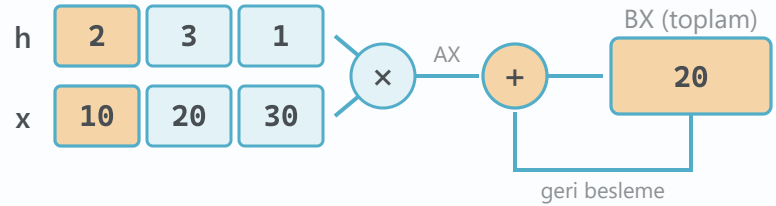
Adım 2 / 7

DEĞİŞKENLER

SI = 0 | CX = 3 | AX = 20 | BX = 0

Kod İzleme: FIR Toplamı 8086 ile

```
1  org 100h
2  mov si, 0
3  mov bx, 0          ; toplam
4  mov cx, 3          ; terim sayısı
5  dongu: mov al, h[si]
6  mul x[si]          ; AX = h × x
▶  add bx, ax         ; toplama ekle
8  inc si
9  loop dongu
10 ret
11 h db 2, 3, 1
12 x db 10, 20, 30
```



AX = 20 y = 2·10 + 3·20 + 1·30 = 110

`add bx, ax` : toplam = **20**. Çarpma ve toplama iki ayrı komut; DSP'de ikisi tek MAC komutudur.

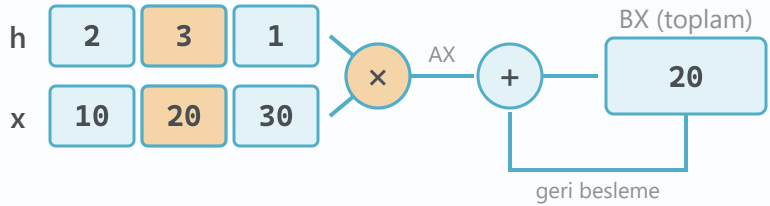
Adım 3 / 7

DEĞİŞKENLER

SI = **0** CX = **3** AX = **20** BX = **20**

Kod İzleme: FIR Toplamı 8086 ile

```
1  org 100h
2  mov si, 0
3  mov bx, 0          ; toplam
4  mov cx, 3          ; terim sayısı
▶  dongu: mov al, h[si]
▶  mul x[si]          ; AX = h × x
7  add bx, ax         ; toplama ekle
8  inc si
9  loop dongu
10 ret
11 h db 2, 3, 1
12 x db 10, 20, 30
```



AX = 60 y = 2·10 + 3·20 + 1·30 = 110

Terim 1: AL = h[1] = 3, `mul` → AX = 3 × 20 = **60**. 8086'da MUL tek başına 70 turdan uzun sürer.

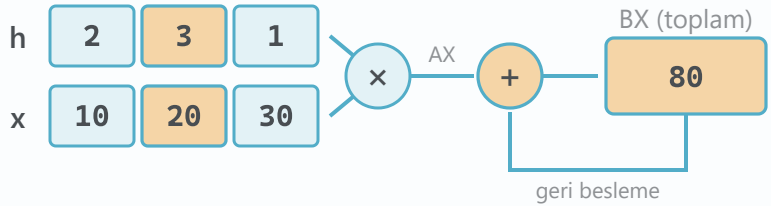
Adım 4 / 7

DEĞİŞKENLER

SI = 1 | CX = 2 | AX = 60 | BX = 20

Kod İzleme: FIR Toplamı 8086 ile

```
1  org 100h
2  mov si, 0
3  mov bx, 0          ; toplam
4  mov cx, 3          ; terim sayısı
5  dongu: mov al, h[si]
6  mul x[si]          ; AX = h × x
▶  add bx, ax         ; toplama ekle
8  inc si
9  loop dongu
10 ret
11 h db 2, 3, 1
12 x db 10, 20, 30
```



AX = 60 y = 2·10 + 3·20 + 1·30 = 110

`add bx, ax` : toplam = **80**. Çarpma ve toplama iki ayrı komut; DSP'de ikisi tek MAC komutudur.

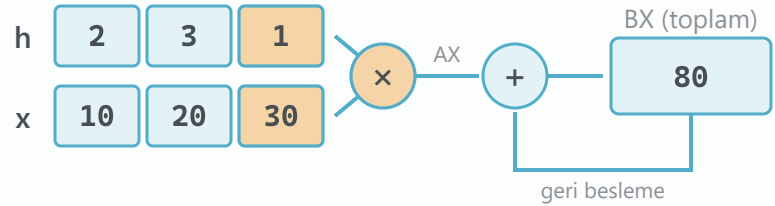
Adım 5 / 7

DEĞİŞKENLER

SI = **1** CX = **2** AX = **60** BX = **80**

Kod İzleme: FIR Toplamı 8086 ile

```
1  org 100h
2  mov si, 0
3  mov bx, 0          ; toplam
4  mov cx, 3          ; terim sayısı
▶  dongu: mov al, h[si]
▶  mul x[si]          ; AX = h × x
7  add bx, ax         ; toplama ekle
8  inc si
9  loop dongu
10 ret
11 h db 2, 3, 1
12 x db 10, 20, 30
```



AX = 30 y = 2·10 + 3·20 + 1·30 = 110

Terim 2: AL = h[2] = 1, `mul` → AX = 1 × 30 = **30**. 8086'da MUL tek başına 70 turdan uzun sürer.

