Bölüm 1.1: Bilgisayarlar

Java ile Nesne Yönelimli Programlama

Temel parçalar | İkili sayılar | İkiye tümleyen | Kesirli sayılar | ASCII ve Unicode | Dizgiler | Saklanan program | von Neumann mimarisi

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Donanım

İşlemci, bellek, disk ve veri yolunun görevini açıklamak; bir programın işlemcide Getir–Çöz–Yürüt döngüsüyle nasıl yürüdüğünü izlemek.

Sayılar

Sayıları ikili, sekizli, onaltılı ve ondalık tabanlar arasında çevirmek; ikili toplama ve ikiye tümleyen ile negatif sayıları hesaplamak.

Metin

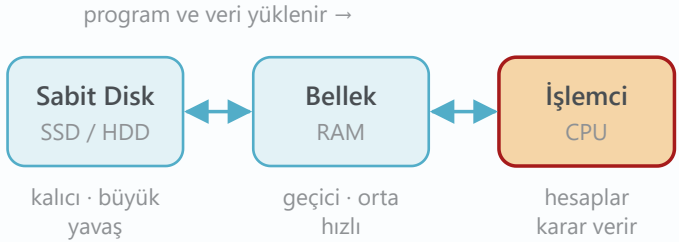
ASCII, Unicode ve UTF-8 arasındaki farkı açıklamak; bir dizginin bellekte sayılarla nasıl temsil edildiğini göstermek.

Java ile bağlantı

`byte` ve `int` sınırlarını ve taşmayı ikiye tümleyenle açıklamak; Java'da tabanlarla çalışmak; assembly komutlarının makine koduna karşılığını göstermek.

Bir Bilgisayarın Temel Parçaları

- **İşlemci** (CPU, *Central Processing Unit*): bilgisayarın "beyni"; tüm **aritmetik** ve **mantıksal** hesaplamaları yapar.
- **Bellek** (RAM, *Random Access Memory*): çalışan programı ve verilerini **geçici** olarak saklar. Kapasitesi ve erişim hızı, bilgisayarın hızını doğrudan etkiler.
- **Sabit disk** (HDD / SSD): verileri **kalıcı** olarak saklar; elektrik kesilince silinmez.



Mutfak benzetmesi: aşçı = işlemci, tezgâh = RAM (küçük ama hemen elinin altında), kiler = disk (büyük ama uzakta). Aşçı malzemeyi kilerden tezgâha alır, orada çalışır.

Bilgisayarın İşleyişi ve Programlama

Temel adımlar:

1. İşlemciye **talimatlar** (komutlar) gönderilir.
2. İşlemci talimatları **sırayla** işler.
3. Sonuçlar kullanıcıya veya diğer yazılımlara sunulur.

Programlama, bilgisayarın istenen görevi yerine getirmesi için ona anlayacağı biçimde talimatlar yazma sürecidir.

İlk Java programımız: tek bir talimat, “ekrana yaz”.

```
public class MerhabaDunya {
    public static void main(String[] args) {
        System.out.println("Merhaba, Dünya!");
    }
}
```

Çıktı:

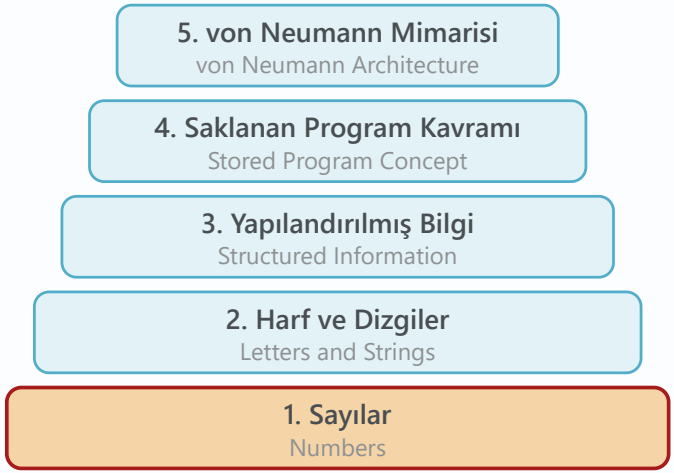
```
Merhaba, Dünya!
```

Java gibi bir dilde yazılan talimatlar, işlemcinin anladığı makine koduna çevrilerek çalıştırılır.

Bir Bilgisayar Nasıl İnşa Edilir?

Bilgisayarı katman katman kuracağız. Her katman bir alttakinin üzerine oturur:

- Önce elektrik sinyallerini **sayılara** dönüştürürüz.
- Sayılarla **harfleri**, harflerle **metinleri** temsil ederiz.
- Sayı dizileriyle **resim, ses ve film** gibi yapılandırılmış bilgiyi saklarız.
- **Komutları da** sayı olarak bellekte saklarız.
- Hepsini **von Neumann mimarisi** bir araya getirir.

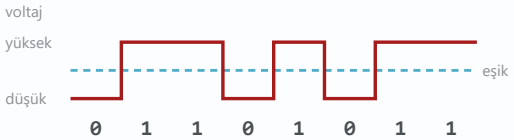


Elektrik Sinyalinden Bite

- Bilgisayarın temel işi, **elektrik sinyallerini** belirli kurallara göre işlemektir; bu kurallar sayesinde pek çok işlev gerçekleşir.
- Sinyalleri insanın anlayacağı **sayı ve sembollerle** ilişkilendirince bilgisayar bize anlamlı bilgi sunabilir.
- Sinyaller **ikili** (binary, taban 2) temsille işlenir: her basamak, yani **bit** (*binary digit*), yalnızca 0 ya da 1 olabilir.
- Bütün işlemler bu ikili temelin üzerine inşa edilmiştir.

0	1
Yanlış (false)	Doğru (true)
Kapalı	Açık
Düşük voltaj	Yüksek voltaj

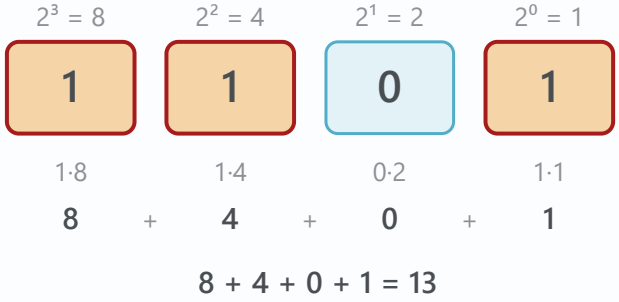
Neden iki durum? Bir telde “voltaj var / yok” ayrımını gürültüye rağmen güvenle yapmak, on farklı voltaj düzeyini ayırt etmekten çok daha kolaydır.



8 bit = 1 **bayt** (byte). Bellek ve dosya boyutları bayt ile ölçülür: 1 KB ≈ 1000 bayt, 1 KiB = 1024 bayt.

Basamak Değeri: $13 = 1101_2$

- Tam sayılar, **2'nin kuvvetlerinin** toplamı olarak yazılır. Her kuvvet 0 ya da 1 kez kullanılır.
- 4 ardışık kuvvetle (2^0, 2^1, 2^2, 2^3) **0–15** arasındaki tam sayılar temsil edilir.
- $13_{10} = 1{\cdot}2^3 + 1{\cdot}2^2 + 0{\cdot}2^1 + 1{\cdot}2^0 = \mathbf{1101_2}$



Diğer tabanlarda aynı kural:

Ondalık (10)	$603_{10} = 6{\cdot}10^2 + 0{\cdot}10^1 + 3{\cdot}10^0$
Sekizli (8)	$207_8 = 2{\cdot}8^2 + 0{\cdot}8^1 + 7{\cdot}8^0 = 135_{10}$
Onaltılı (16)	$FF_{16} = 15{\cdot}16^1 + 15{\cdot}16^0 = 255_{10}$

Alt simge tabanı gösterir: 1101_2 “bir bir sıfır bir, ikili” diye okunur, “bin yüz bir” diye değil.

Herhangi Bir Taban ve Bit Sınırları

- Taban **b ≥ 2** ve **d** basamak ile 0 ile **$b^d - 1$** arasındaki her tam sayı yazılabilir.
- Basamakların alabileceği değerler: 0, 1, …, **b − 1** (ikilide 0–1, sekizlide 0–7, onaltılıda 0–9 ve A–F).
- 64 bitlik bir bilgisayar, işaretsiz olarak $2^{64} - 1$ = **18.446.744.073.709.551.615**'e kadar sayabilir.

Bit	En büyük ($2^d - 1$)
4	15
8	255
16	65.535
32	4.294.967.295

Java'nın tam sayı tipleri **işaretlidir**: bitlerin yarısı negatiflere ayrılır.

```
System.out.println(Byte.MAX_VALUE);    // 8 bit
System.out.println(Short.MAX_VALUE);   // 16 bit
System.out.println(Integer.MAX_VALUE); // 32 bit
System.out.println(Long.MAX_VALUE);    // 64 bit
// 64 bitin hepsi 1, işaretsiz okunursa:
System.out.println(Long.toUnsignedString(-1L));
```

Çıktı:

```
127
32767
2147483647
9223372036854775807
18446744073709551615
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi ÷ 2 kalan

ikili ""

`sayi` = 13 ile başlıyoruz.

Adım 1 / 16

DEĞİŞKENLER

- sayi = **13**
- ikili = **–**
- kalan = **–**

ÇIKTI

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
▶    String ikili = "";
4    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi ÷ 2 kalan

ikili ""

Sonuç dizgisi boş başlar: `ikili = ""`.

Adım 2 / 16

DEĞİŞKENLER

- `sayi = 13`
- `ikili = ""`
- `kalan = –`

ÇIKTI

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
▶    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi ÷ 2 kalan

ikili ""

Koşul `13 > 0` doğru → döngüye gir.

Adım 3 / 16

DEĞİŞKENLER

- sayi = 13
- ikili = ""
- kalan = –

ÇIKTI

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
▶      int kalan = sayi % 2;
▶      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1

ikili `""`

Kalan `13 % 2 = 1` ; bu, sayının **en sağdaki** henüz yazılmamış bitidir.

Adım 4 / 16

DEĞİŞKENLER

- `sayi = 13`
- `ikili = ""`
- `kalan = 1`

ÇIKTI

```
13 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
▶      ikili = kalan + ikili;
▶      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1

ikili **1**

Kalan dizginin **önüne** eklenir → "1"; sayı yarıya iner: 13 / 2 = 6 (tam sayı bölmesi).

