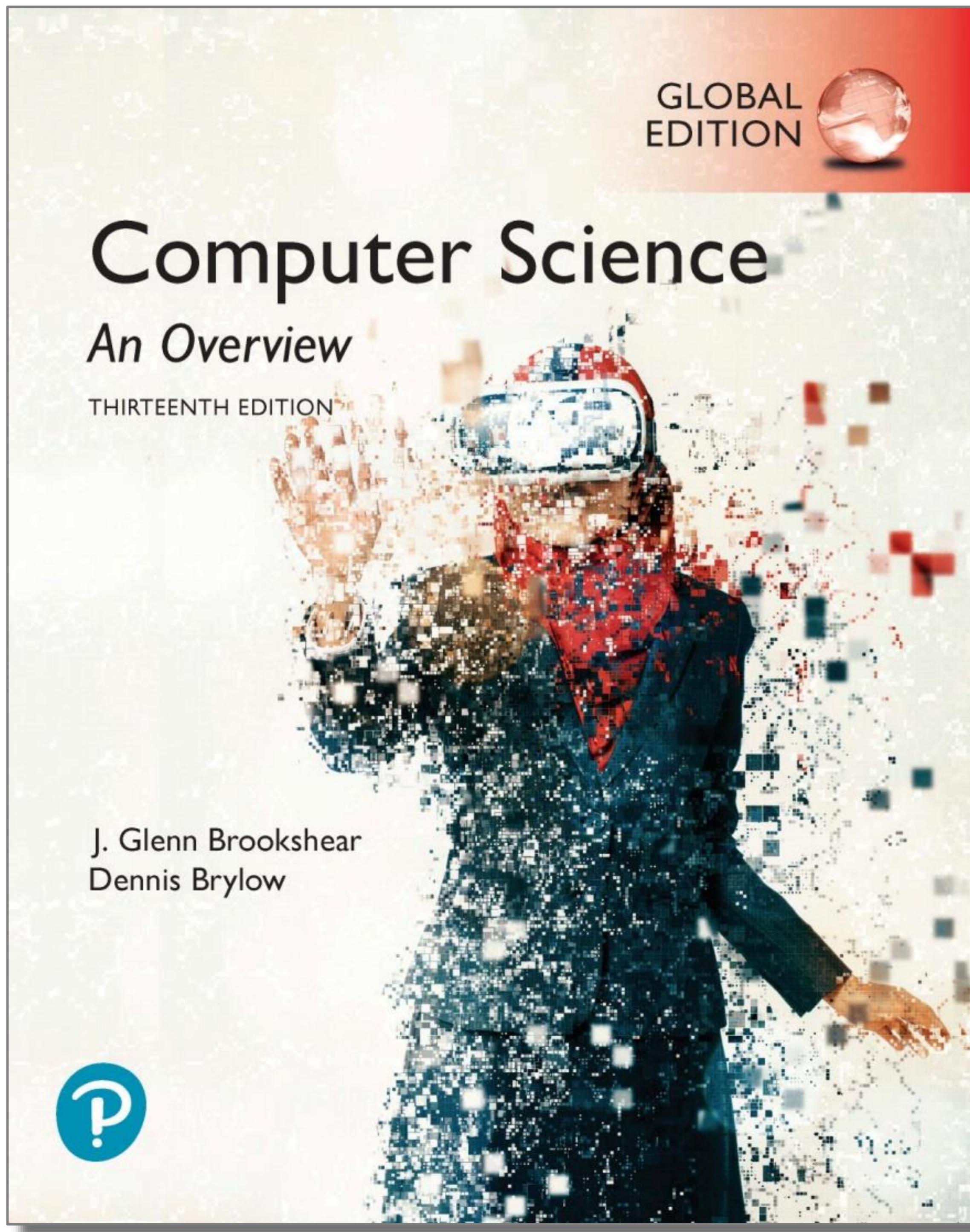


Bilgisayar Bilimine Giriş

13. Baskı, Global Edition



Bölüm 2

Veri İşleme

Bölüm 2: Veri İşleme

- 2.1 Bilgisayar Mimarisi
- 2.2 Makine Dili
- 2.3 Program Yürütme
- 2.4 Aritmetik/Mantık
- 2.5 Diğer Cihazlarla İletişim
- 2.6 Veri İşlemeyi Programlamak
- 2.7 Diğer Mimariler

2.1 Bilgisayar Mimarisi

Bir mikro izlenici 3 birimden oluşur

- Central Processing Unit (CPU) (Merkezi İşlem Birimi)
 - 1 Aritmetik/Mantık Birimi
 - 2 Control Birimi
 - 3 Yazmaç Birimi (Register Unit)
 - Genel yazmaçlar
 - Özel amaçlı yazmaçlar
- Veriyolu(Bus)
- Ana Bellek

işlençinin kendi içindeki bellek
alrı işlençi ile
ayrı mırda galisir

Ram
nizi
1 dir

işlençi
mırı
2 ise

Şekil 2.1 Veriyolu(Bus) ile birbirine bağlanmış Ana Bellek ve Merkezi İşlem Birimleri (CPU)

1 işlemci
1 RAM } Birçok
sayı

Merkezi işlem birimi

Aritmetik/Mantık
Birimleri

Kıyas, Toplam ...

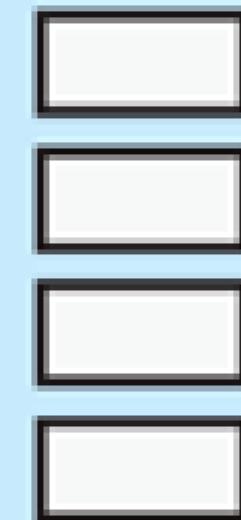
Her işlemde
singolinde
işlemci 1
calısır
over block
adlıysa
%10 da fast
orthanca
işlemci
yapar

işlerin sentan gitmesi için
singol gereklidir

Kontrol
Birimleri

Bir birimden diğer birime
giderken

Yazmaçlar



⋮
⋮



Veriyolu

Ana Bellek

En azından
bir
bir
bir
adımdan
iniş



Veriyolu işlemci mimarisine göre değişir

Laptoplarda → Mobil ıslenci → duvara göre ıslenci kültesini düşür

Mikro ıslenci

en karmaşık elektrik devresi

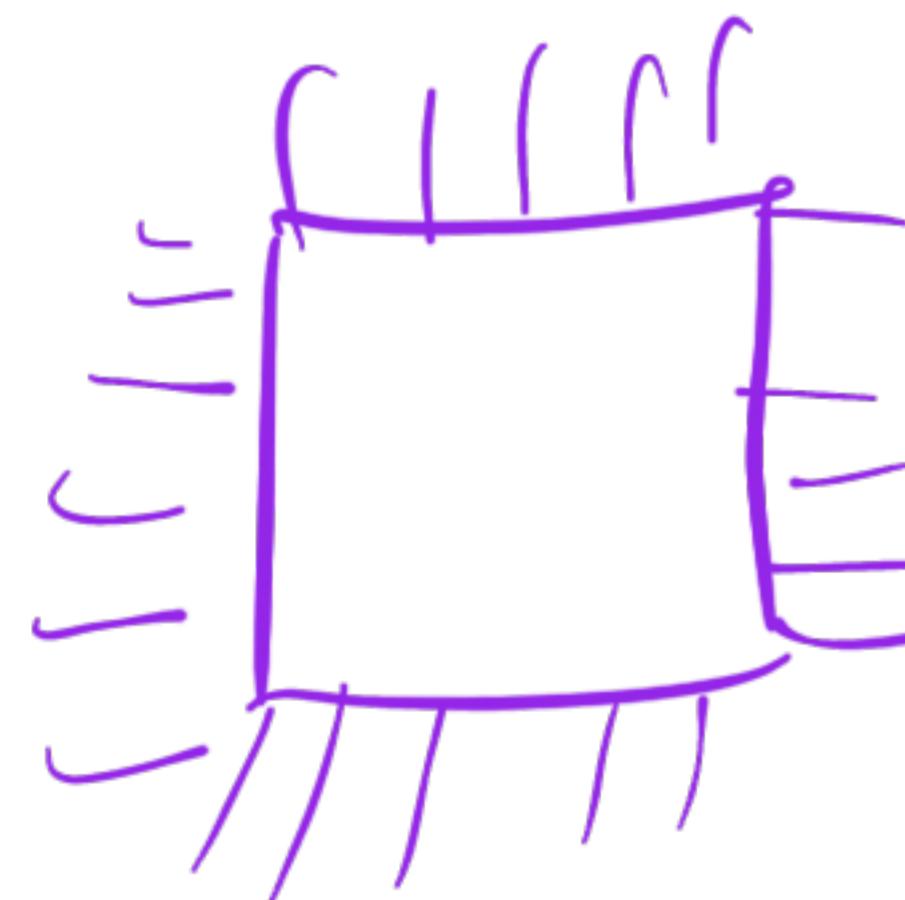
Pil
← → ıslenci
Ekran

Depolanmış Program Kavramı

Bir program bit desenleri şeklinde kodlanıp Ana Bellekte depolanabilir. Kontrol Birimi tarafından çıkartılabilir, çözülebilir ve çalıştırılabilir.

2.2 Makine Dili

- **Makine talimatı:** Bir talimat (yönerge) CPU tarafından çalıştırılabilir bir bit deseni olarak kodlanır.
- **Makine dili:** Makinenin anladığı bütün talimatların dizisidir.