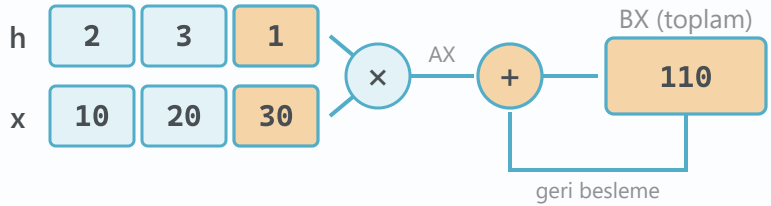
Adım 6 / 7

DEĞİŞKENLER

SI = 2 | CX = 1 | AX = 30 | BX = 80

Kod İzleme: FIR Toplamı 8086 ile

```
1  org 100h
2  mov si, 0
3  mov bx, 0          ; toplam
4  mov cx, 3          ; terim sayısı
5  dongu: mov al, h[si]
6  mul x[si]          ; AX = h × x
▶  add bx, ax         ; toplama ekle
8  inc si
9  loop dongu
10 ret
11 h db 2, 3, 1
12 x db 10, 20, 30
```



AX = 30 y = 2·10 + 3·20 + 1·30 = 110

`add bx, ax` : toplam = **110**. Üç terim bitti: y = **110**. Her terim için ayrı MUL + ADD (+ döngü yükü) gerekti; DSP bunu tek **MAC** komutuyla, tek turda yapar.

Adım 7 / 7

DEĞİŞKENLER

SI = **2** CX = **1** AX = **30** BX = **110**

Sembolik İşlemci (Symbolic Processor)

- **Sembolik hesaplama** için tasarlanmıştır: sayılar yerine semboller, listeler ve kurallar işlenir.
- **PROLOG** (ve **LISP**) işlemcileri olarak da adlandırılır.
- Anlamsal (semantik) hesaplama ve sembolik mantık kuralları üzerine odaklanır.
- Matematiksel ifadeleri ve sembolleri işler; karmaşık ifadeleri sadeleştirebilir, türev alabilir.

Uygulama alanları

- Uzman sistemler (expert systems)
- Makine zekâsı (machine intelligence)
- Kural tabanlı çıkarım

```
baba(ali, veli).
dede(X, Z) :- baba(X, Y),
              baba(Y, Z).
```

Prolog kuralı: sembolik çıkarım

1980'lerin LISP makineleri (Symbolics) ve Japonya'nın 5. Nesil Bilgisayar projesi bu türdendi; bugün bu işler genel amaçlı işlemcilerde yazılımla yapılır.

İŞLEMCİ TÜRLERİ

Grafik İşlem Birimi (GPU)

- 2B ve 3B grafiklerin **hızlandırılması** için tasarlanmıştır.
- Gerçek zamanlı, yüksek kaliteli grafikler üretir.
- Görüntü oluşturma (render) görevlerinde etkin ve verimlidir: **gölgelendirme** (shading), **dokulama** (texturing), rasterleştirme.
- Binlerce basit çekirdek aynı programı farklı piksellerde çalıştırır (SIMD/SIMT).
- Grafik kartı ve hızlandırıcı (accelerator) gibi özel donanımlarda kullanılır.