Adım 5 / 16

DEĞİŞKENLER

- `sayi = 6`
- `ikili = "1"`
- `kalan = 1`

ÇIKTI

```
13 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
▶    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1

ikili **1**

Koşul `6 > 0` doğru → döngüye gir.

Adım 6 / 16

DEĞİŞKENLER

- `sayi = 6`
- `ikili = "1"`
- `kalan = 1`

ÇIKTI

```
13 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
▶      int kalan = sayi % 2;
▶      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1
6	→ 3	0

ikili **1**

Kalan `6 % 2 = 0`; bu, sayının **en sağdaki** henüz yazılmamış bitidir.

Adım 7 / 16

DEĞİŞKENLER

`sayi = 6`

`ikili = "1"`

`kalan = 0`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
▶      ikili = kalan + ikili;
▶      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1
6	→ 3	0

ikili **0** 1

Kalan dizginin **önüne** eklenir → "01"; sayı yarıya iner: 6 / 2 = 3 (tam sayı bölmesi).

Adım 8 / 16

DEĞİŞKENLER

`sayi = 3`

`ikili = "01"`

`kalan = 0`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
▶    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1
6	→ 3	0

ikili **0** **1**

Koşul `3 > 0` doğru → döngüye gir.

Adım 9 / 16

DEĞİŞKENLER

- `sayi = 3`
- `ikili = "01"`
- `kalan = 0`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
▶      int kalan = sayi % 2;
▶      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1
6	→ 3	0
3	→ 1	1

ikili **0** **1**

Kalan `3 % 2 = 1` ; bu, sayının **en sağdaki** henüz yazılmamış bitidir.

Adım 10 / 16

DEĞİŞKENLER

`sayi = 3`

`ikili = "01"`

`kalan = 1`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
3 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
▶      ikili = kalan + ikili;
▶      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1
6	→ 3	0
3	→ 1	1

ikili **1 0 1**

Kalan dizginin **önüne** eklenir → "101"; sayı yarıya iner: 3 / 2 = 1 (tam sayı bölmesi).

Adım 11 / 16

DEĞİŞKENLER

`sayi = 1`

`ikili = "101"`

`kalan = 1`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
3 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
▶    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1
6	→ 3	0
3	→ 1	1

ikili **1** **0** **1**

Koşul `1 > 0` doğru → döngüye gir.

Adım 12 / 16

DEĞİŞKENLER

- `sayi = 1`
- `ikili = "101"`
- `kalan = 1`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
3 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
▶      int kalan = sayi % 2;
▶      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1
6	→ 3	0
3	→ 1	1
1	→ 0	1

ikili: 1 0 1

Kalan `1 % 2 = 1`; bu, sayının **en sağdaki** henüz yazılmamış bitidir.

Adım 13 / 16

DEĞİŞKENLER

- `sayi = 1`
- `ikili = "101"`
- `kalan = 1`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
3 % 2 = 1
1 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
▶      ikili = kalan + ikili;
▶      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```

sayi	÷ 2	kalan
13	→ 6	1
6	→ 3	0
3	→ 1	1
1	→ 0	1

ikili: 1 1 0 1

Kalan dizginin **önüne** eklenir → "1101"; sayı yarıya iner: 1 / 2 = 0 (tam sayı bölmesi).

Adım 14 / 16

DEĞİŞKENLER

- `sayi = 0`
- `ikili = "1101"`
- `kalan = 1`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
3 % 2 = 1
1 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
▶    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
10   System.out.println("Sonuç: " + ikili);
11 }
```



Koşul `0 > 0` yanlış → döngü biter. Kalanlar **aşağıdan yukarı** okunur.

Adım 15 / 16

DEĞİŞKENLER

`sayi = 0`

`ikili = "1101"`

`kalan = –`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
3 % 2 = 1
1 % 2 = 1
```

Ondalıktan İkiliye: 2'ye Bölme Yöntemi

```
1  public static void main(String[] args) {
2    int sayi = 13;
3    String ikili = "";
4    while (sayi > 0) {
5      int kalan = sayi % 2;
6      System.out.println(sayi + " % 2 = " + kalan);
7      ikili = kalan + ikili;
8      sayi = sayi / 2;
9    }
▶    System.out.println("Sonuç: " + ikili);
11 }
```



Sonuç yazdırılır: 13_{10} = **1101_2**.

Adım 16 / 16

DEĞİŞKENLER

`sayi = 0`

`ikili = "1101"`

`kalan = –`

ÇIKTI

```
13 % 2 = 1
6 % 2 = 0
3 % 2 = 1
1 % 2 = 1
Sonuç: 1101
```

SAYILAR

Java'da Taban Dönüşümleri

```
System.out.println(Integer.toBinaryString(13));  // ikili
System.out.println(Integer.toOctalString(135));  // sekizli
System.out.println(Integer.toHexString(255));    // onaltılı
System.out.println(Integer.parseInt("1101", 2)); // 2 → 10
System.out.println(Integer.parseInt("207", 8));  // 8 → 10
System.out.println(0b1101 + " " + 0207 + " " + 0xFF);
```

Çıktı:

```
1101
207
ff
13
135
13 135 255
```

`0b` öneki

İkili sabit: `0b1101` = 13 (Java 7+).

`0` öneki

Sekizli sabit: `0207` = 135. `010` on değil, **8**'dir!

`0x` öneki

Onaltılı sabit: `0xFF` = 255. Renk ve bayt değerlerinde sık kullanılır.

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

	1
	57
+	28
	85

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.

elde								
57	0	0	1	1	1	0	0	1
28 +	0	0	0	1	1	1	0	0
85								

İki sayıyı alt alta yazarız: 57 = 00111001_2 ve 28 = 00011100_2. Toplama **sağdan** başlar.

Adım 1 / 9

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

```
   1
  57
+ 28
----
  85
```

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.



Sütun 1: 1 + 0 = 1 → 1 yaz, elde yok.

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

```
   1
  57
+ 28
----
  85
```

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.

elde								
57	0	0	1	1	1	0	0	1
28 +	0	0	0	1	1	1	0	0
85							0	1

Sütun 2: 0 + 0 = 0 → 0 yaz, elde yok.

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

```
   1
  57
+ 28
----
  85
```

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.

elde								
57	0	0	1	1	1	0	0	1
28 +	0	0	0	1	1	1	0	0
85						1	0	1

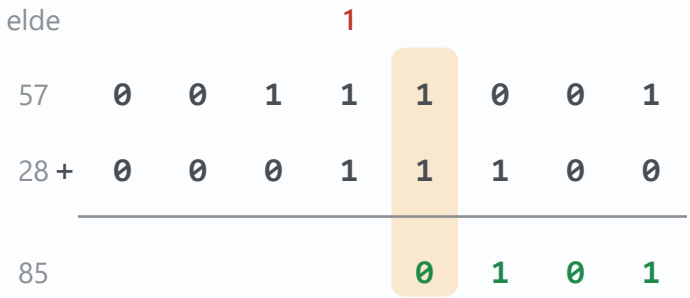
Sütun 3: 0 + 1 = 1 → 1 yaz, elde yok.

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

```
   1
  57
+ 28
----
  85
```

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.



Sütun 4: 1 + 1 = 2 = **10_2** → 0 yaz, **1 elde**.

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

```
   1
  57
+ 28
----
  85
```

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.

elde			1	1				
57	0	0	1	1	1	0	0	1
28 +	0	0	0	1	1	1	0	0
85				1	0	1	0	1

Sütun 5: 1 + 1 + 1 (elde) = 3 = $\mathbf{11_2}$ → 1 yaz, **1 elde**.

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

```
   1
  57
+ 28
----
  85
```

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.

elde		1	1	1				
57	0	0	1	1	1	0	0	1
28 +	0	0	0	1	1	1	0	0
85			0	1	0	1	0	1

Sütun 6: 1 + 0 + 1 (elde) = 2 = **10_2** → 0 yaz, **1 elde**.

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

```
   1
  57
+ 28
----
  85
```

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.

elde		1	1	1				
57	0	0	1	1	1	0	0	1
28 +	0	0	0	1	1	1	0	0
85		1	0	1	0	1	0	1

Sütun 7: 0 + 0 + 1 (elde) = 1 → 1 yaz, elde yok.

Toplama Her Tabanda Aynı Kurallarla Yapılır

Hangi taban kullanılırsa kullanılsın geçerli ilkeler: **basamak değerleri**, **elde** (taşıma, *carry*) ve doğru sonuç.

	1
	57
+	28
	85

Onlukta 7 + 8 = 15 → 5 yaz, 1 elde. İkilide 1 + 1 = 2 = 10_2 → 0 yaz, 1 elde.

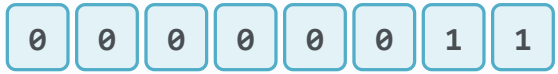
elde		1	1	1				
57	0	0	1	1	1	0	0	1
28 +	0	0	0	1	1	1	0	0
85	0	1	0	1	0	1	0	1

Sütun 8: 0 + 0 = 0 → 0 yaz, elde yok. Sonuç: 01010101_2 = 64 + 16 + 4 + 1 = **85** ✓

Negatif Sayılar: İkiye Tümleyen

- Negatif tam sayılar genellikle **ikiye tümleyen** (*two's complement*) ile temsil edilir.
- Kural: pozitif sayının **bitlerini ters çevir**, sonra **1 ekle**.
- En soldaki bit **işaret biti** gibi davranır: 0 → pozitif, 1 → negatif.
- 8 bitle aralık: **−128 … 127**. n bitle: -2^{n-1} … $2^{n-1} - 1$.