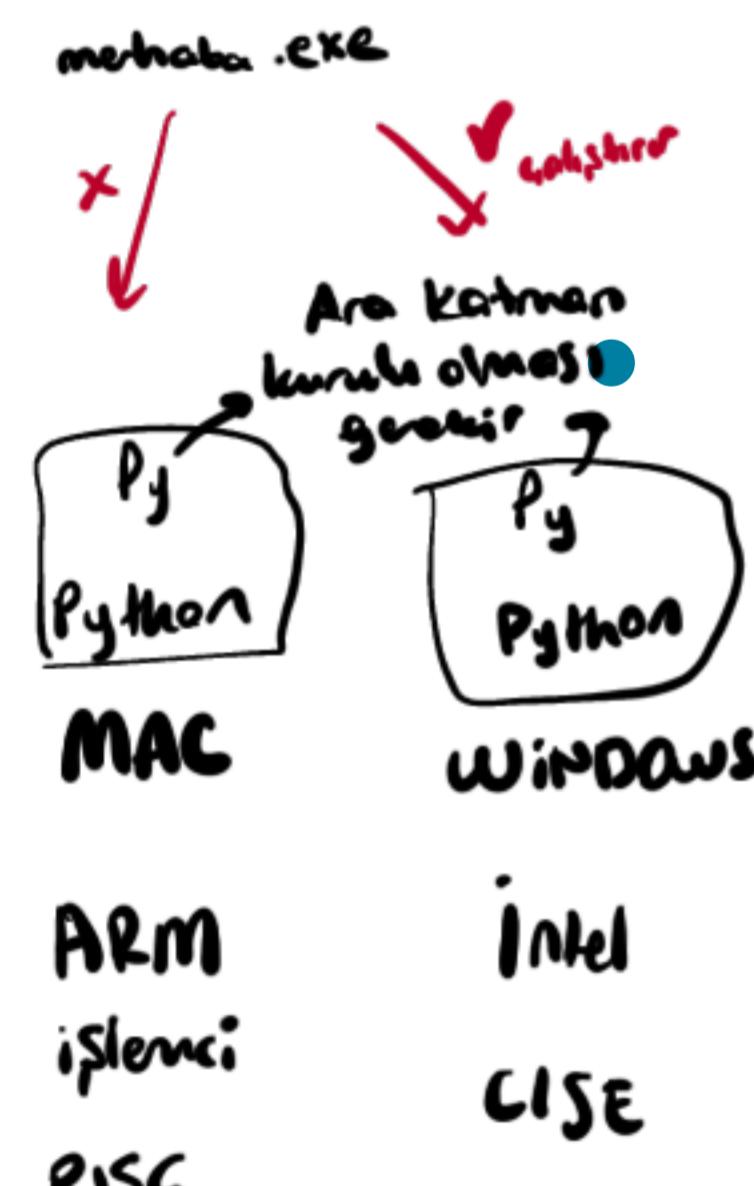
komut

C derleyip programı
↳ mikro izlenici komutları (makine Dili)

Bunlar birbiriyle uyumlu çalışmaz (Android'de seçtiğiniz
Intel'de çalışır)

Makine Dili Felsefeleri

- Azaltılmış Komut Kümeli Mikroişlemciler(Reduced Instruction Set Computing)(RISC) → **Değerlendirme:** Uzun kod
Avantaj: Dönerimsal Basit Ucuz Az Elektrik
- Az, basit, etkili ve hızlı komutlar
 - Örnekler: Apple/IBM/Motorola'nın PowerPC'si ve ARM



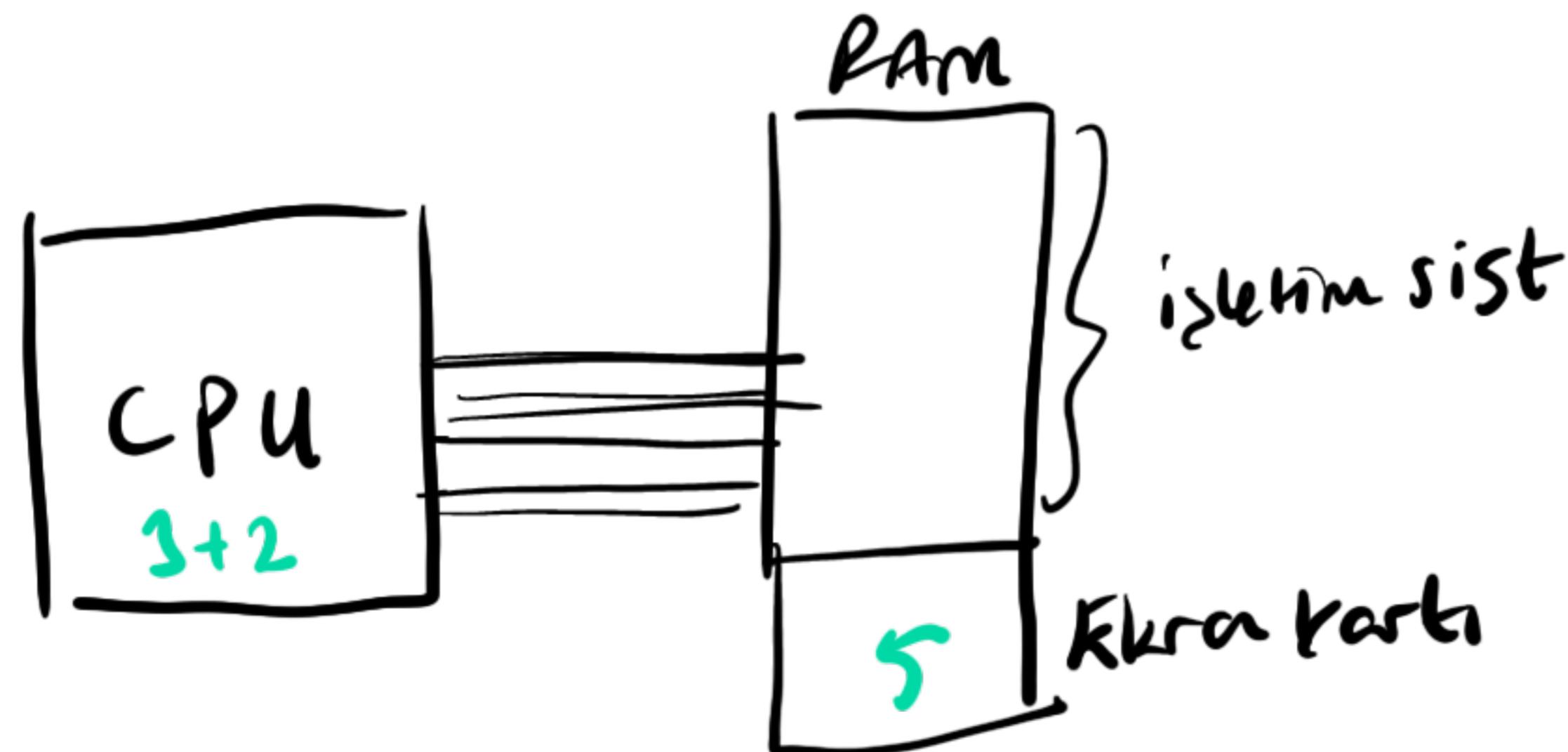
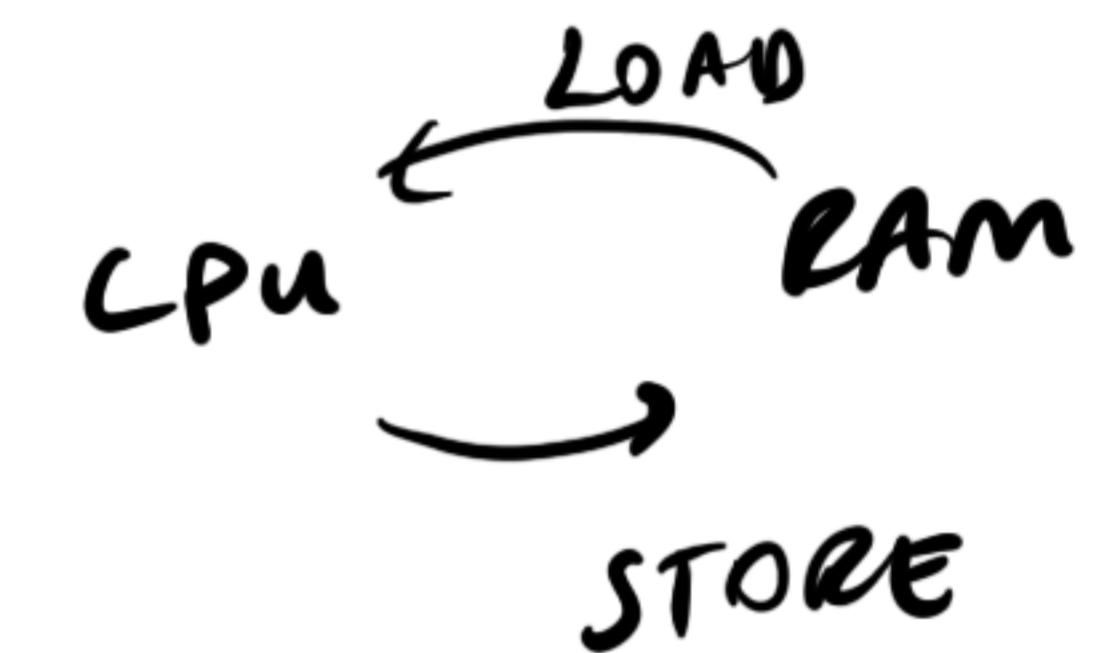
Karmaşık Komut Kümeli Mikroişlemciler(Complex Instruction Set Computing) (CISC)