CPU

birkaç güçlü çekirdek · gecikmeyi azaltır · karmaşık akış

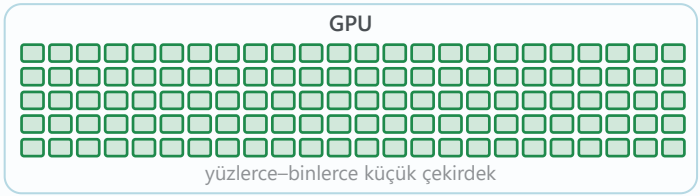
GPU

binlerce basit çekirdek · verimi artırır · paralel veri

GPGPU

CUDA/OpenCL ile bilimsel hesap ve derin öğrenme





İşlemci Türleri: Özet Tablo

Tür	Temel fikir	Güçlü yanı	Örnek
CISC	zengin, karmaşık komutlar	kısa kod, uyumluluk	8086, Pentium, 68000
RISC	basit komut + boru hattı	hız, düşük güç	ARM, MIPS, RISC-V
EPIC	derleyici paralelliği belirler	komut düzeyi paralellik	IA-64 Itanium
Skalar	komut başına bir veri	genel amaçlı	çoğu CPU çekirdeği
Vektör	komut başına veri dizisi	bilimsel hesap	Cray-1, AVX
Dizi (SIMD)	çok PE, aynı komut	büyük paralellik	ILLIAC IV, GPU
DSP	MAC, özel adresleme	gerçek zamanlı sinyal	TMS320, SHARC
Sembolik	sembol/kural işleme	yapay zekâ (klasik)	LISP makineleri
GPU	binlerce basit çekirdek	grafik, derin öğrenme	NVIDIA, AMD Radeon

INTEL X86 EVRİMİ

Mikroişlemcilerin Evrimi

1947 · Transistör

Bell Laboratuvarları'nda (Bardeen, Brattain, Shockley) icat edildi, 1948'de duyuruldu; mikroelektronik devrimi başladı.

Mikroişlemcilerin Evrimi

1947 · Transistör

Bell Laboratuvarları'nda (Bardeen, Brattain, Shockley) icat edildi, 1948'de duyuruldu; mikroelektronik devrimi başladı.

1958 · Tümleşik devre (IC)

Texas Instruments'ta Jack Kilby ilk tümleşik devreyi gösterdi. Birçok transistör tek bir yonga üzerinde birleştirilerek karmaşık devreler kurulabildi.

Mikroişlemcilerin Evrimi

1947 · Transistör

Bell Laboratuvarları'nda (Bardeen, Brattain, Shockley) icat edildi, 1948'de duyuruldu; mikroelektronik devrimi başladı.

1958 · Tümleşik devre (IC)

Texas Instruments'ta Jack Kilby ilk tümleşik devreyi gösterdi. Birçok transistör tek bir yonga üzerinde birleştirilerek karmaşık devreler kurulabildi.

1971 · İlk mikroişlemci

Intel 4004: bütün bir CPU tek yongada. Mikroişlemci bilgisayarın merkezî işlem birimi (CPU) görevini üstlendi.

INTEL X86 EVRİMİ

Mikroişlemcilerin Evrimi

1947 · Transistör

Bell Laboratuvarları'nda (Bardeen, Brattain, Shockley) icat edildi, 1948'de duyuruldu; mikroelektronik devrimi başladı.

1958 · Tümleşik devre (IC)

Texas Instruments'ta Jack Kilby ilk tümleşik devreyi gösterdi. Birçok transistör tek bir yonga üzerinde birleştirilerek karmaşık devreler kurulabildi.

1971 · İlk mikroişlemci

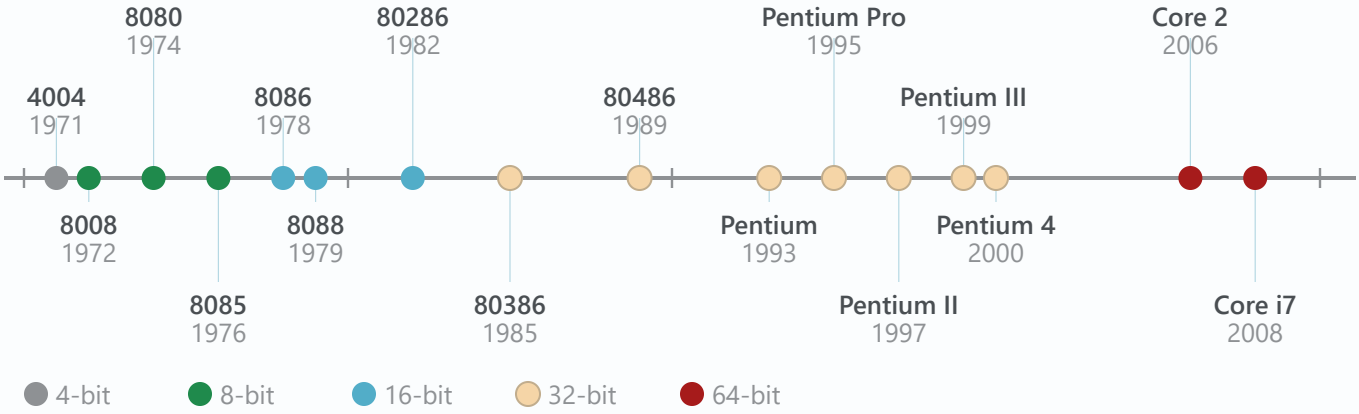
Intel 4004: bütün bir CPU tek yongada. Mikroişlemci bilgisayarın merkezî işlem birimi (CPU) görevini üstlendi.