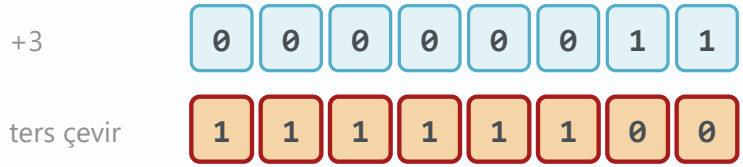
+3



8 bitlik bir bilgisayarda 3 sayısı: **00000011**.

Negatif Sayılar: İkiye Tümleyen

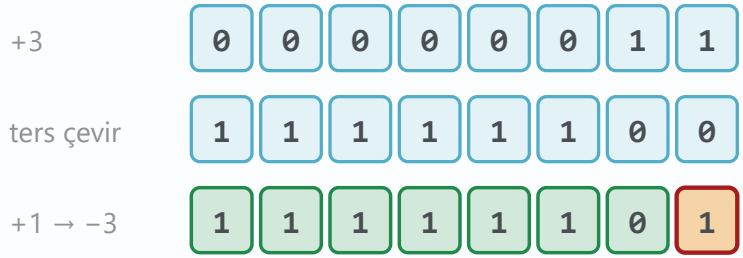
- Negatif tam sayılar genellikle **ikiye tümleyen** (*two's complement*) ile temsil edilir.
- Kural: pozitif sayının **bitlerini ters çevir**, sonra **1 ekle**.
- En soldaki bit **işaret biti** gibi davranır: 0 → pozitif, 1 → negatif.
- 8 bitle aralık: **−128 … 127**. n bitle: -2^{n-1} … $2^{n-1} - 1$.



1. adım: bütün bitler **ters çevrilir** (0 ↔ 1): 00000011 → 11111100 (birler tümleyeni).

Negatif Sayılar: İkiye Tümleyen

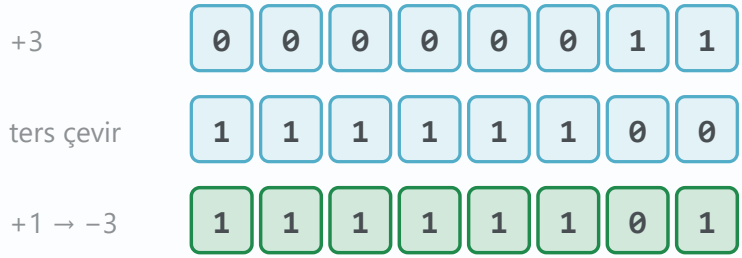
- Negatif tam sayılar genellikle **ikiye tümleyen** (*two's complement*) ile temsil edilir.
- Kural: pozitif sayının **bitlerini ters çevir**, sonra **1 ekle**.
- En soldaki bit **işaret biti** gibi davranır: 0 → pozitif, 1 → negatif.
- 8 bitle aralık: **−128 … 127**. n bitle: -2^{n-1} … $2^{n-1} - 1$.



2. adım: **1 eklenir**: 11111100 + 1 → **11111101**. Bu, −3'ün temsilidir.

Negatif Sayılar: İkiye Tümleyen

- Negatif tam sayılar genellikle **ikiye tümleyen** (*two's complement*) ile temsil edilir.
- Kural: pozitif sayının **bitlerini ters çevir**, sonra **1 ekle**.
- En soldaki bit **işaret biti** gibi davranır: 0 → pozitif, 1 → negatif.
- 8 bitle aralık: **−128 … 127**. n bitle: -2^{n-1} … $2^{n-1} - 1$.

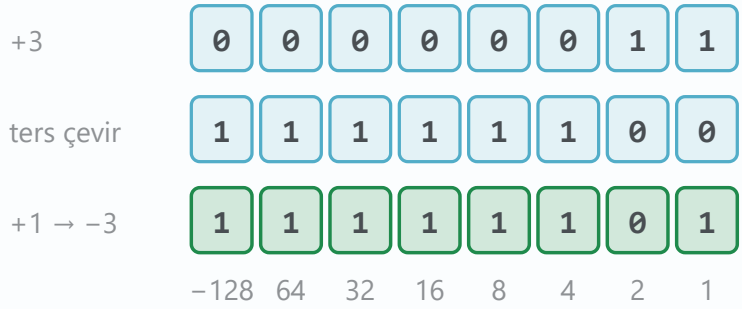


Sağlama: 00000011 + 11111101 = 1 00000000
9. bit sığmaz, atılır → 00000000 = 0 ✓

Neden işe yarar? 3 + (−3) toplandığında taşan 9. bit atılır ve sonuç tam olarak 0 olur: toplama devresi aynen kullanılır.

Negatif Sayılar: İkiye Tümleyen

- Negatif tam sayılar genellikle **ikiye tümleyen** (*two's complement*) ile temsil edilir.
- Kural: pozitif sayının **bitlerini ters çevir**, sonra **1 ekle**.
- En soldaki bit **işaret biti** gibi davranır: 0 → pozitif, 1 → negatif.
- 8 bitle aralık: **−128 … 127**. n bitle: -2^{n-1} … $2^{n-1} - 1$.



−128 + 64 + 32 + 16 + 8 + 4 + 1 = −3

Başka bir bakış: en soldaki bitin ağırlığı **−128**'dir. Bu yüzden en soldaki bit 1 ise sayı negatiftir.

SAYILAR

Java'da İkiye Tümleyen ve Taşma

```
int sayi = 3;
int eksi = ~sayi + 1; // ters çevir, 1 ekle
System.out.println(eksi);
byte b = (byte) 0b11111101;
System.out.println(b);
System.out.println(Integer.toBinaryString(-3));
System.out.println(Byte.MIN_VALUE + " .. "
    + Byte.MAX_VALUE);
```

```
-3
-3
11111111111111111111111111111101
-128 .. 127
```

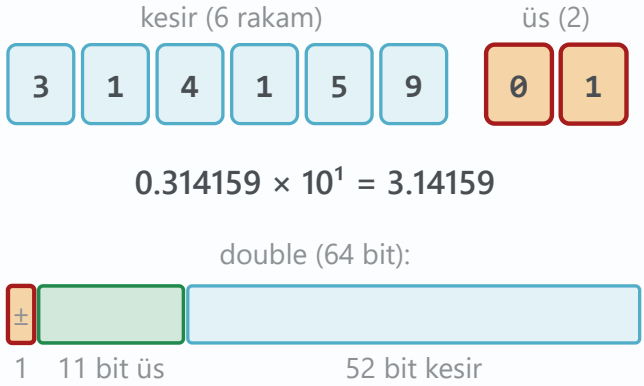
```
byte b = 127;
b++;
System.out.println(b);
int m = Integer.MAX_VALUE;
System.out.println(m + 1);
System.out.println(Math.addExact(m, 1));
```

```
-128
-2147483648
Exception in thread "main"
  java.lang.ArithmeticException:
  integer overflow
```

Taşma (overflow): sonuç tipin aralığına sığmazsa Java **hata vermez**, sayı "saat gibi" başa sarar: 127 + 1 → −128. Kontrollü toplama için `Math.addExact` kullanılır.

Çok Büyük ve Çok Küçük Sayılar: Kayan Nokta

- Sınırlı sayıda rakamla ancak sınırlı sayıda değer temsil edilebilir.
- Çok büyük / küçük sayılar için sayı **kesir** (mantis) ve **üs** olarak iki parçaya ayrılır; parçalar ayrı ayrı saklanır.
- Örnek: 8 rakamlık **31415901** gösteriminde ilk altı rakam (314159) kesri, son iki rakam (01) 10'un kuvvetini, yani 10^1'i belirtir.
- Bilgisayarlar aynı fikri **ikili** tabanda uygular: **IEEE 754** standardı. Java'da `float` 32 bit, `double` 64 bittir.



Adı buradan gelir: "kayan nokta" (*floating point*), üs değiştikçe ondalık noktanın yerinin kaymasıdır: $0.314159 \times 10^3 = 314.159$.

SAYILAR

Bitlerle Etkileşimli Deney

Sayı **13** | **göster** | **ters çevir** | **+1** | **× 2 (sola kaydır)** | **sıfırla**

±128	64	32	16	8	4	2	1
0	0	0	0	1	1	0	1

İkili **0000 1101**	İşaretsiz (0–255) **13**	İşaretli byte (−128–127) **13**
Onaltılı **0x0D**	Sekizli **015**	ASCII karakter **—**

Bir bite tıklanınca 0 ↔ 1 olur. −3 için: 3 → **ters çevir** → **+1**. İşaretli değer, en soldaki bitin ağırlığı −128 sayılarak bulunur.

Harfleri Sayılarla Temsil Etmek: ASCII

- Metin işlemek için her harfe bir **sayı** karşılık getirilmelidir: buna **karakter kodlaması** (character encoding) denir.
- **ASCII** (*American Standard Code for Information Interchange*), 1963'te ABD standartlar kurumu ASA (bugünkü **ANSI**) tarafından yayımlanmıştır.
- Büyük ve küçük harfleri, rakamları, noktalama ve özel karakterleri sayıyla temsil eder.
- **7 bitlik** bir koddur: **128** karakter (0–127). 0–31 ve 127 kontrol karakteri, 32–126 yazdırılabilir.
- **8 bitlik** genişletmeler (örneğin Türkçe için ISO 8859-9, Windows-1254) 128–255 arasına ek harfler koyarak **256** sembole çıkar.
- Tek bir 8 bitlik tablo, dünyanın tüm dillerine **yetersizdir**.