- Fazla, kullanışlı ve güçlü komutlar
- Örnek: Intel

yüksek performans
Dönerimsiz karmaşık
pahalı
gök elektrik

Makine Komut Tipleri

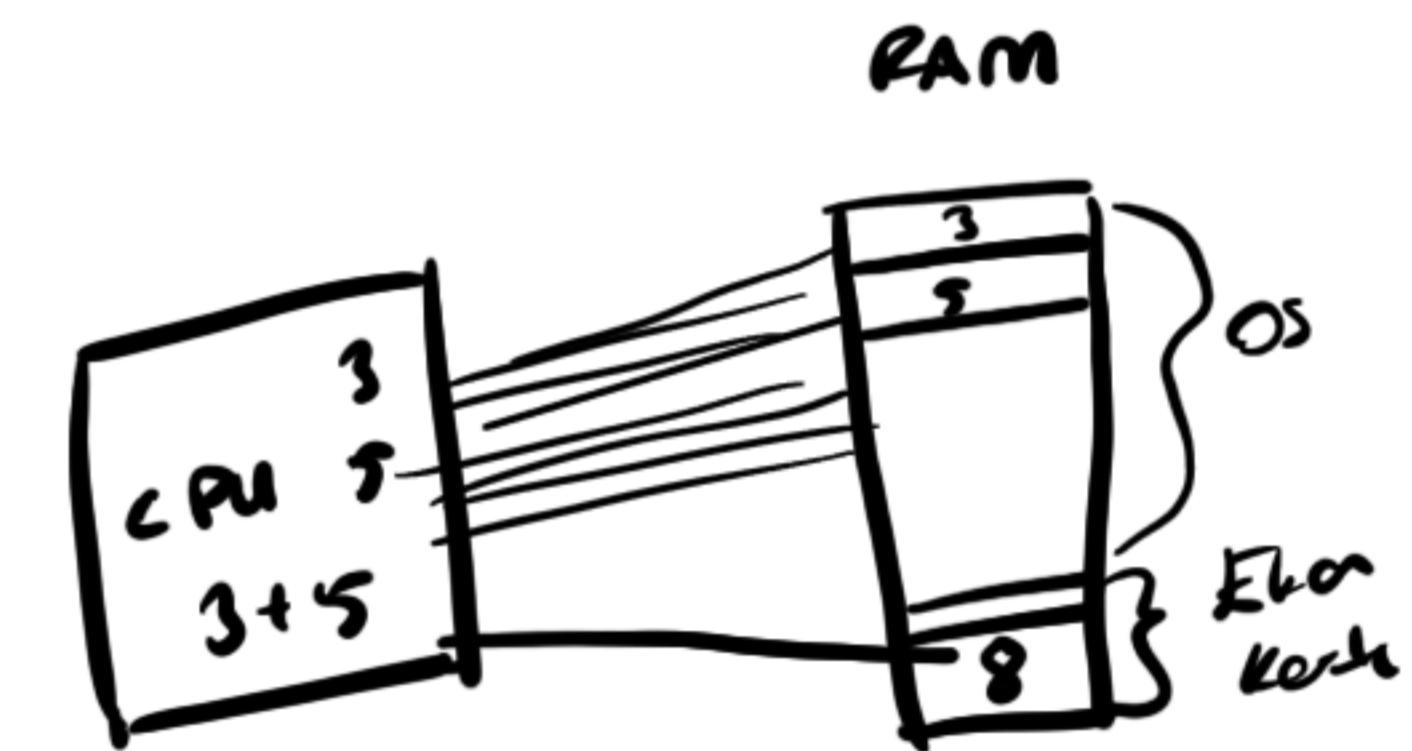
- Veri Transfer: Veriyi bir yerden başka bir yere aktarma(Örnek: LOAD, STORE)
- Aritmetik/Mantık: Bit desenlerinde işlemler (Örnek: +, -, *, /, AND, OR, SHIFT, ROTATE)
- Kontrol: Programın direkt çalıştırılması (Örnek: JUMP, BRANCH)



2.2 Bellekte depolanmış veride toplama

Bilginin önce RAM'de olması lazım.

- Adım 1.** Eklenecek değeri bellekten al ve yazmaca yerleştir.
- Adım 2.** Eklenecek diğer değeri bellekten al ve diğer yazmaca yerleştir.
- Adım 3.** Adım 1 ve 2'de girdi olarak kullanılan yazımcılardaki bilgileri alarak toplama devresini aktifleştir ve sonucu tutmak için tasarlanmış diğer yazmaca ver.
- Adım 4.** Sonucu bellekte depola.
- Adım 5.** Dur.



Şekil 2.3 Bellekte depolanmış değerde bölme

Adım 1. Bellekten bir sayıyı bir yazmaca YÜKLE

Adım 2. Bellekten başka bir sayıyı başka bir yazmaca YÜKLE

Adım 3. Eğer ikinci sayı 0 ise
Adım 6'ya ATLA

Adım 4. İlk yazmaçtaki sayıyı ikinci yazmaçtaki sayıya böl ve sonucu bir üçüncü *kalıcı da başka bir yazmaç ta tutabilir.* yazmaçta tut

Adım 5. Üçüncü yazmacın içeriğini belleğe KAYDET

Adım 6. Dur.

Makine komutunun bölümleri

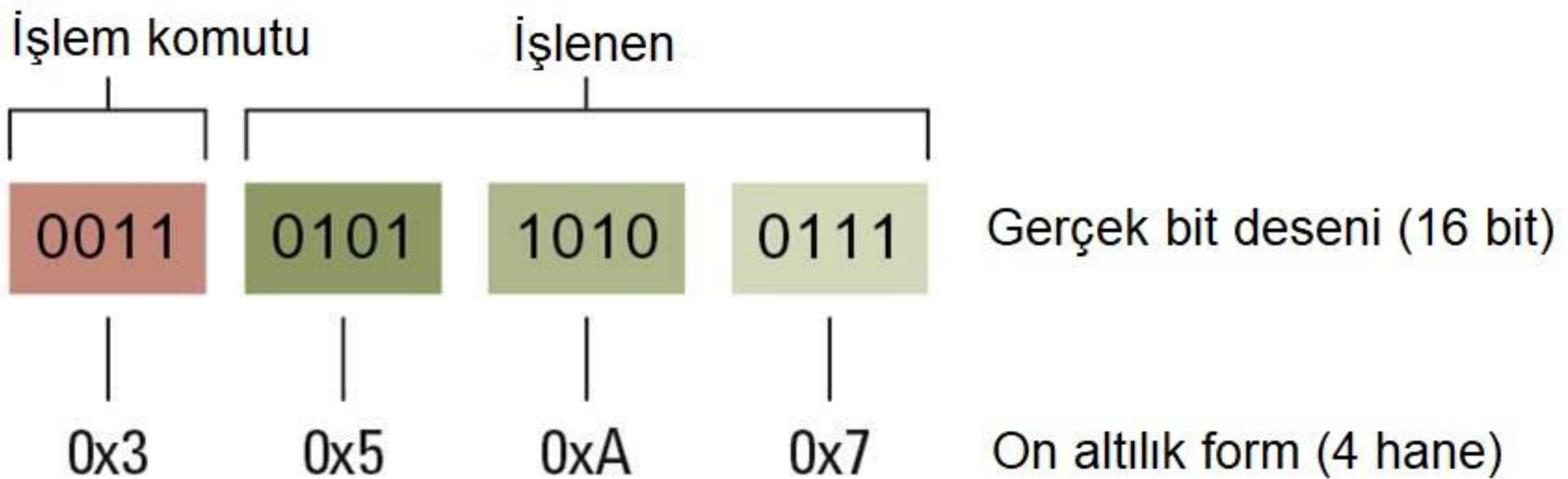
- **İşlem komutu(Op-code):** Uygulanacak işlemi belirtir
- **İşlenen(Operand):** İşlem hakkında daha kapsamlı bir bilgi verir
 - İşlenen değişkenler işlem komutuna göre yorumlanır