Sonuç

Bilgisayarlar oda büyüklüğünden masaüstüne, oradan cebe sığdı: daha **küçük**, daha **güçlü**, daha **hızlı** ve daha **ucuz**.

INTEL X86 EVRİMİ

Intel İşlemcileri: Zaman Çizelgesi



x86 adı 8086'dan sonraki 80186, 80286, 80386, 80486 adlarının ortak “86” sonekinden gelir. Hepsi 8086 kodunu çalıştırabilir: **geriye uyumluluk**.

INTEL X86 EVRİMİ

4 ve 8 Bitlik Mikroişlemciler

İşlemci	Yıl	Saat	Transistör	Komut hızı / not
4004 (4-bit)	1971	740 kHz	2.300	≈ 60.000 komut/s; Ted Hoff, Federico Faggin, Stanley Mazor
4040 (4-bit)	1974	740 kHz	≈ 3.000	4004'ün geliştirilmiş sürümü (kesme, daha çok komut)
8008 (8-bit)	1972	500 kHz	3.500	≈ 50.000 komut/s; ilk 8 bitlik
8080 (8-bit)	1974	2 MHz	6.000	8008'in ≈10 katı hız; 16-bit adres (64 KB); ilk kişisel bilgisayarlar (Altair 8800)
8085 (8-bit)	1976	3 MHz	6.500	tek +5 V besleme; 16-bit adres yolu; ≈ 769.000 komut/s (en kısa komut 1,3 µs)

INTEL X86 EVRİMİ

16 Bitlik Mikroişlemciler

İşlemci	Yıl	Saat	Transistör	Özellik
8086	1978	5 · 8 · 10 MHz	29.000	16-bit veri yolu, 20-bit adres yolu (1 MB); donanımsal çarpma/bölme; ≈ 0,33–0,75 MIPS
8088	1979	5 · 8 MHz	29.000	8086'nın 8-bit dış veri yollu ucuz sürümü; IBM PC (1981, 4,77 MHz)
80186 / 80188	1982	6 MHz'den	≈ 55.000	yonga üstünde kesme denetleyicisi, saat üreteci, yerel yol denetleyicisi, sayaçlar, DMA; 80188 ucuz (8-bit dış yol) sürüm
80286	1982	6 – 12,5 MHz	134.000	16-bit veri, 24-bit adres yolu (16 MB); korumalı mod; ≈ 0,9–2,7 MIPS

8086 komutları ortalama 10'dan fazla tur sürdüğü için 5 MHz'de yaklaşık **0,33 MIPS** elde edilir.

INTEL X86 EVRİMİ

32 Bitlik Mikroişlemciler

İşlemci	Yıl	Saat	Transistör	Özellik
80386	1985	16 – 33 MHz	275.000	32-bit veri ve adres yolu (4 GB), sayfalama; sürümler: 386DX, 386SX (1988; 16-bit dış veri, 24-bit adres), 386SL (dizüstü)
80486	1989	16 – 100 MHz	1,2 milyon	yonga üstü 8 KB önbellek, entegre FPU, 5 aşamalı boru hattı; sürümler: 486DX, 486SX (FPU'suz), DX2, DX4
Pentium	1993	60 · 66 MHz (ilk)	3,1 milyon	süperskalar (2 boru hattı); ayrı **8 KB komut + 8 KB veri** önbelleği; 64-bit dış veri yolu