Karakter	Ondalık	İkili
`'A'`	65	`1000001`
`'a'`	97	`1100001`
`'0'`	48	`0110000`
boşluk	32	`0100000`
yeni satır `\n`	10	`0001010`

Küçük harf = büyük harf + **32**. İkilide fark yalnızca tek bir bittir (2^5).

ASCII Tablosu: Yazdırılabilir Karakterler

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32	SP 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
48	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
64	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
80	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
96	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
112	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127

Kod = satır + sütun. Örnek: `'J'` = 64 + 10 = **74**. Her hücrede üstte karakter, altta ondalık kod.

0–31: kontrol karakterleri; ör. 9 sekme (`\t`), 10 yeni satır (`\n`), 13 satır başı (`\r`). Tablonun tamamı: ASCII tablosu.

Bütün Diller İçin Tek Tablo: Unicode

- **Unicode**, dünyadaki tüm yazı sistemlerinin harf, sembol ve işaretlerine birer numara (**kod noktası**, *code point*) verir: `U+0041` = A, `U+00E7` = ç.
- **Unicode Konsorsiyumu** tarafından geliştirilir (1991); uluslararası karşılığı **ISO/IEC 10646** ile eş zamanlı tutulur.
- İlk sürüm 16 bitlikti (65.536 karakter). Bu yetmeyince alan **U+10FFFF**'e genişletildi: 1.114.112 kod noktası, bugün 150.000'den fazla karakter.
- Her karakteri sabit 32 bitle (UTF-32) saklamak, İngilizce gibi metinlerde gereğinden fazla yer harcar. Bu yüzden **UTF** (*Unicode Transformation Format*) kodlamaları tanımlanmıştır.

Kar.	Kod noktası	Ondalık	UTF-8 baytları
A	`U+0041`	65	`41`
ç	`U+00E7`	231	`C3 A7`
ğ	`U+011F`	287	`C4 9F`
€	`U+20AC`	8364	`E2 82 AC`
😀	`U+1F600`	128512	`F0 9F 98 80`

İlk 128 kod noktası ASCII ile birebir aynıdır.

UTF-8: Değişken Uzunluklu Kodlama

- Sık kullanılan karakterler **az**, nadir olanlar **çok** bayt kullanır: 1–4 bayt.
- ASCII metin UTF-8'de de aynen geçerlidir (geriye uyumlu). Bugün web sayfalarının çok büyük bölümü UTF-8'dir.
- İlk baytın başındaki bitler, karakterin kaç bayt süreceğini söyler.
- **UTF-16** çoğu karakteri 2 baytla yazar (`ş` = `01 5F` , `€` = `20 AC`); Java'nın `char` tipi bir UTF-16 birimidir.

Kod noktası	Bayt kalıbı
U+0000–U+007F	`0xxxxxxx`
U+0080–U+07FF	`110xxxxx 10xxxxxx`
U+0800–U+FFFF	`1110xxxx 10xxxxxx ×2`
U+10000–U+10FFFF	`11110xxx 10xxxxxx ×3`

U+00E7 = 231 = 00011 100111 (11 bit)



renkli: kalıp bitleri · diğerleri: kod noktasının bitleri

```
String kelime = "çikolata";
System.out.println(kelime.length());
System.out.println(kelime.getBytes(
    StandardCharsets.UTF_8).length);
```

Çıktı:

```
8
9
```

Emoji de Bir Karakterdir

- Duygu, ifade ve nesneleri hızlıca aktaran simgelerdir; bilgisayar onları **metin** olarak işler.
- Standartları **Unicode Konsorsiyumu** belirler: her emoji benzersiz bir kod noktasıdır, ör. 😀 = `U+1F600`.
- Bazı emojiler **birden çok kod noktasının** birleşimidir: 👨‍👧 = adam + ZWJ + kadın + ZWJ + kız (5 kod noktası; ZWJ: sıfır genişlikli birleştirici).
- Kod her cihazda **aynıdır**, böylece emoji doğru anlaşılır; ama **çizimi** (glif) üreticiye göre farklı görünebilir.

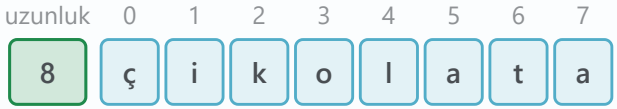
```
String yuz = "😀";
System.out.println(yuz.length());
System.out.println(
    yuz.codePointCount(0, yuz.length()));
String aile = "👨‍👧";
System.out.println(aile.length());
```

Çıktı:

```
2
1
8
```

Java `String`'i karakterleri **UTF-16** ile saklar: `char` 16 bittir, U+FFFF'in üstündeki karakterler **iki** `char` (vekil çift, *surrogate pair*) tutar. Bu yüzden `length()` "karakter sayısı" değil, `char` sayısıdır.

Dizginin Karakter Kodlarını Yazdırmak



Dizgi bellekte karakter dizisi ve uzunluk olarak durur.

Adım 1 / 11

```
▶ String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
3   char ch = kelime.charAt(i);
4   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

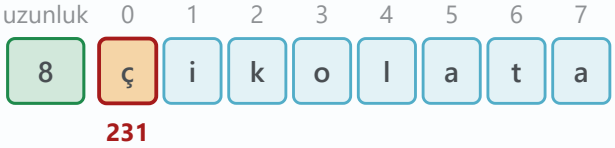
DEĞİŞKENLER

`kelime = "çikolata"` `i = –` `ch = –`

ÇIKTI

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 0: `charAt(0)` = 'ç'; `(int)` dönüşümü kodunu verir: **231**. (Unicode U+00E7)

Adım 2 / 11

```
1 String kelime = "çikolata";
▶ for (int i = 0; i < kelime.length(); i++) {
▶   char ch = kelime.charAt(i);
▶   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

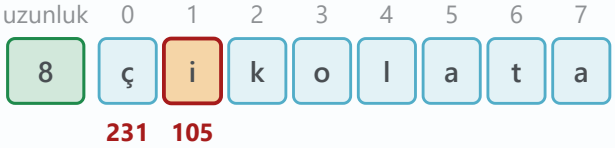
`kelime = "çikolata"` `i = 0` `ch = 'ç'`

ÇIKTI

```
231
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 1: `charAt(1)` = 'i'; `(int)` dönüşümü kodunu verir: **105**.

Adım 3 / 11

```
1 String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
▶   char ch = kelime.charAt(i);
▶   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

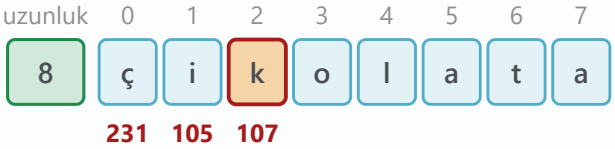
`kelime = "çikolata"` `i = 1` `ch = 'i'`

ÇIKTI

```
231 105
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 2: `charAt(2)` = 'k'; `(int)` dönüşümü kodunu verir: **107**.


```
1 String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
▶   char ch = kelime.charAt(i);
▶   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

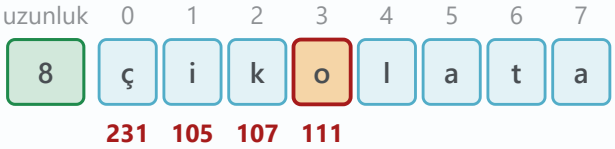
`kelime = "çikolata"` `i = 2` `ch = 'k'`

ÇIKTI

```
231 105 107
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 3: `charAt(3)` = 'o'; `(int)` dönüşümü kodunu verir: **111**.

Adım 5 / 11

```
1 String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
▶   char ch = kelime.charAt(i);
▶   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

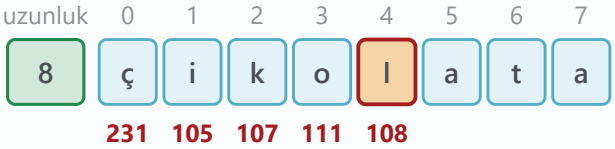
`kelime = "çikolata"` `i = 3` `ch = 'o'`

ÇIKTI

```
231 105 107 111
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 4: `charAt(4)` = 'l'; `(int)` dönüşümü kodunu verir: **108**.