Topla = 3 + 2
~~~~ ~~~~~
Komut
(nisi) operand operand



Şekil 2.5 Bir Makine dili komutunun yapısı

a₆ $\overbrace{a_6 \quad a_7}^{0011}$



Mikro işlenci Simülatörü

lesil - Beltek

GRI - mikro işlenci

35 A7
ile

00110101001

komut

Store

işlenci belirtilen
nin beltek
bu hücresine
bölümünü yaz

→ A7 = 167
decimal

167.
sırası
koy

uyarı hatası sebebi nerede bittiğini söyle 0000

firewall

siber güvenlik
istil mekanları

The Breckhead Machine

1010

1111

5201

0000

(16'lik sayı)
sistemi

2.3 Program Çalıştırma

35A7

İşlenici ile yapmak
Kum taşları ile
yapmak

C Tuğlalar ile
yapmak

Python beton
blokları ile
yapmak

- İki özel amaçlı kayıt tarafından kontrol edilir
 - Komut yazmacı
 - Mevcut komutu tutar
 - Program sayacı
 - Sıradaki komutun adresini tutar (Sıradaki komutun adresini tutar)
her bir satırını işaret eder)
- Makine döngüsü: (bu üç adımı tekrarlar)
 - Bilgiyi getir, çözümle, yürüt(yap)

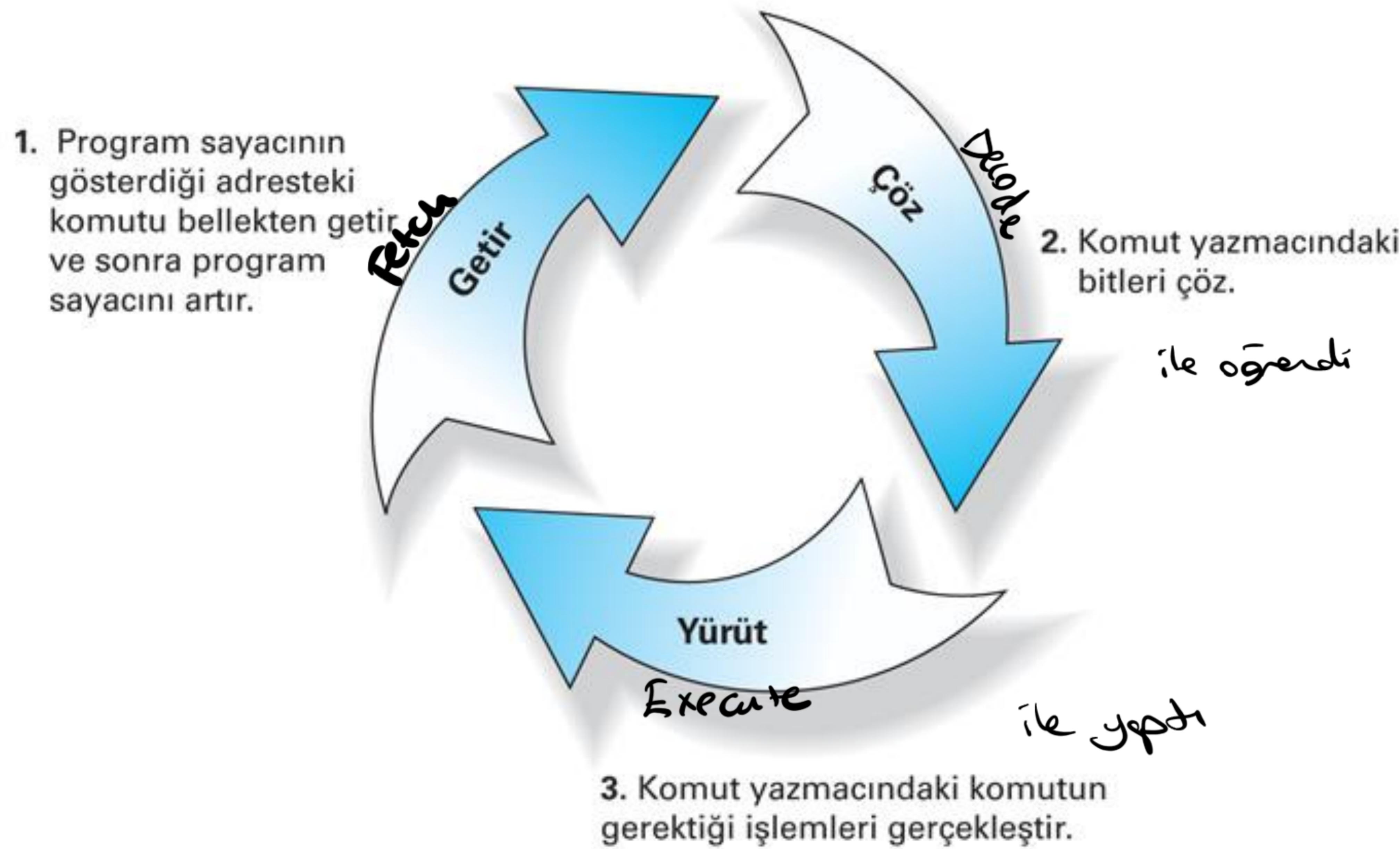
Fetch

Decode

Execute

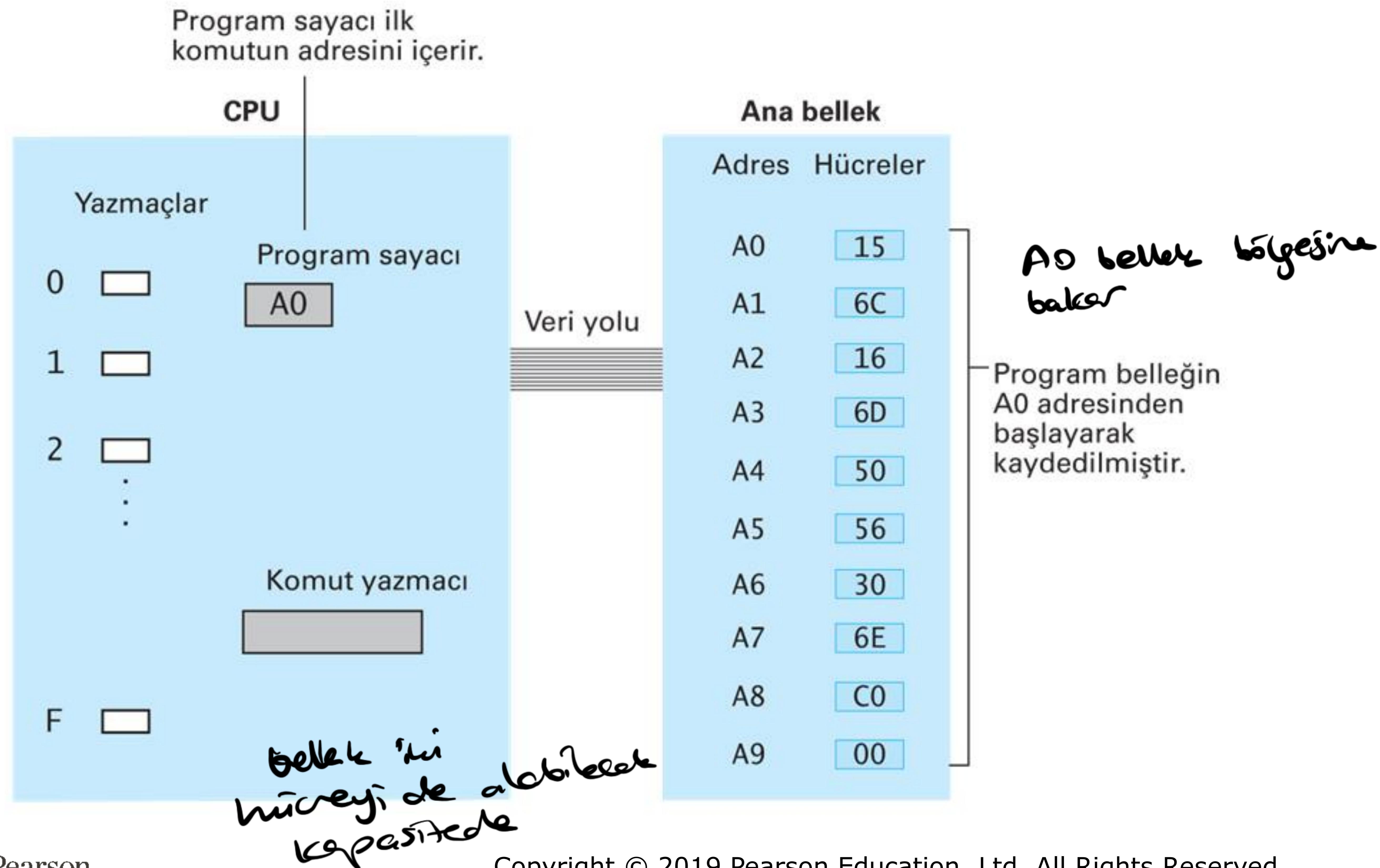
(
oaklı
bilgisi
komutu
antrenör

Şekil 2.8 Makine döngüsü

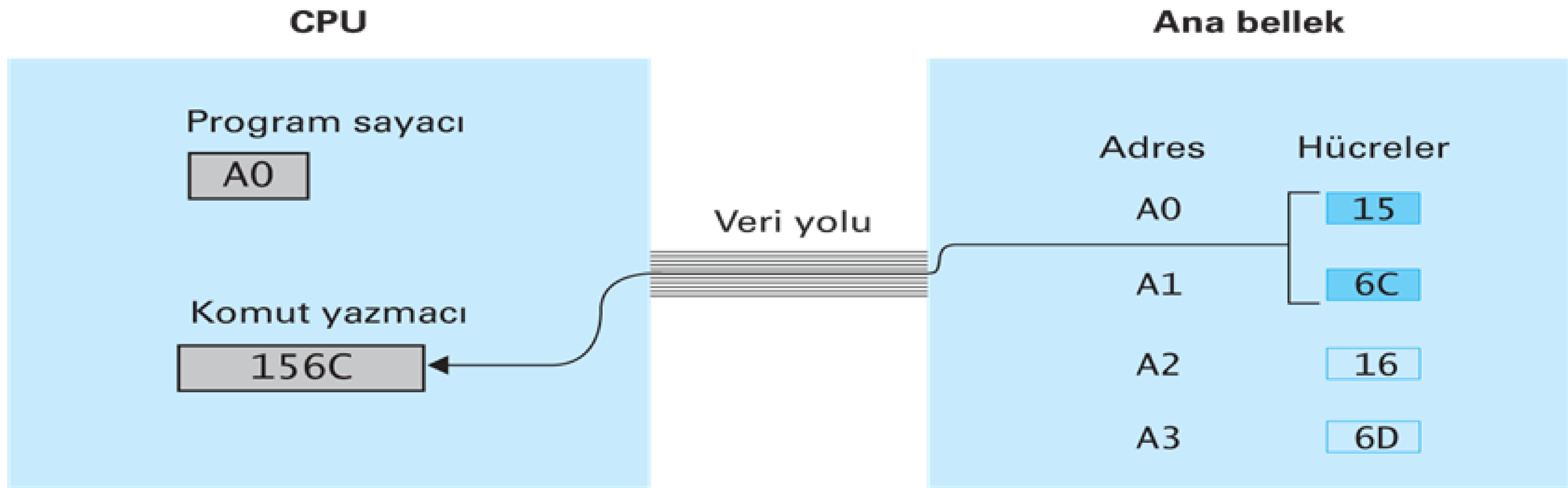


İşletim sistemi sonusunda doğrudan direktiğe ulaşır
mikro işlemci hizmet bu döngü hizmetine bağlanır

Şekil 2.10 Şekil 2.7'deki program ana bellekte depolanmış ve yürütülmeye hazır



Şekil 2.11 Makine döngüsünün bilgiyi alma adımlının işleyişi



- a. Getir adımlının başında A0 adresinde başlayan komut bellekten okunur ve komut yazmacına yerleştirilir.

Şekil 2.11 Makine döngüsünün bilgiyi alma adımlının işleyişi(devamı)



- b. Daha sonra program sayacı artırılarak bellekteki bir sonraki komutu

load
com → CPU

high level program
language

C, Python

with
compiler

machine code

Interpretation

after execute

compilation

before execute

High Level Code → Low Level Code

$Z = X + Y$

LDA X

ADD Y

STA Z

→

Assembly Code

LDA 3

ADD 4

STA 5

→ Machine
Code

1001 0011

1100 0000

1100 1111

2.4 Aritmetik/Mantık Komutları

- Mantık işlemleri:
 - VE(AND), VEYA(OR), Özel OR(XOR)
- Döndürme ve kaydırma işlemleri:
 - Dairesel kaydırma, mantıksal kaydırma, aritmetik kaydırma
- Aritmetik işlemler:
 - ekle, çıkar, çarp, böl

ARİTMETİK MANTIK Operasyonları

Mantık İşlemleri

"8" b1 üzerinde işler

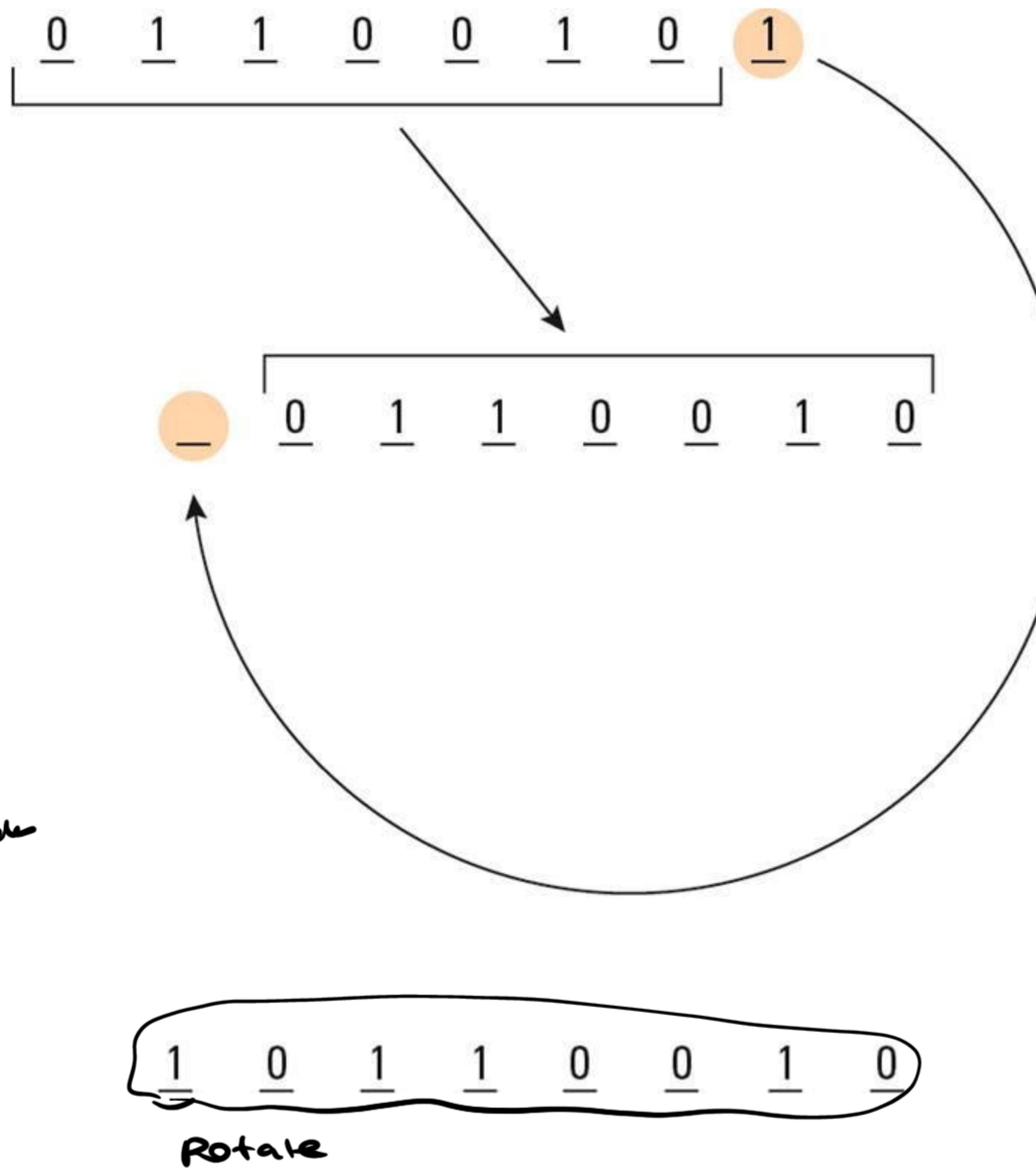
- Tek bir çıkış dizgesi oluşturmak için iki dizgenin bitlerini birleştiren **bitsel işlemlere** genişletilebilirler. İki dizgeyi bir mantık işlemeye sokmak için dizgelerin aynı sıralarında bulunan bitler basit mantık işlemeye sokulurlar.

$$\begin{array}{r} 10011010 \\ \text{AND } 11001001 \\ \hline 10001000 \end{array}$$