INTEL X86 EVRİMİ

64 Bitlik Mikroişlemciler

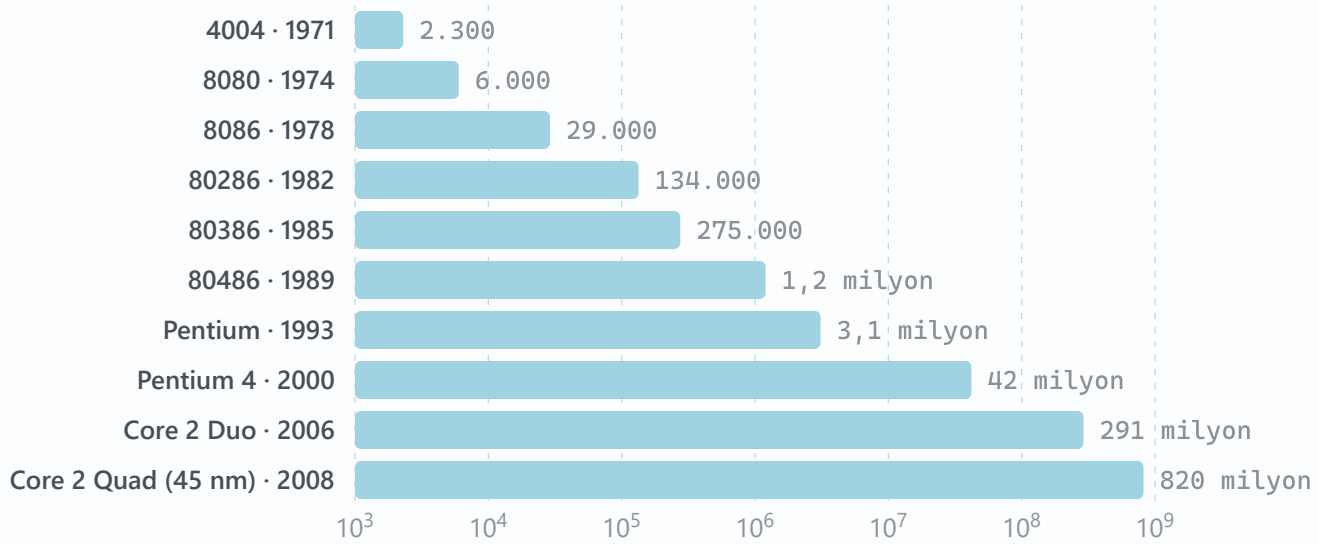
İşlemci	Yıl	Saat	Transistör	Özellik
Core 2	2006	1,2 – 3 GHz	291 milyon (Duo)	çekirdek başına 64 KB L1 (32 KB komut + 32 KB veri), 2–4 MB paylaşımlı L2; sürümler: Duo, Quad, Extreme
Core i7	2008	2,66 – 3,33 GHz	731 milyon	4 çekirdek, Hyper-Threading, entegre bellek denetleyicisi, 8 MB L3
Core i5	2009	2,4 – 3,6 GHz	—	orta sınıf; Turbo Boost
Core i3	2010	2,93 – 3,33 GHz	—	giriş sınıfı; 2 çekirdek, entegre grafik

64 bitlik x86 (x86-64) ilk kez **AMD** tarafından (AMD64, Opteron/Athlon 64, 2003) sunuldu; Intel bunu 2004'te Pentium 4 ile benimsedi.

Saat aralıkları 1. nesil masaüstü/mobil modellere göredir.

INTEL X86 EVRİMİ

Transistör Sayısı ve Moore Yasası



Moore yasası (1965/1975): yonga başına transistör sayısı yaklaşık **her iki yılda bir iki katına** çıkar. 4004'ten Core 2 Quad'a 37 yılda ≈ 2^{18} kat artış.

INTEL X86 EVRİMİ · TEMEL ÖZELLİKLER

8080 ve 8086

Mikroişlemcilerin yapısı, örgütlenimi ve teknolojisi; ilk genel amaçlı mikroişlemci 8080'den, 29.000 transistörlü 8086'ya ve 820 milyon transistörlü dört çekirdekli Core 2'ye kadar dramatik biçimde değişti.

8080 (1974)

- İlk **genel amaçlı** mikroişlemci olarak anılır.
- 8-bit veri yolu, 16-bit adres yolu (64 KB).
- İlk kişisel bilgisayarlarda (Altair 8800) kullanıldı.

8086 (1978)

- 16-bit veri yolu ve 16 bitlik, daha çok sayıda yazmaç.
- 20-bit adres yolu: **1 MB** adreslenebilir bellek.
- Yalnızca **gerçek mod** (real mode).
- Komutları çalıştırmadan önce **önden getirip** (prefetch) **6 baytlık komut kuyruğunda** bekletir → getirme ile yürütme üst üste biner.

8086'da önbellek (cache) yoktur; hız kazancını komut kuyruğu sağlar. Yonga üstü önbellek x86'da 80486 ile geldi.

INTEL X86 EVRİMİ · TEMEL ÖZELLİKLER

80286, 80386 ve 80486

80286 (1982)

- **16 MB** adreslenebilir bellek (24-bit adres).
- Gerçek mod + **16 bitlik korumalı mod** (protected mode).
- 16-bit veri yolu ve yazmaçlar.

80386 (1985)

- Intel'in **ilk 32 bitlik** mikroişlemcisi.
- Çoklu görevi (multitasking) donanımla destekler.
- 32 bitlik korumalı mod, **sayfalama** (paging), sanal 8086 modu.
- **4 GB** adreslenebilir bellek.

80486 (1989)

- Yonga üstü **önbellek** (8 KB) ve komut **boru hattı**.
- Sayfa yazma koruması (CR0.WP biti).
- Karmaşık aritmetik için **entegre matematik işlemcisi** (FPU); önceden ayrı 80387 yongasıydı.

Korumalı mod: her programın belleği diğerlerinden yalıtılır; bir programın hatası işletim sistemini çökertemez. Modern işletim sistemlerinin temeli budur.

Pentium ve Pentium Pro

Pentium (1993)

- **Süperskalar:** iki boru hattıyla (U, V) komutlar paralel çalışabilir.
- Dallanma tahmini (ilk x86).
- **Sayfa boyutu uzantısı** (PSE): 4 KB'nin yanında 4 MB'lık sayfalar → sayfalamada iyileştirme.
- Ayrı komut ve veri önbellekleri.