```
1 String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
▶   char ch = kelime.charAt(i);
▶   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

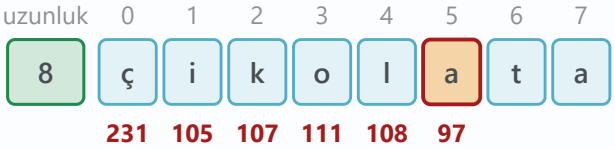
`kelime = "çikolata"` `i = 4` `ch = 'l'`

ÇIKTI

```
231 105 107 111 108
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 5: `charAt(5)` = 'a'; `(int)` dönüşümü kodunu verir: **97**.

Adım 7 / 11

```
1 String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
▶   char ch = kelime.charAt(i);
▶   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

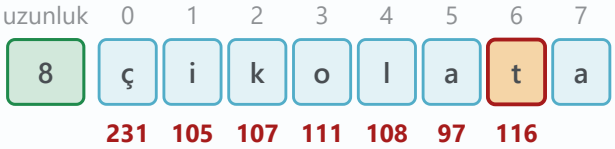
`kelime = "çikolata"` `i = 5` `ch = 'a'`

ÇIKTI

```
231 105 107 111 108 97
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 6: `charAt(6)` = 't'; `(int)` dönüşümü kodunu verir: **116**.

Adım 8 / 11

```
1 String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
▶   char ch = kelime.charAt(i);
▶   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

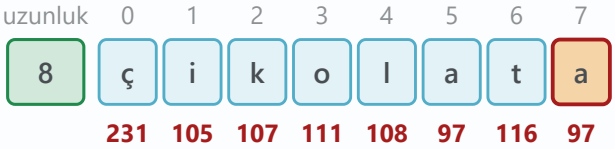
`kelime = "çikolata"` `i = 6` `ch = 't'`

ÇIKTI

```
231 105 107 111 108 97 116
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 7: `charAt(7)` = 'a'; `(int)` dönüşümü kodunu verir: **97**.

Adım 9 / 11

```
1 String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
▶   char ch = kelime.charAt(i);
▶   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

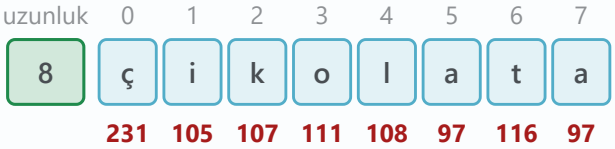
`kelime = "çikolata"` `i = 7` `ch = 'a'`

ÇIKTI

```
231 105 107 111 108 97 116 97
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



i = 8: koşul `8 < 8` yanlış → döngü biter.

Adım 10 / 11

```
1 String kelime = "çikolata";
▶ for (int i = 0; i < kelime.length(); i++) {
3   char ch = kelime.charAt(i);
4   System.out.print((int) ch + " ");
5 }
6 System.out.println();
7 System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

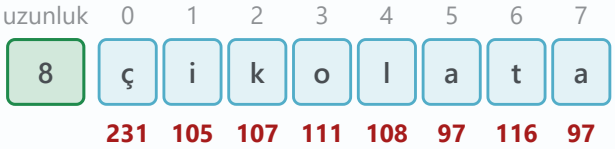
`kelime = "çikolata"` `i = 8` `ch = –`

ÇIKTI

```
231 105 107 111 108 97 116 97
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Dizginin Karakter Kodlarını Yazdırmak



Satır sonu ve uzunluk yazdırılır: `length()` = 8.

Adım 11 / 11

```
1 String kelime = "çikolata";
2 for (int i = 0; i < kelime.length(); i++) {
3   char ch = kelime.charAt(i);
4   System.out.print((int) ch + " ");
5 }
▶ System.out.println();
▶ System.out.println("Uzunluk: " + kelime.length());
```

DEĞİŞKENLER

`kelime = "çikolata"` `i = –` `ch = –`

ÇIKTI

```
231 105 107 111 108 97 116 97
Uzunluk: 8
```

Dizginin uzunluğu ayrıca tutulur, metnin nerede bittiği böyle bilinir. “ç” ASCII'de yoktur: 231 onun Unicode kod noktasıdır (U+00E7); aynı değer Latin-1 ve Türkçe ISO 8859-9 tablolarında da 231'dir.

Karakter Kodları: Etkileşimli Deney

Metin `çikolata 😀` | Java | ağaç | € 🫂

ç	i	k	o	l	a
231	105	107	111	108	97
U+00E7	U+0069	U+006B	U+006F	U+006C	U+0061
C3 A7	69	6B	6F	6C	61

t	a	SP	😀
116	97	32	128512
U+0074	U+0061	U+0020	U+1F600
74	61	20	F0 9F 98 80

Kod noktası: **10** · Java `length()` (UTF-16 char): **11** · UTF-8 bayt: **14**

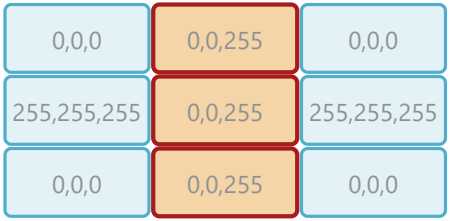
Her kutu: karakter · kod noktası (ondalık ve U+onaltılı) · UTF-8 baytları. Renkli kutular ASCII dışıdır. İlk 12 kod noktası gösterilir.

YAPILANDIRILMIŞ BİLGİ

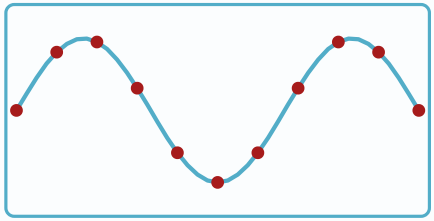
Her Türlü Bilgi Sayılarla Temsil Edilir

Sayı **dizileri**, bilgiyi saklamamızı ve bilgisayarla etkileşim kurmamızı sağlar.

Resim: piksel = (R, G, B)



Ses: basınç örnekleri



Film: saniyede 24–30 kare



Resimler

Her piksel kırmızı, yeşil, mavi miktarını veren **üç sayıdır** (0–255). Mavi piksel: (0, 0, 255).

Sesler

Havadaki ses basıncı zaman içinde düzenli aralıklarla **ölçülür** (örnekleme); CD kalitesi saniyede 44.100 örnek.

Filmler

Saniyede 24 veya 30 kare hızında ardışık **resimler**; her kare, resim gibi sayı dizisidir.

YAPILANDIRILMIŞ BİLGİ

Örnek: Bir Video Ne Kadar Yer Tutar?