$$\begin{array}{r} 10011010 \\ \text{OR } 11001001 \\ \hline 11011011 \end{array}$$

$$\begin{array}{r} 10011010 \\ \text{XOR } 11001001 \\ \hline 01010011 \end{array}$$

Şekil 2.12 0x65 bit desenini sağa doğru bir bit kaydırma işlemi (Rightshift)



Orijinal bit deseni

Right shift yapmak
sayıyı 2ye bölmek denektir.

Bitler bir birim sağa kayıyor. En sağdaki bit bir uçtan "düşüyor" ve diğer uçtaki boşluğa

$$\underline{110} \rightarrow 6$$

$$\underline{011} \rightarrow 3$$

Öm: 5×3

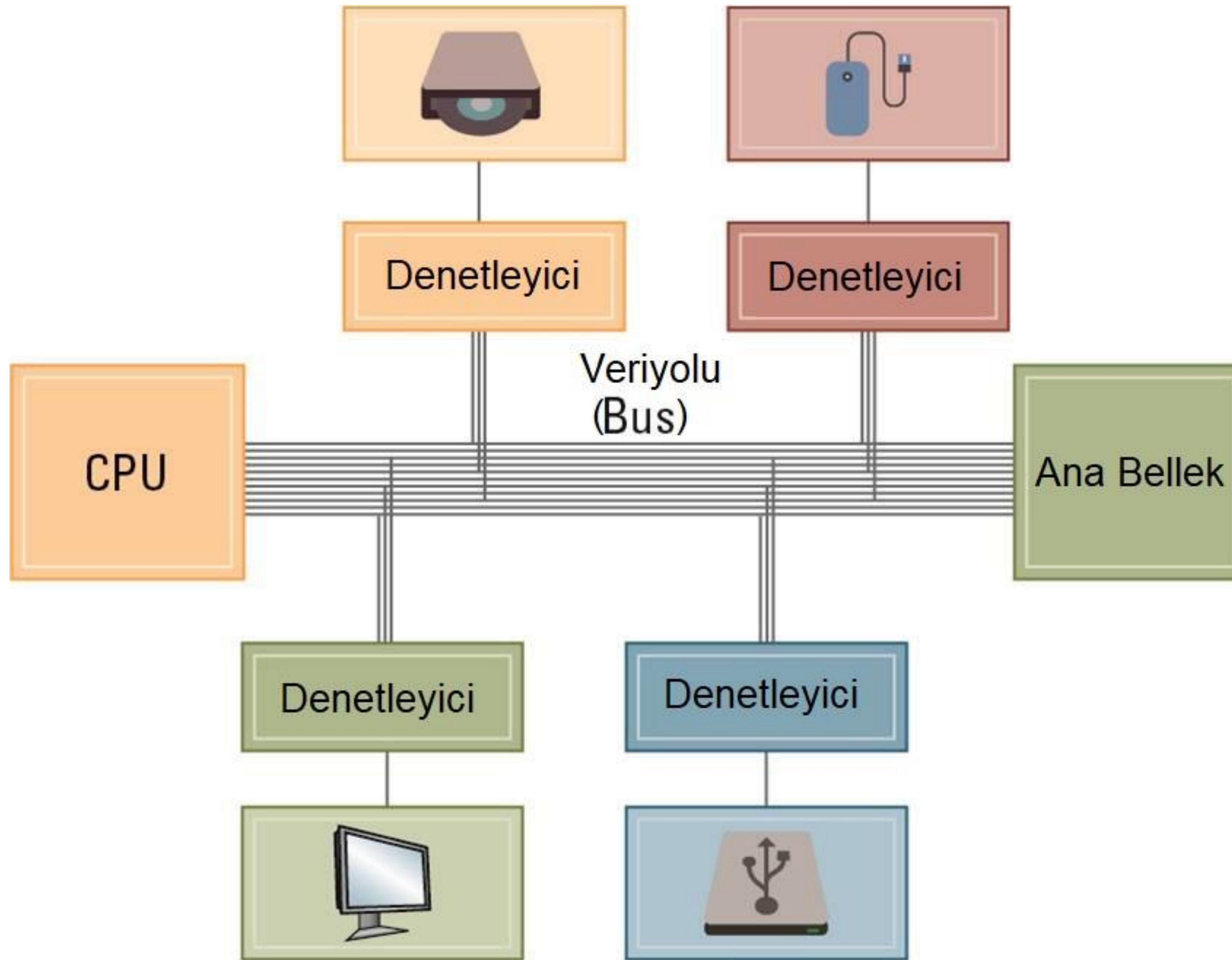
$$\begin{array}{r} 00000101 \\ + 00001010 \\ \hline 00001111 \end{array} \rightarrow 15 \text{ toplam}$$

Elde edilen bit deseni

2.5 Diğer aygıtlarla iletişim

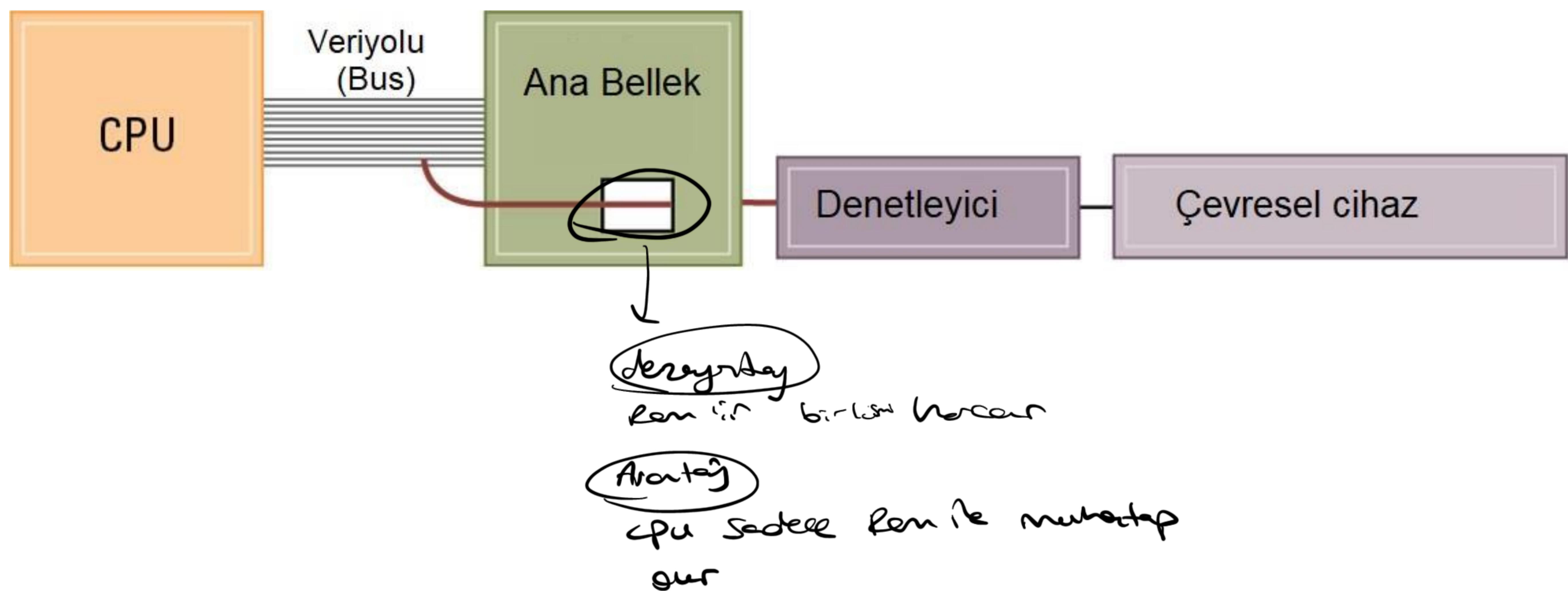
- **Denetleyici:** Bilgisayarın diğer aygıtlarla iletişimini yönetir
 - Özelleştirilmiş (aygıtın cinsine göre)
 - Genel amaçlı (USB, HDMI)
- **Kapı (Port):** Aygıtın bilgisayara bağlandığı yer
-Liman-
- **Bellek-haritalı giriş/çıkış:** CPU'ya bellek konumuymuş gibi gözüken aygıtlardır

Şekil 2.13 Makinenin veriyoluna bağlı Denetleyiciler



Şekil 2.14 Bellek-haritalı giriş/çıkış'ın görsel bir gösterimi

Memory - mapped I/O



Diğer aygıtlarla iletişim (devamı)

- **Direkt bellek erişimi (DMA):** Ana belleğin veriyolu üzerindeki bir denetleyici tarafından erişimi
- **El sıkışma:** Bilgisayar ve çevresel cihaz arasındaki veri transferi süreci *bönder ve Ailen arası bitti hızla veri transferi*



Bilg. Bağlantı Cihazları'nın Bilgi ile iletişimini

Diger aygitlarla iletisim (devamı)

- Popüler iletisim medyası

- Paralel iletisim: Çoklu sinyalin aynı anda, her biri ayrı “hatta” gitmesi (bilgisayarın iç veriyolu)
- Seri iletisim: Sinyallerin teker teker aynı ve tek “hat” üzerinden gitmesi (USB, FireWire)

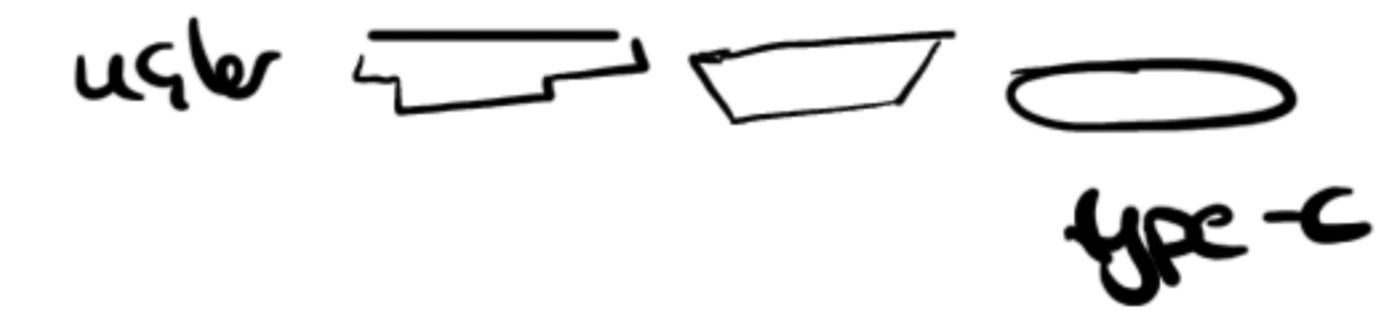
Az kullanım

Kablo maliyetleri:
donanım karmaşıklığı

Bilg. içinde
kullanılır

dışında pek
kullanılmaz günümde

usb 1995



Veri İletişim Oranları

- Ölçü birimleri
 - bps: saniye başına gönderilen bit(bits per second)
 - Kbps: Kilo-bps (1,000 bps) *byte*
 - Mbps: Mega-bps (1,000,000 bps) *iciñ*
 - Gbps: Giga-bps (1,000,000,000 bps) *8'e
601*
- Bant-genişliği: Mümkün olan maksimum hız

Fiber kablolarla genişleyiş



2.6 Veri programlama işlemi

- Programlama dilleri kullanıcıyı ana makine dilinin karmaşıklığından kurtarır:
 - Bir tek Python ifadesi onlarca belki de yüzlerce makine komut haritası kaplayabilir
 - Programcının, işlemcinin hangi mimariyi kullandığını bilmesine gerek yoktur(RISC yada CISC)

Python Kodlama dilinde bit düzene problemleri

AND