Pentium Pro (1995)

- Gelişmiş **dallanma tahmini** (branch prediction).
- **Veri akış analizi** (data flow analysis) ile sıra dışı (out-of-order) yürütme.
- **Spekülatif yürütme** (speculative execution).
- Aynı paket içinde **düzey 2 (L2) önbellek**.
- PAE: 32 bitlik sanal adres, **36 bitlik fiziksel** adrese çevrilir (64 GB).

Pentium Pro'nun bu üç tekniğine birlikte Intel “dinamik yürütme” (dynamic execution) adını verdi.

Pentium II, III ve 4

Pentium II (1997)

- **MMX** teknolojisini (multimedya için tamsayı SIMD komutları) Pentium Pro çekirdeğiyle birleştirdi.
- Video, ses ve grafik verisini etkili işler.

Pentium III (1999)

- **SSE** (Streaming SIMD Extensions): **70 yeni komut**, kayan noktalı SIMD.
- 3B grafik yazılımını destekler.
- En yüksek saat hızı 1,4 GHz.

Pentium 4 (2000)

- Multimedya için kayan nokta geliştirmeleri (**SSE2**, sonra SSE3).
- Çok derin boru hattı (NetBurst), 3,8 GHz'e kadar.
- Hyper-Threading.

MMX ilk kez 1997'de “Pentium MMX” ile geldi, Pentium II onu devraldı.

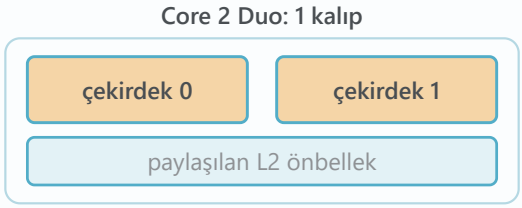
Core ve Core 2

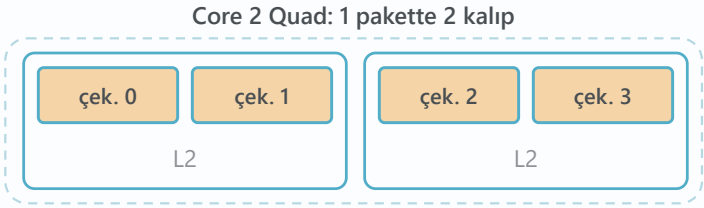
Core (Core Duo, 2006)

- İki çekirdeğin **aynı kalıpta** (die) L2 önbelleğini paylaştığı ilk Intel işlemcisi.
- **Sanallaştırma teknolojisi** (VT-x) eklendi.
- Düşük güç (mobil kökenli tasarım).

Core 2 (2006)

- **64 bitlik** mimari (Intel 64).
- 64 bitlik yazmaç kümesi ve adresleme modları.
- **Core 2 Quad**: tek pakette dört çekirdek (iki çift çekirdekli kalıp).





Intel'in ilk çift çekirdekli masaüstü işlemcisi 2005'teki Pentium D'dir (çekirdeklerin L2 önbellekleri ayrı).

x86 Mimarisinin Avantajları

Uyumluluk (compatibility)
Eski yazılımlar değiştirilmeden yeni x86 işlemcilerde çalışır: 1978'de 8086 için yazılmış kod bugün de çalışabilir.

Performans (performance)
Her yeni işlemci nesli öncekinden daha hızlı ve verimli oldu.

Çeşitlilik (versatility)
Kişisel bilgisayarlardan sunuculara, gömülü sistemlere ve mobil cihazlara kadar geniş bir kullanım yelpazesi.

Geniş endüstri desteği
Büyük bir donanım ve yazılım sağlayıcı ekosistemi tarafından desteklenir; bu destek yenilik ve gelişmenin itici gücü oldu.

INTEL X86 EVRİMİ

x86 Mimarisinin Dezavantajları

Karmaşık komut kümesi
Binlerce komut ve değişken uzunluklu kodlama; çözücü donanımı karmaşıklaşır, kodu performans için optimize etmek zorlaşır.

Güç tüketimi
x86'nın evrimi güç tüketiminde önemli bir artış getirdi; pil ömrünün kritik olduğu aygıtlarda büyük bir sorun.

Isı yayılımı
İşlemciler güçlendikçe daha çok ısınır; soğutma maliyeti ve gürültü artar.

Maliyet ve lisans
x86 mimarisi Intel'e aittir; yalnızca lisans anlaşması olan firmalar (ör. AMD) üretebilir, rekabet sınırlıdır.

Bu nedenlerle telefon ve tabletlerde **ARM** (RISC) baskındır; x86 masaüstü ve sunucuda güçlüdür.