```
long piksel = 1920L * 1080;  // Full HD bir kare
long kare = piksel * 3;      // R, G, B: 3 bayt
long saniye = kare * 30;     // saniyede 30 kare
System.out.println("Piksel: " + piksel);
System.out.println("Kare (bayt): " + kare);
System.out.println("1 saniye (bayt): " + saniye);
System.out.println("1 dakika (MB): "
    + saniye * 60 / 1_000_000);
// 44,1 kHz, 16 bit (2 bayt), stereo, 1 dakika
long ses = 44_100L * 2 * 2 * 60;
System.out.println("1 dk CD sesi (bayt): " + ses);
```

Çıktı:

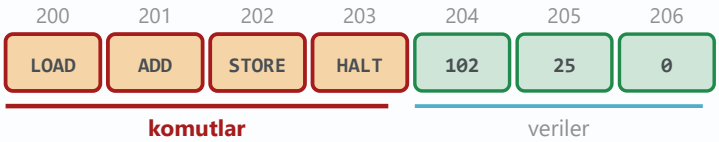
```
Piksel: 2073600
Kare (bayt): 6220800
1 saniye (bayt): 186624000
1 dakika (MB): 11197
1 dk CD sesi (bayt): 10584000
```

Sıkıştırmasız bir dakikalık Full HD video ≈ **11 GB**! Gerçekte video ve ses **sıkıştırılarak** (H.264, MP3…) saklanır; ama sıkıştırılmış dosya da yine bir sayı dizisidir. Not: `1920L` yazımı çarpımı `long` tipinde yapar; büyük sonuçlar `int`'e sığmaz.

SAKLANAN PROGRAM

Saklanan Program Kavramı

- Modern bilgisayarların temel ilkesidir: **komutlar da veriler gibi sayıdır** ve aynı bellekte saklanır.
- Bellekte saklanan komutlar ve veriler birlikte **programı** oluşturur. Her komut bir görevi veya işlemi temsil eder.
- Komutlar **ardışık** olarak işlenir.
- Bir koşul sağlandığında program farklı bir komuta **atlayabilir**; böylece programlar duruma göre farklı yollar izler (ileride `if` ve döngüler bunu kullanır).



ENIAC (1945)

- Program, kablolar ve anahtarlarla **elle** kurulur.
- Yeni bir program için günlerce yeniden kablolama.

EDVAC tasarımı (1945)

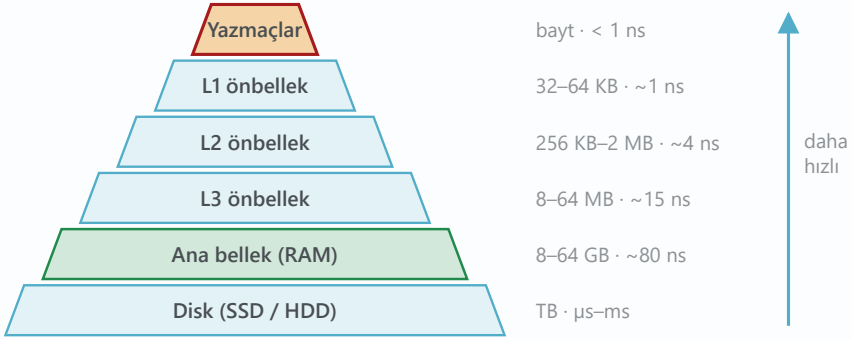
+ Komutlar ve veriler bellekte **sıralı** saklanır.
+ Program değiştirmek = belleğe yeni sayılar yazmak: esnek ve genişletilebilir.

Fikri J. von Neumann'ın 1945 tarihli EDVAC raporu yaydı; saklanan programı ilk çalıştıran makine Manchester “Baby” oldu (1948).

SAKLANAN PROGRAM

Bellek Hiyerarşisi

- **Bellek**, çalışan programların ve verilerin geçici olarak saklandığı yerdir.
- **L1 önbellek** (cache): en hızlısı, işlemcinin içinde.
- **L2 önbellek**: L1'i destekler, daha büyüktür. Çoğu işlemcide bir de **L3** vardır.
- **Ana bellek** (RAM): çok daha büyük ama daha yavaş.
- İşlemciye yakın bellek daha hızlı ama daha küçük ve pahalıdır; hiyerarşi performansı doğrudan etkiler.



Boyut ve süreler yaklaşık değerlerdir, işlemciye göre değişir. 1 ns = saniyenin milyarda biri.

SAKLANAN PROGRAM

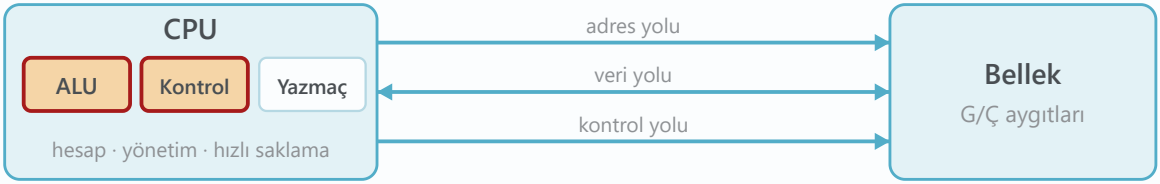
İşlemci (CPU) ve Veri Yolu (Bus)

Merkezî İşlem Birimi (CPU)

- Bilgisayarın "beyni"; tüm hesaplamalar burada yapılır.
- **Aritmetik Mantık Birimi** (ALU, *Arithmetic Logic Unit*): toplama, çıkarma, karşılaştırma gibi hesapları yapar.
- **Kontrol Birimi** (Control Unit): komutları yorumlar, işlemleri sıraya koyar ve yönlendirir.
- **Yazmaçlar** (register): geçici veriler için çok küçük ama çok hızlı saklama alanları.

Veri Yolu (Bus)

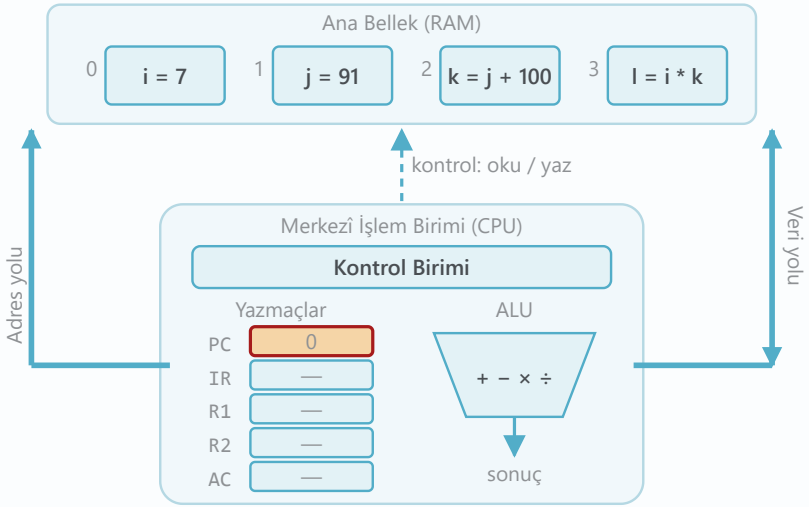
- Birçok paralel telden oluşan bir **iletişim yolu** sistemidir.
- **Adres sinyalleri**: hangi bellek hücresine veya aygıta erişileceğini seçer.
- **Veri sinyalleri**: işlenecek ya da saklanacak asıl veriyi taşır.
- **Kontrol sinyalleri**: işlemi yönlendirir; ör. "oku" mu "yaz" mı.



Elektronik bağlantısı: 32 hatlı bir adres yolu 2^{32} farklı adres seçebilir → 4 GiB bellek. Bu yüzden 32 bitlik sistemler en fazla yaklaşık 4 GB RAM kullanabiliyordu.

SAKLANAN PROGRAM

CPU – Bellek – Veri Yolu – Yazmaçlar



- CPU, **adres yolundan** hangi hücreyi istediğini söyler.
- Hücrenin içeriği **veri yolundan** gelir (okuma) ya da gider (yazma).
- **PC** (program sayacı): sıradaki komutun adresi.
- **IR** (komut yazmacı): o an çözülen komut.
- **R1, R2, AC** (akümülatör): ara sonuçlar.
- ALU yazmaçlardaki sayılarla hesap yapar.

SAKLANAN PROGRAM

Bellek, Adresler ve Program Sayacı

- Bellek, veri ve talimatların saklandığı alandır; bir program **uzun bir talimat listesidir**.
- Her bellek hücresinin bir **adresi** vardır; hücrede bir talimatın ya da verinin **sayıyla kodlanmış** hali bulunur.
- Talimatlar bellekten tek tek alınıp işlemcideki **yazmaçlara** yüklenir ve sırayla çalıştırılır.
- En önemli yazmaçlardan biri **program sayacıdır** (PC, *program counter*): sıradaki talimatın adresini tutar.
- İşlemci normalde adres sırasıyla ilerler; ama bazı talimatlar bu **sırayı değiştirir**: 204'teki "JUMP 7" PC'yi 211 yapar.

Adres	İçerik	Tür
200	`ADD to R1`	talimat
201	`102`	veri
202	…	
203	…	
204	`JUMP 7`	talimat (atla)
…	…	
211	sonraki talimat	talimat

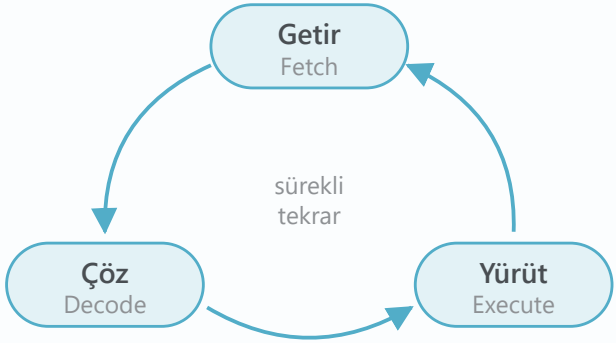
Bellek hücresi talimat mı veri mi, ona bakarak anlaşılmaz: ikisi de sayıdır. Fark, işlemcinin o hücreyi **nasıl kullandığıdır**.

SAKLANAN PROGRAM

Getir–Çöz–Yürüt Döngüsü

Getir–Çöz–Yürüt (*Fetch–Decode–Execute*) döngüsü, programların işlemcide nasıl yürütüldüğünü tanımlar.

1. **Getir** (Fetch): işlemci, PC'nin gösterdiği adresteki sıradaki komutu bellekten alır; PC bir sonraki adrese ilerler.
2. **Çöz** (Decode): komut çözülür: hangi işlem, hangi yazmaçlar, hangi adres? Her komutun belirli bir işlevi vardır.
3. **Yürüt** (Execute): komutun gerektirdiği işlem yapılır: veri işleme, sonuç hesaplama ya da bir işlemi başlatma.



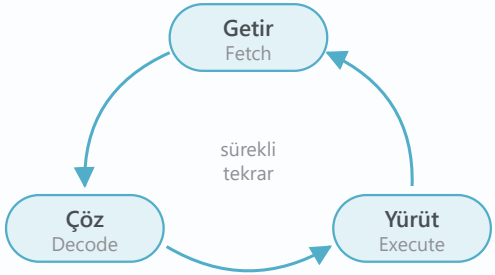
Ardından bir sonraki komut alınır ve döngü, bilgisayar açık olduğu sürece **sürekli** tekrarlanır.

3 GHz'lik bir işlemcinin saati saniyede 3 milyar kez “tık” eder; modern işlemciler birden çok komutu üst üste bindirerek (boru hattı, *pipeline*) işler.

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
▶ 200: LOAD  R1, [205] ; R1 ← bellek[205]
  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
  202: STORE R1, [207] ; bellek[207] ← R1
  203: HALT            ; dur
  204: ...
  205: 102             ; veri
  206: 25              ; veri
  207: 0               ; sonuç buraya
```



Program ve veriler bellekte. Başlangıçta **PC = 200**.

Adım 1 / 12

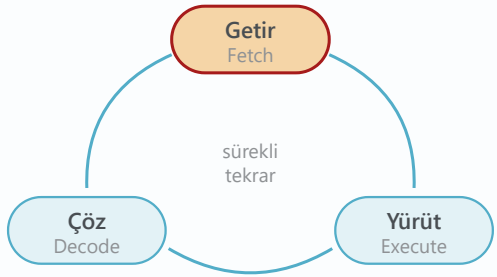
DEĞİŞKENLER

`PC = 200` `IR = –` `R1 = 0` `bellek[207] = 0`

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
▶ 200: LOAD  R1, [205] ; R1 ← bellek[205]
  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
  202: STORE R1, [207] ; bellek[207] ← R1
  203: HALT            ; dur
  204: ...
  205: 102             ; veri
  206: 25              ; veri
  207: 0               ; sonuç buraya
```



Getir: 200'deki komut IR'ye alınır, PC 201'e ilerler.

Adım 2 / 12

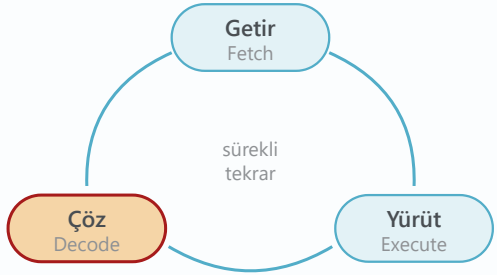
DEĞİŞKENLER

`PC = 201` `IR = LOAD R1,[205]` `R1 = 0` `bellek[207] = 0`

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
200: LOAD  R1, [205] ; R1 ← bellek[205]
201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
202: STORE R1, [207] ; bellek[207] ← R1
203: HALT            ; dur
204: ...
205: 102             ; veri
206: 25              ; veri
207: 0               ; sonuç buraya
```



Çöz: işlem = LOAD (yükle), hedef = R1, adres = 205.