```
print(bin(0b10011010 & 0b11001001))  
# '0b10001000' Çıktısı verir
```

Montagel
C'de

&&
||

OR

```
print(bin(0b10011010 | 0b11001001))  
# '0b11011011' Çıktısı verir
```

NOT

```
print(~bin(0b10011010 ^ 0b11001001))  
# '0b1010011' Çıktısı verir
```

ilk 2 biti
negates

Kontrol yapıları

- If ifadesi:

```
if (su_sicakligi > 140):  
    print('Su çok sıcak!')
```

- While ifadesi:

```
while (n < 10):  
    print(n)  
    n = n + 1
```

Fonksiyonlar

- **Fonksiyon:** Verilen parametre(ler) üzerinde gösterilmesi gereken işlem serisine verilen addır
- **Fonksiyon çıktısı:** İfadedeki fonksiyonun görünümü

```
x = 1034
```

```
y = 1056
```

```
z = 2078
```

```
biggest = max(x, y, z)
```

```
print(biggest) # '2078' yazdırır
```

Fonksiyonlar (devamı)

- **Argüman değeri:** Parametreye girilmiş değer
- Fonksiyonlar bir değere **geri döner**
- **void fonksiyonları**, veya **prosedürler**, bir değere geri dönmezler

```
sideA = 3.0
sideB = 4.0
# Pisago teoremini kullanarak üçüncü kenarın
uzunluğunu hesaplar
hypotenuse = math.sqrt(sideA**2 + sideB**2)
print(hypotenuse)
```

Giriş / Çıkış

veri tipi olmadığı
için fonksiyon şeklinde

```
# Nizami bir üçgenin hipotenüsünü hesaplar
import math

# Kenar uzunluklarının girişini, ilk deneme
sideA = int(input('A kenarının uzunluğu
nadir?')) sadece metin olerek alır.
sideB = int(input('B kenarının uzunluğu
nadir?'))

# Üçün kenarı Pisagor Teoremi ile hesaplar
hypotenuse = math.sqrt(sideA**2 + sideB**2)

print(hypotenuse)
```

Maraton İdman Programı

```
# Maraton idman yardımcısı.  
  
import math  
  
# Bu fonksiyon dakikaları ve saniyeleri  
# sadece saniye şekline dönüştürür.  
def total_seconds(min, sec):  
    return min * 60 + sec  
  