Saat Hızı Tek Başına Performans Değildir

$$\textbf{CPU süresi} = N \times CPI \times T = \textbf{N} \times \textbf{CPI} / \textbf{f}$$

Tanımlar

N: çalışan komut sayısı · CPI: komut başına ortalama saat turu · f: saat frekansı (T = 1/f).

Saat Hızı Tek Başına Performans Değildir

$$\textbf{CPU süresi} = N \times CPI \times T = \textbf{N} \times \textbf{CPI} / \textbf{f}$$

Tanımlar

N: çalışan komut sayısı · CPI: komut başına ortalama saat turu · f: saat frekansı (T = 1/f).

İşlemci A: f = 2 GHz, CPI = 2

N = 10^9 → süre = $10^9 \times 2 / (2 \times 10^9)$ = **1 s**

Saat Hızı Tek Başına Performans Değildir

CPU süresi = N × CPI × T = **N × CPI / f**

Tanımlar

N: çalışan komut sayısı · CPI: komut başına ortalama saat turu · f: saat frekansı (T = 1/f).

İşlemci A: f = 2 GHz, CPI = 2

N = 10^9 → süre = $10^9 \times 2 / (2 \times 10^9)$ = **1 s**

İşlemci B: f = 1,5 GHz, CPI = 1

süre = $10^9 \times 1 / (1{,}5 \times 10^9) \approx$ **0,67 s** → saati düşük olan B **%50 daha hızlı**!

Saat Hızı Tek Başına Performans Değildir

CPU süresi = N × CPI × T = **N × CPI / f**

Tanımlar

N: çalışan komut sayısı · CPI: komut başına ortalama saat turu · f: saat frekansı (T = 1/f).

İşlemci A: f = 2 GHz, CPI = 2

N = 10^9 → süre = $10^9 \times 2 / (2 \times 10^9)$ = **1 s**

İşlemci B: f = 1,5 GHz, CPI = 1

süre = $10^9 \times 1 / (1{,}5 \times 10^9) \approx$ **0,67 s** → saati düşük olan B **%50 daha hızlı**!

Sonuç

Saat hızı "saniyedeki tur sayısı"dır, komut sayısı değil. Komut/s = f / CPI; mimari (boru hattı, önbellek, süperskalar) CPI'yı düşürür.

MODERN ÖZELLİKLER

Modern İşlemci Özellikleri (1)

Saat hızı (clock speed)

Saniyedeki saat turu sayısı (GHz). Komut hızı ayrıca CPI'ya bağlıdır.

Komut kümesi mimarisi (ISA)

CISC ve RISC gibi mimariler verimlilik ve karmaşıklığı etkiler; yazılım ile donanım arasındaki sözleşmedir.

Önbellek (cache)

Sık kullanılan veriyi işlemcinin yanında tutan küçük ve çok hızlı bellek (L1, L2, L3).

Çok çekirdek (multi-core)

Birden çok görevin gerçekten aynı anda yürütülmesini sağlar, toplam performansı artırır.

Sanallaştırma (virtualization)

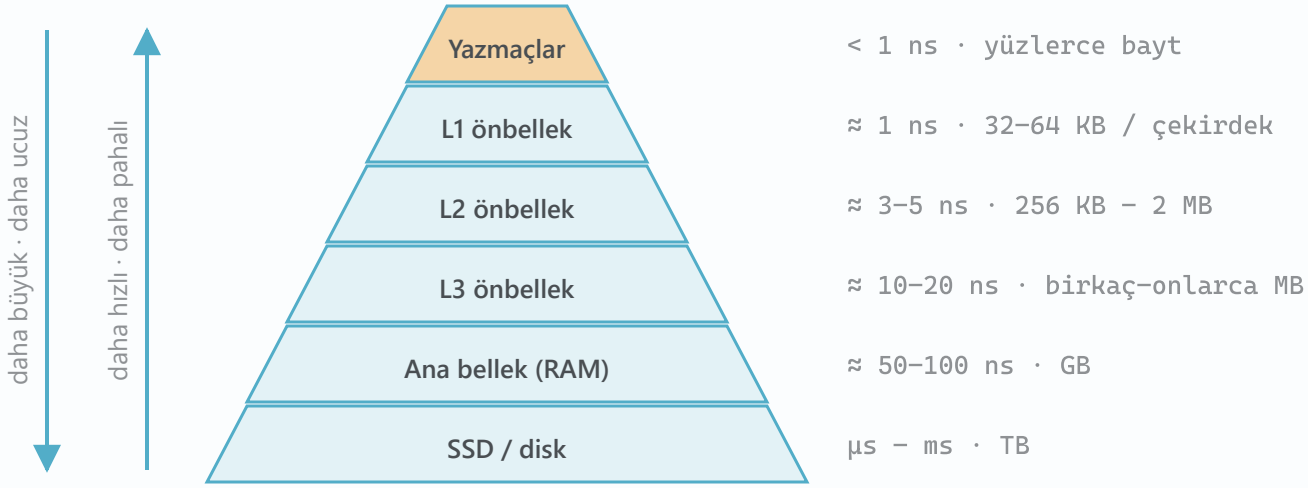
Aynı donanım üzerinde birden çok işletim sisteminin (sanal makine) yalıtılmış olarak çalışmasını sağlar (Intel VT-x, AMD-V).