Adım 3 / 12

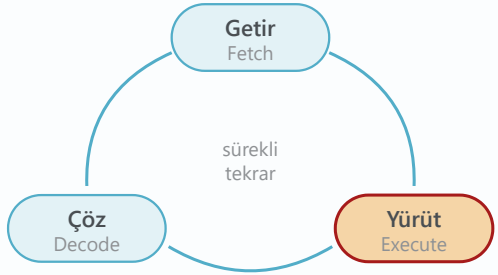
DEĞİŞKENLER

PC = 201 | IR = LOAD R1,[205] | R1 = 0 | bellek[207] = 0

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
▶ 200: LOAD  R1, [205] ; R1 ← bellek[205]
  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
  202: STORE R1, [207] ; bellek[207] ← R1
  203: HALT            ; dur
  204: ...
▶ 205: 102             ; veri
  206: 25              ; veri
  207: 0               ; sonuç buraya
```



Yürüt: bellek[205] = 102 okunur, R1'e yazılır.

Adım 4 / 12

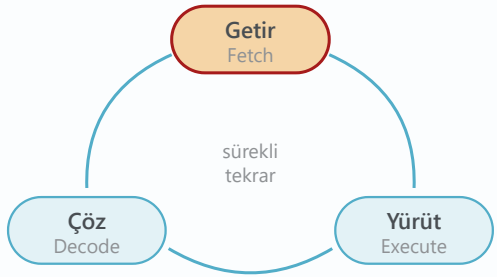
DEĞİŞKENLER

PC = 201 | IR = LOAD R1,[205] | R1 = 102 | bellek[207] = 0

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
1  200: LOAD  R1, [205] ; R1 ← bellek[205]
▶  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
3  202: STORE R1, [207] ; bellek[207] ← R1
4  203: HALT            ; dur
5  204: ...
6  205: 102             ; veri
7  206: 25              ; veri
8  207: 0               ; sonuç buraya
```



Getir: PC = 201'deki komut alınır, PC 202 olur.

Adım 5 / 12

DEĞİŞKENLER

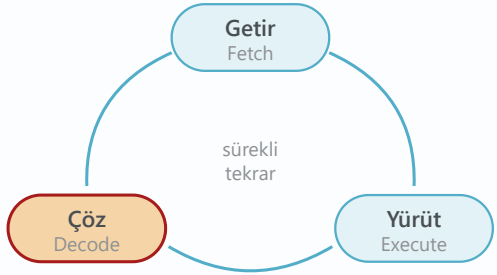
PC = 202 | IR = ADD R1,[206]

R1 = 102 | bellek[207] = 0

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
1  200: LOAD  R1, [205] ; R1 ← bellek[205]
▶  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
3  202: STORE R1, [207] ; bellek[207] ← R1
4  203: HALT            ; dur
5  204: ...
6  205: 102             ; veri
7  206: 25              ; veri
8  207: 0               ; sonuç buraya
```



Çöz: işlem = ADD (topla), R1 ile bellek[206] toplanacak.

Adım 6 / 12

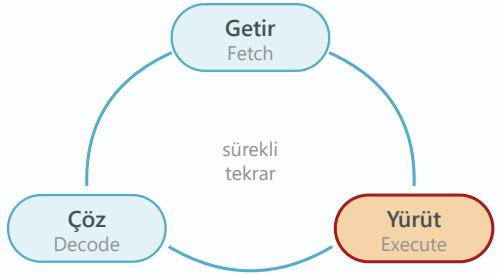
DEĞİŞKENLER

`PC = 202` `IR = ADD R1,[206]` `R1 = 102` `bellek[207] = 0`

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
1  200: LOAD  R1, [205] ; R1 ← bellek[205]
▶  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
3  202: STORE R1, [207] ; bellek[207] ← R1
4  203: HALT            ; dur
5  204: ...
6  205: 102             ; veri
▶  206: 25              ; veri
8  207: 0               ; sonuç buraya
```



Yürüt: ALU 102 + 25 = **127** hesaplar, sonuç R1'e.

Adım 7 / 12

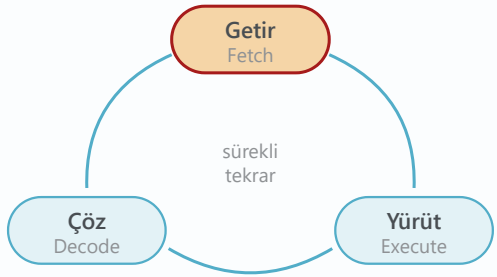
DEĞİŞKENLER

`PC = 202` `IR = ADD R1,[206]` `R1 = 127` `bellek[207] = 0`

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
1  200: LOAD  R1, [205] ; R1 ← bellek[205]
2  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
▶  202: STORE R1, [207] ; bellek[207] ← R1
4  203: HALT            ; dur
5  204: ...
6  205: 102             ; veri
7  206: 25              ; veri
8  207: 0               ; sonuç buraya
```



Getir: 202'deki STORE alınır, PC 203 olur.

Adım 8 / 12

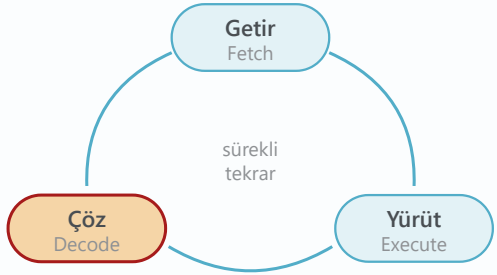
DEĞİŞKENLER

PC = 203 | IR = STORE R1,[207]

R1 = 127 | bellek[207] = 0

Oyuncak Bir Bilgisayarda 102 + 25

```
1  200: LOAD  R1, [205] ; R1 ← bellek[205]
2  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
▶  202: STORE R1, [207] ; bellek[207] ← R1
4  203: HALT            ; dur
5  204: ...
6  205: 102             ; veri
7  206: 25              ; veri
8  207: 0               ; sonuç buraya
```



Çöz: R1'in değeri 207 adresine yazılacak (kontrol yolu: “yaz”).

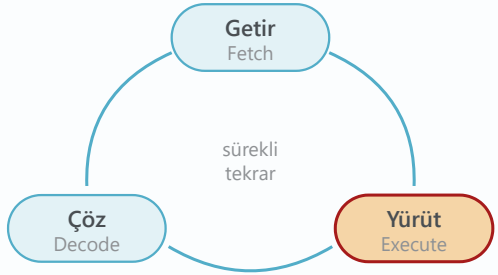
Adım 9 / 12

DEĞİŞKENLER

PC = **203** | IR = **STORE R1,[207]** | R1 = **127** | bellek[207] = **0**

Oyuncak Bir Bilgisayarda 102 + 25

```
200: LOAD  R1, [205] ; R1 ← bellek[205]
201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
▶ 202: STORE R1, [207] ; bellek[207] ← R1
203: HALT            ; dur
204: ...
205: 102             ; veri
206: 25              ; veri
▶ 207: 0             ; sonuç buraya
```



Yürüt: bellek[207] ← 127. Sonuç artık bellekte.

Adım 10 / 12

DEĞİŞKENLER

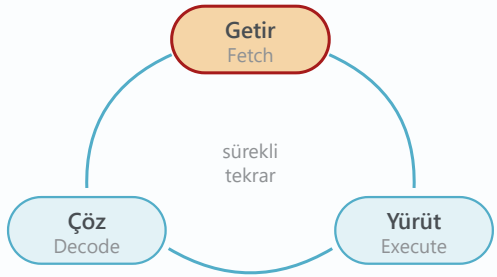
PC = 203 | IR = STORE R1,[207]

R1 = 127 | bellek[207] = 127

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
1  200: LOAD  R1, [205] ; R1 ← bellek[205]
2  201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
3  202: STORE R1, [207] ; bellek[207] ← R1
▶  203: HALT            ; dur
5  204: ...
6  205: 102             ; veri
7  206: 25              ; veri
8  207: 0               ; sonuç buraya
```



Getir: 203'teki HALT alınır, PC 204 olur.

Adım 11 / 12

DEĞİŞKENLER

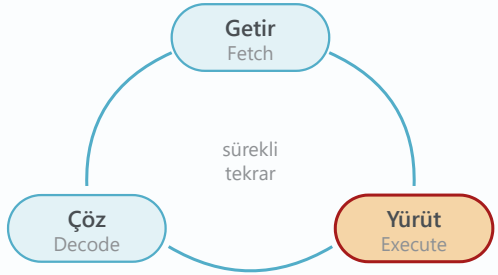
PC = **204** IR = **HALT** R1 = **127**

bellek[207] = **127**

SAKLANAN PROGRAM · ADIM ADIM

Oyuncak Bir Bilgisayarda 102 + 25

```
1 200: LOAD  R1, [205] ; R1 ← bellek[205]
2 201: ADD   R1, [206] ; R1 ← R1 + bellek[206]
3 202: STORE R1, [207] ; bellek[207] ← R1
▶ 203: HALT            ; dur
5 204: ...
6 205: 102             ; veri
7 206: 25              ; veri
8 207: 0               ; sonuç buraya
```



Çöz + Yürüt: HALT → işlemci durur. Java'daki `toplam = a + b;` satırı da sonunda bu tür komutlara dönüşür.

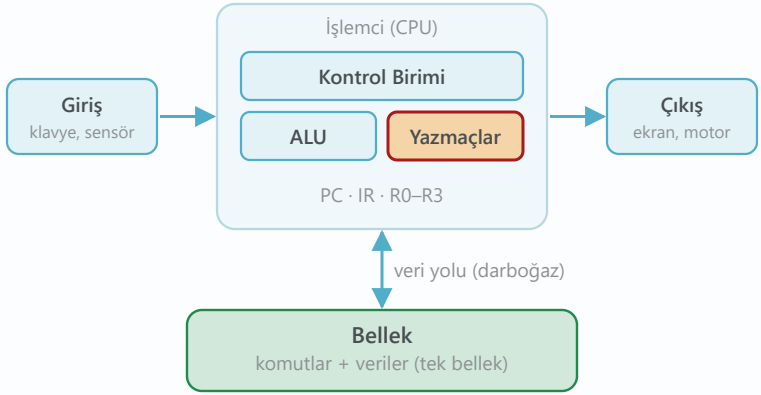
Adım 12 / 12

DEĞİŞKENLER

PC = **204** | IR = **HALT** | R1 = **127** | bellek[207] = **127**

von Neumann Mimarisi

- Saklanan program fikrinin donanım planıdır: **işlemci**, **bellek**, **giriş/çıkış** ve bunları bağlayan **veri yolu**.
- Komutlar ve veriler **aynı bellekte** durur ve aynı yoldan taşınır.
- Her komut bir **işlem kodu** (*opcode*) ile temsil edilir; işlem kodu hangi işlemin yapılacağını belirtir.
- Tek yol, işlemciyi bekletebilir: **von Neumann darboğazı**. Önbellekler bu yüzden vardır.