# bu fonksiyon saatte alınan mil bilgisini kullanıp bir  
milin kaç saniyede alınacağını hesaplar  
def speed(time):  
    return 3600 / time
```

oluşt. bir fonk. formül

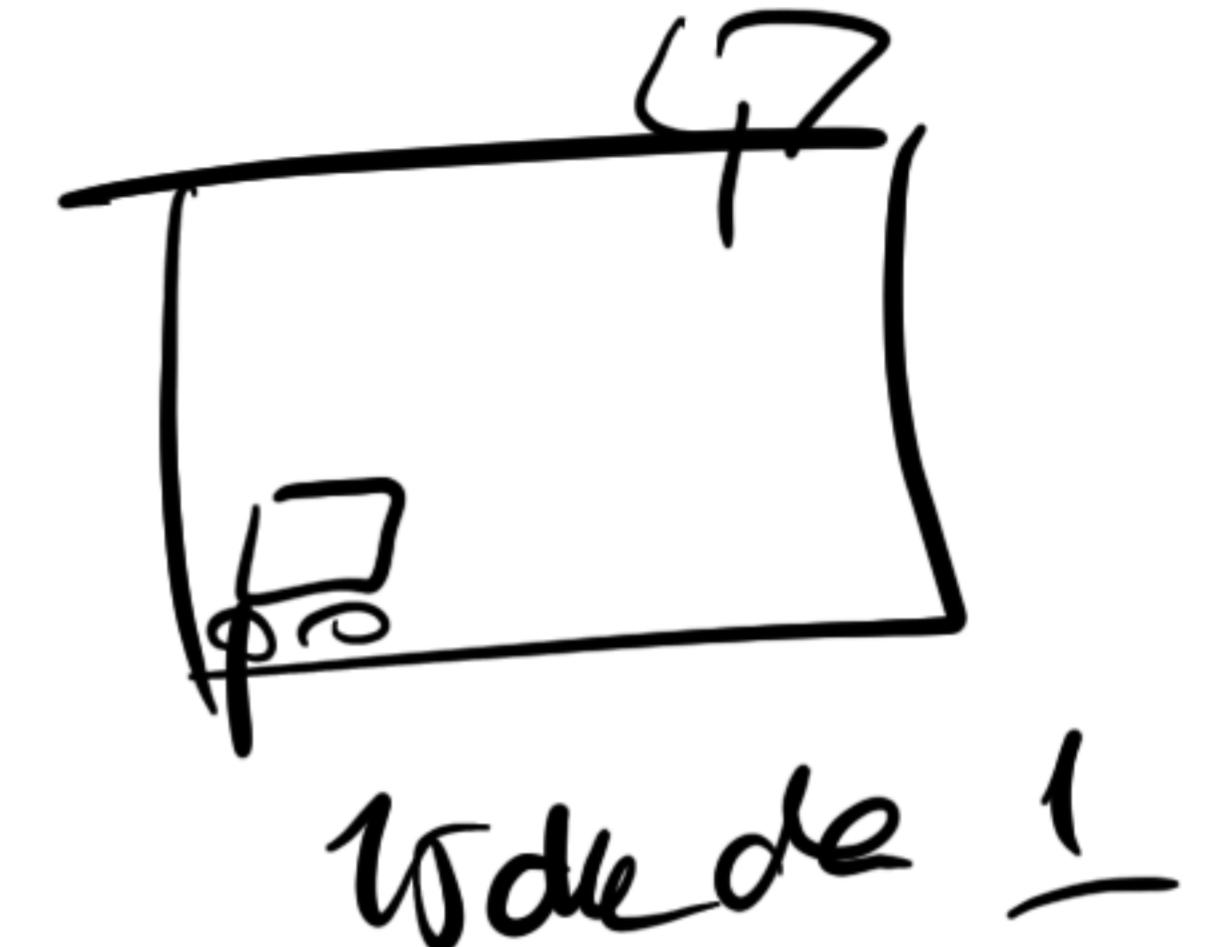
Maraton İdman Programı(devamı)

```
# Kullanıcıdan bir mil için gerekli süreyi ve mesafeyi  
al.  
pace_minutes = int(input('Minutes per mile? '))  
pace_seconds = int(input('Seconds per mile? '))  
miles = int(input('Total miles? '))  
  
# Hızı hesapla ve yazdır.  
mph = speed(total_seconds(pace_minutes, pace_seconds))  
print('Hızınız ' + str(mph) + ' mph'dır')  
  
# Çalışma için ayrılmış zamanı hesapla.  
total = miles * total_seconds(pace_minutes, pace_seconds)  
elapsed_minutes = total // 60  
elapsed_seconds = total % 60  
  