Güç yönetimi

Yük azalınca saat ve gerilim düşürülür (DVFS), boştaki çekirdekler uyutulur.

MODERN ÖZELLİKLER

Önbellek ve Bellek Hiyerarşisi



Programlar aynı ve yakın veriye tekrar tekrar erişir (**yerellik ilkesi**). Önbellek bu veriyi yakında tutarak ortalama erişim süresini RAM'inkinin çok altına indirir. Süreler yaklaşıktır.

MODERN ÖZELLİKLER

Modern İşlemci Özellikleri (2)

Güç yönetimi (power management)

Modern işlemciler güç tüketimini azaltır ve enerji verimliliğini artırır: Turbo/SpeedStep, uyku durumları (C-states).

Grafik işleme

Entegre grafik işlem birimi (iGPU) içerir; grafik yoğun görevler daha hızlı ve verimli işlenir.

Güvenlik özellikleri

Donanım düzeyinde şifreleme (AES-NI), güvenli önyükleme (secure boot), yalıtılmış güvenli bölgeler (TPM, SGX/TrustZone).

Bağlantı (connectivity)

Özellikle mobil ve gömülü yongalarda (SoC) Wi-Fi, Bluetooth ve Ethernet gibi yerleşik ağ birimleri.

Makine öğrenmesi

Yapay zekâ için özel birimler: NPU, matris hızlandırıcıları (Intel AMX), vektör nokta çarpım komutları.

Yonga üstü sistem (SoC)

CPU, GPU, NPU, bellek denetleyicisi ve G/Ç'nin tek yongada birleşmesi; mikrodenetleyici fikrine geri dönüş.

ÖZET

Özet

- Mikroişlemci, bellekteki komutları **getir–çöz–yürüt** döngüsüyle çalıştıran programlanabilir CPU yongasıdır; mikrodenetleyici bellek ve çevre birimlerini de aynı yongada taşır.
- CPU; **ALU**, **kontrol birimi**, **yazmaçlar** ve **yol arayüzünden** oluşur; dış dünyayla adres, veri ve kontrol yollarıyla konuşur.
- Periyot T = 1/f; çalışma süresi = N × CPI / f. Saat hızı tek başına performans ölçüsü değildir.
- **CISC** zengin komutlar, **RISC** basit komut + boru hattı, **EPIC** derleyici güdümlü paralellik sunar; vektör, SIMD, DSP, GPU ve sembolik işlemciler özel işlere yöneliktir.
- Intel 4004 (1971) → 8086 (1978, 16 bit, 1 MB) → 80386 (32 bit) → Core 2 (64 bit): geriye uyumluluk x86'nın en büyük gücüdür.
- Modern işlemciler önbellek, çok çekirdek, sanallaştırma, güç yönetimi ve özel hızlandırıcılarla performans kazanır.

Alıştırmalar

1. 5 MHz'lik bir 8086'da saat periyodu ve 4 T'lik bir yol döngüsünün süresi nedir?
2. 24 bitlik adres yolu kaç bayt belleği adresleyebilir? Hangi işlemcide vardır?
3. 5 aşamalı bir boru hattında 100 komut kaç turda biter? Boru hattısız kaç tur gerekir?
4. $N = 2 \times 10^9$ komutluk bir program CPI = 1,5 olan 3 GHz'lik işlemcide kaç saniye sürer?
5. Çamaşır makinesi kontrol kartı ve dizüstü bilgisayar için mikroişlemci mi mikrodenetleyici mi seçilir? Neden?
6. EPIC paketinde 128 bit nasıl paylaştırılır?

▶ **İpuçları / cevaplar**

Kaynaklar

- Tanenbaum ve Bos, *Modern Operating Systems*, 3. baskı, Pearson.
- Brey, *The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium …*, 8. baskı, Pearson.
- Stallings, *Computer Organization and Architecture*, 10. baskı, Pearson.
- Patterson ve Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufmann.
- Intel Corporation, *8086 16-Bit HMOS Microprocessor* veri sayfası ve Intel 64 and IA-32 Architectures Software Developer's Manual.

Son

Bölüm 1.1: Giriş

Sonraki bölüm: Bölüm 1.2: Sayı Sistemleri · Terim sözlüğü