print('Çalışma süreniz ' + str(elapsed_minutes) +  
      ' dakikadır ' + str(elapsed_seconds) + '  
secs')
```

Şekil 2.15 Örnek maraton idman verileri

Bir mil için geçen süre				Toplam geçen süre		
Dakika	Saniye	Mil	Hız (mil/saat)	Dakika	Saniye	
9	14	5	6.49819494584	46	10	
8	0	3	7.5	24	0	
7	45	6	7.74193548387	46	30	
7	25	1	8.08988764044	7	25	

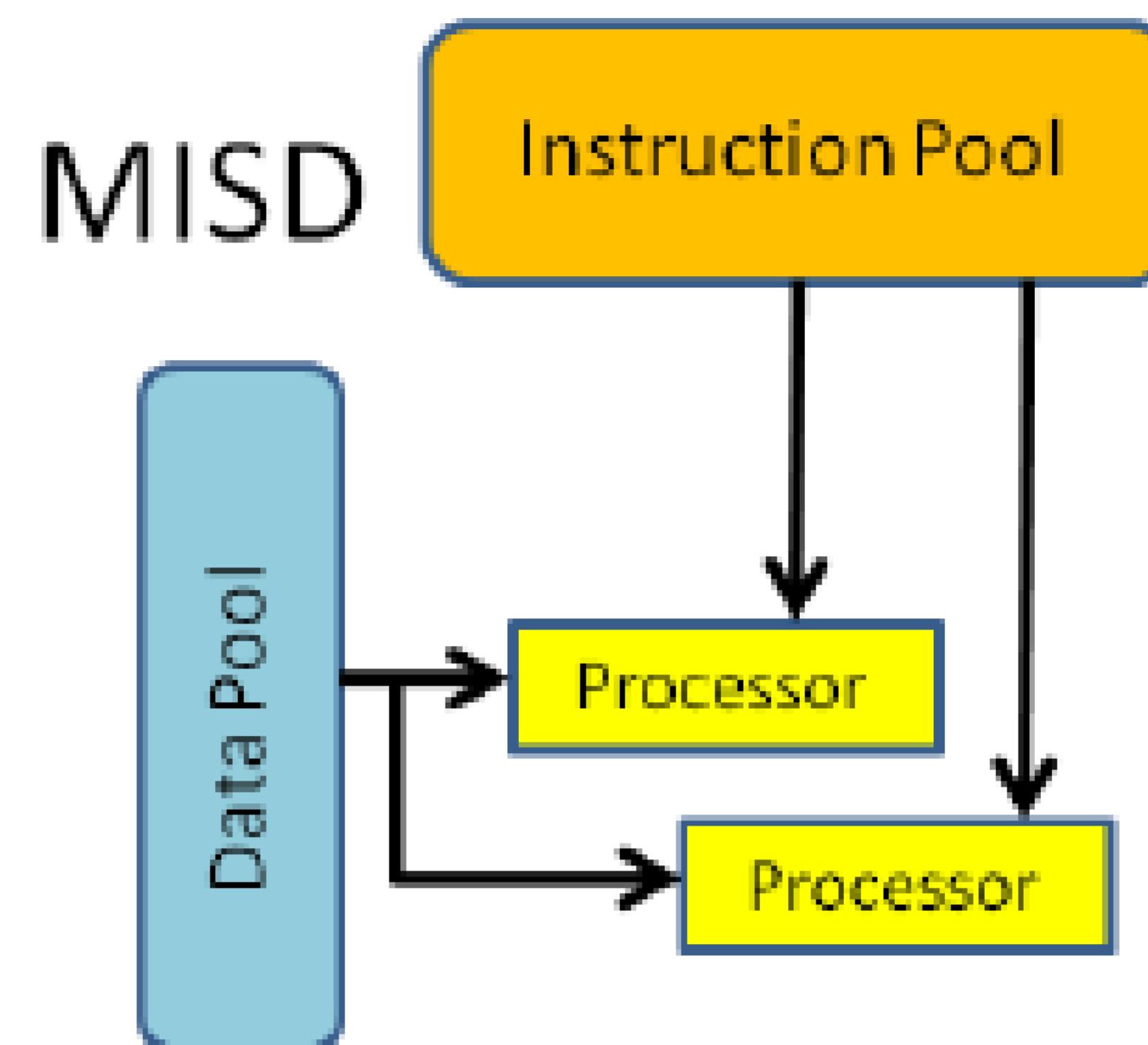
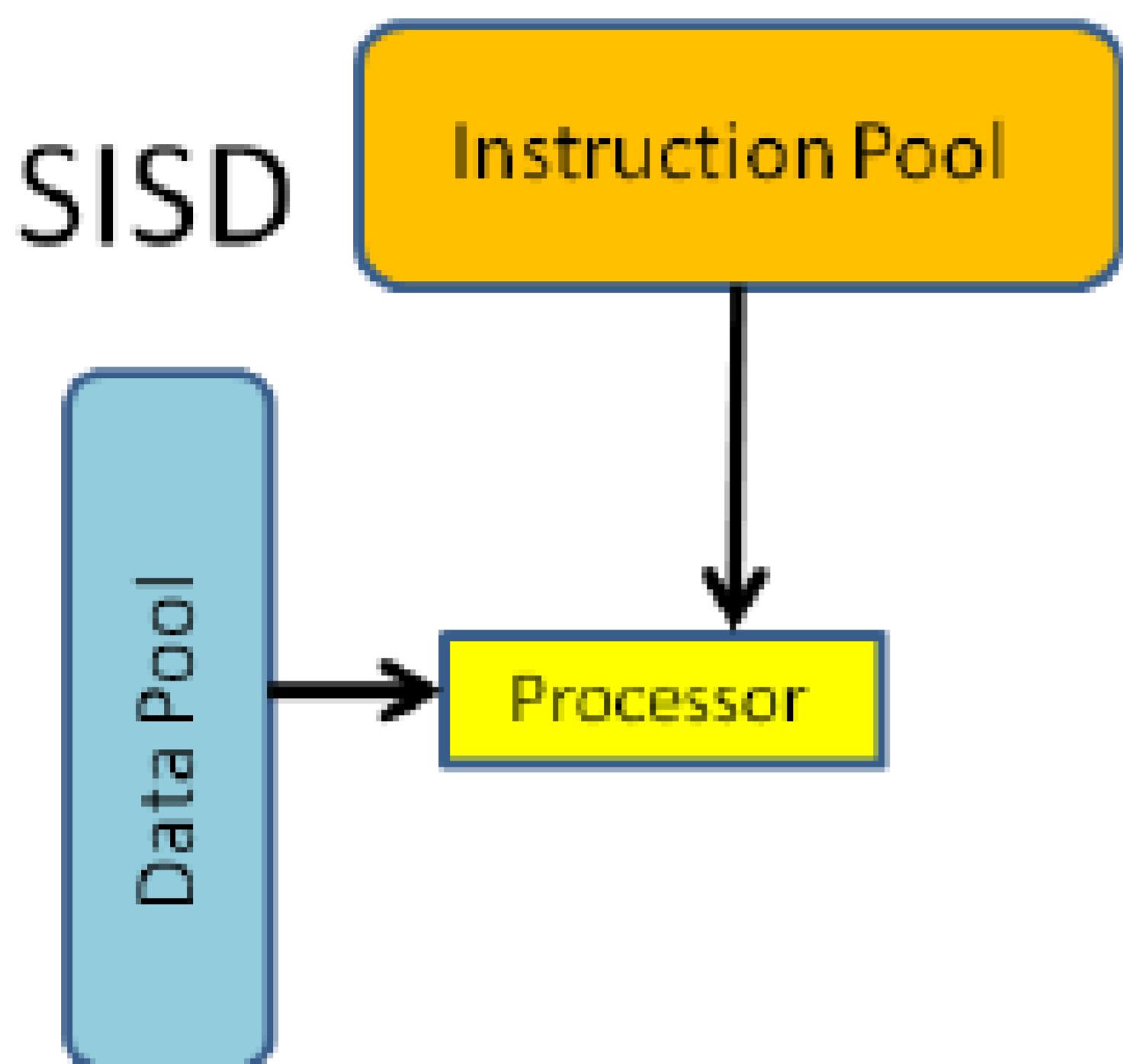
Paralel İşlemci



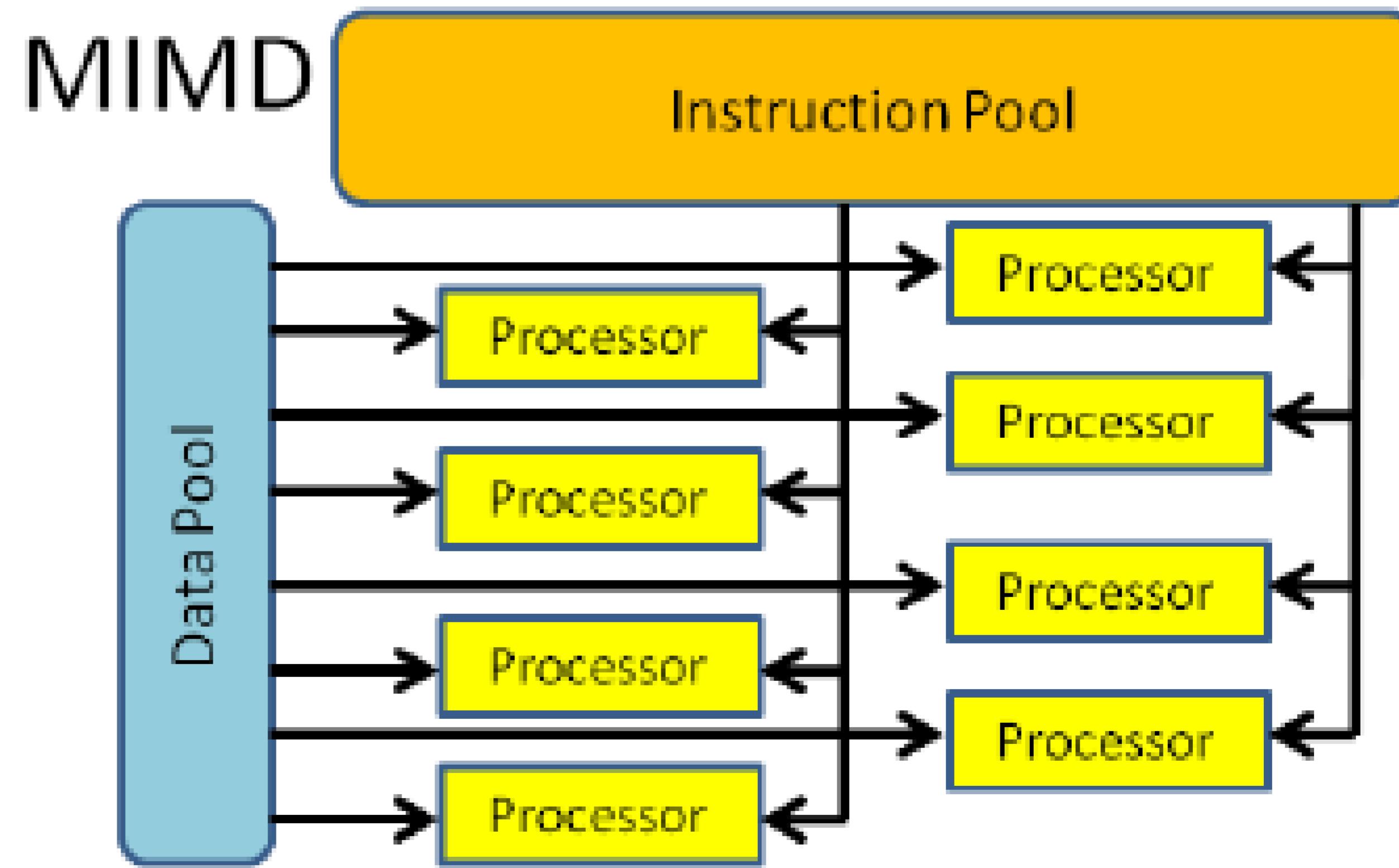
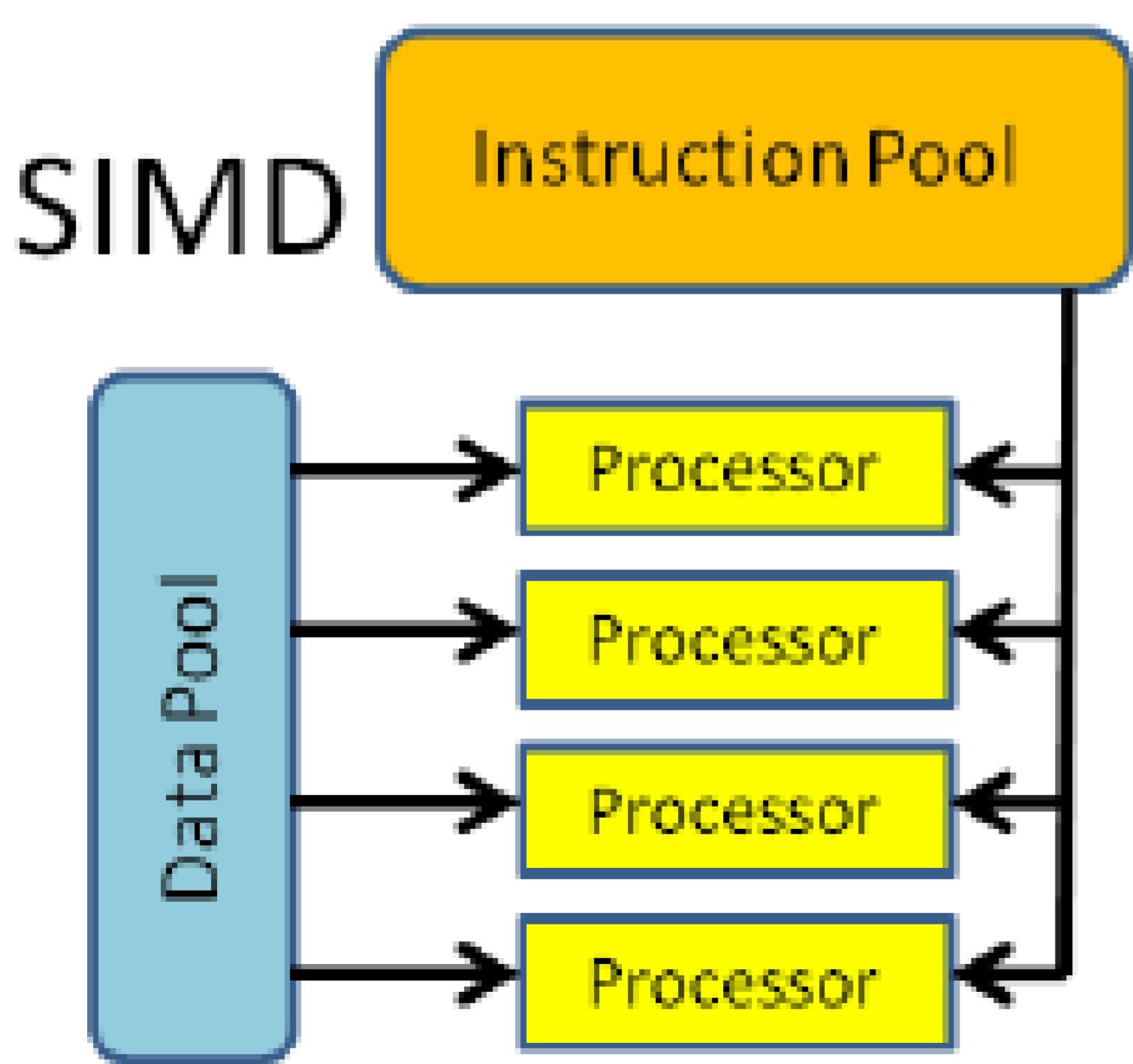
2.7 Diğer Mimariler

- Üretilen işi artırın teknolojiler:
 - Küme komut işleme (Pipelining): Makine döngülerinin birden fazlasının aynı anda çalıştırılması (Sıra hatti)
 - Paralel işleme: Birbirine benzeyen birçok işlemci kullanır
 - SISD: Tek komut, Tek veri
 - Paralel işleme olmaz
 - MIMD: Çoklu komut, Çoklu veri
 - Farklı programlar, farklı veri
 - SIMD: Tek komut, Çoğul veri
 - Aynı program, farklı veri
- Farklı işlemciler
paralel programlama
kodu nasıl birden fazla
işlemciyi kullanarak çalışracığını
ve sonra nasıl toplayeceğini
gösterir.

Mikroişlemci, iyi bilmeden paralel hesaplamayı biliyoruyız



İşleniciler aynı veri havuzundan



Aynı kod ve forth