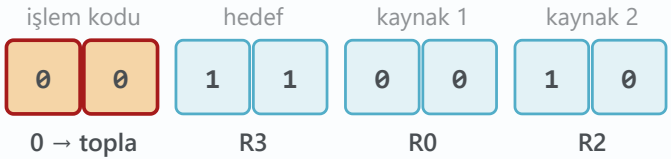
Karşılaştırma: **Harvard mimarisinde** komut ve veri belleği ayrıdır; Arduino'daki AVR gibi pek çok mikrodenetleyici bu yapıyı kullanır.

Komutlar da Sayıdır: Kodlama

Bir oyuncak bilgisayar düşünelim: **8 bit**, **4 yazmaç** (R0–R3), **256** adet 8 bitlik bellek hücresi ve **4 komut**: toplama (00), çıkarma (01), çarpma (10), bölme (11).

- İlk 2 bit **işlem kodu** (4 komut → 2 bit).
- Sonraki 2 bit **hedef yazmaç** (destination register).
- Son 4 bit, iki **işlenen yazmaç** (2'şer bit).

"R0 ile R2'nin içeriğini topla, sonucu R3'e yaz" = **0 3 0 2** → `00110010`



İşlemcinin "çöz" adımı Java'da: basamak değeriyle bölme!

```
int komut = 0b00110010;
int islemKodu = komut / 64;      // ilk 2 bit
int hedef = (komut / 16) % 4;    // sonraki 2 bit
int kaynak1 = (komut / 4) % 4;   // sonraki 2 bit
int kaynak2 = komut % 4;         // son 2 bit
System.out.println(komut);
System.out.println(islemKodu + " " + hedef
    + " " + kaynak1 + " " + kaynak2);
```

Çıktı:

```
50
0 3 0 2
```

Komut Çözme: Kod İzleme

```
1 int komut = 0b00110010;
2 int islemKodu = komut / 64;      // ilk 2 bit
3 int hedef = (komut / 16) % 4;    // sonraki 2 bit
4 int kaynak1 = (komut / 4) % 4;   // sonraki 2 bit
5 int kaynak2 = komut % 4;         // son 2 bit
6 System.out.println(komut);
7 System.out.println(islemKodu + " " + hedef
8     + " " + kaynak1 + " " + kaynak2);
```



komut = 50

32 + 16 + 2 = 50

Komut 8 bitlik bir sayıdır: `00 11 00 10` = 50. Bitler dört ikili gruba ayrılır.

Adım 1 / 7

DEĞİŞKENLER

`komut = 50`

ÇIKTI

Komut Çözme: Kod İzleme

```
1 int komut = 0b00110010;
▶ int islemKodu = komut / 64;      // ilk 2 bit
3 int hedef = (komut / 16) % 4;    // sonraki 2 bit
4 int kaynak1 = (komut / 4) % 4;   // sonraki 2 bit
5 int kaynak2 = komut % 4;         // son 2 bit
6 System.out.println(komut);
7 System.out.println(islemKodu + " " + hedef
8     + " " + kaynak1 + " " + kaynak2);
```



islemKodu = 0 → toplama

`50 / 64 = 0`

64 = 2^6 ile bölmek sağdaki 6 biti atar; geriye ilk 2 bit kalır: `00` = 0, yani **toplama**.

Adım 2 / 7

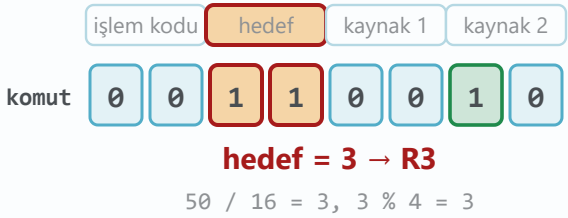
DEĞİŞKENLER

`komut = 50` `islemKodu = 0`

ÇIKTI

Komut Çözme: Kod İzleme

```
1 int komut = 0b00110010;
2 int islemKodu = komut / 64;      // ilk 2 bit
▶ int hedef = (komut / 16) % 4;    // sonraki 2 bit
4 int kaynak1 = (komut / 4) % 4;   // sonraki 2 bit
5 int kaynak2 = komut % 4;         // son 2 bit
6 System.out.println(komut);
7 System.out.println(islemKodu + " " + hedef
8     + " " + kaynak1 + " " + kaynak2);
```



`50 / 16 = 3, 3 % 4 = 3`

16 ile bölmek sağdaki 4 biti atar (`0011` = 3); `% 4` son 2 biti alır: 3 → **R3**.

Adım 3 / 7

DEĞİŞKENLER

`komut = 50` `islemKodu = 0` `hedef = 3`

ÇIKTI

Komut Çözme: Kod İzleme

```
1 int komut = 0b00110010;
2 int islemKodu = komut / 64;      // ilk 2 bit
3 int hedef = (komut / 16) % 4;    // sonraki 2 bit
▶ int kaynak1 = (komut / 4) % 4;   // sonraki 2 bit
5 int kaynak2 = komut % 4;         // son 2 bit
6 System.out.println(komut);
7 System.out.println(islemKodu + " " + hedef
8     + " " + kaynak1 + " " + kaynak2);
```



kaynak1 = 0 → R0

`50 / 4 = 12, 12 % 4 = 0`

4 ile bölmek sağdaki 2 biti atar (`001100` = 12); `12 % 4` = 0 → **R0**.

Adım 4 / 7

DEĞİŞKENLER

`komut = 50` `islemKodu = 0` `hedef = 3` `kaynak1 = 0`

ÇIKTI

Komut Çözme: Kod İzleme

```
1 int komut = 0b00110010;
2 int islemKodu = komut / 64;      // ilk 2 bit
3 int hedef = (komut / 16) % 4;    // sonraki 2 bit
4 int kaynak1 = (komut / 4) % 4;   // sonraki 2 bit
▶ int kaynak2 = komut % 4;         // son 2 bit
6 System.out.println(komut);
7 System.out.println(islemKodu + " " + hedef
8     + " " + kaynak1 + " " + kaynak2);
```



kaynak2 = 2 → R2

`50 % 4 = 2`

`% 4` doğrudan son 2 biti verir: `10` = 2 → **R2**.

Adım 5 / 7

DEĞİŞKENLER

`komut = 50` `islemKodu = 0`
`hedef = 3` `kaynak1 = 0`
`kaynak2 = 2`

ÇIKTI

Komut Çözme: Kod İzleme

```
1 int komut = 0b00110010;
2 int islemKodu = komut / 64;      // ilk 2 bit
3 int hedef = (komut / 16) % 4;    // sonraki 2 bit
4 int kaynak1 = (komut / 4) % 4;   // sonraki 2 bit
5 int kaynak2 = komut % 4;         // son 2 bit
▶ System.out.println(komut);
7 System.out.println(islemKodu + " " + hedef
8     + " " + kaynak1 + " " + kaynak2);
```



komut = 50

Komut sayı olarak yazılır: 50.

Adım 6 / 7

DEĞİŞKENLER

`komut = 50` `islemKodu = 0` `hedef = 3` `kaynak1 = 0` `kaynak2 = 2`

ÇIKTI

```
50
```

Komut Çözme: Kod İzleme

```
1 int komut = 0b00110010;
2 int islemKodu = komut / 64;      // ilk 2 bit
3 int hedef = (komut / 16) % 4;    // sonraki 2 bit
4 int kaynak1 = (komut / 4) % 4;   // sonraki 2 bit
5 int kaynak2 = komut % 4;         // son 2 bit
6 System.out.println(komut);
▶ System.out.println(islemKodu + " " + hedef
▶     + " " + kaynak1 + " " + kaynak2);
```



R3 ← R0 + R2

Alanlar: `0 3 0 2`. Yani 50 sayısı, işlemci için "R3 ← R0 + R2" komutudur.

Adım 7 / 7

DEĞİŞKENLER

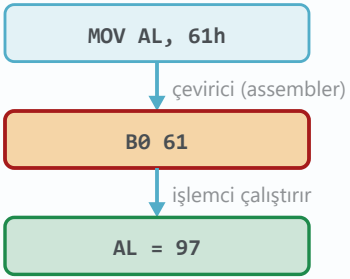
`komut = 50` `islemKodu = 0` `hedef = 3` `kaynak1 = 0` `kaynak2 = 2`

ÇIKTI

```
50
0 3 0 2
```

Assembly Dili

- Assembly dili belirli bir **işlemci mimarisine** özeldir: x86 için yazılan kod ARM'da çalışmaz.
- Her satır genellikle **tek bir** makine komutuna karşılık gelir.
- Bilgisayarın alt düzeyde nasıl çalıştığını anlamak için önemlidir.
- **Performans optimizasyonu** ve donanımla doğrudan etkileşim (sürücüler, gömülü sistemler) için kullanılır.



x86 komutu	Anlamı	Makine kodu
`MOV AL, 61h`	AL yazmacına 61h (= 97) değerini taşı	`B0 61`
`MOV AX, BX`	BX'teki değeri AX'e kopyala	`66 89 D8`
`ADD EAX, 10`	EAX'e 10 ekle	`83 C0 0A`
`XOR EAX, EAX`	EAX'i sıfırla (x XOR x = 0)	`31 C0`

61h = 0x61 = 97 = ASCII `'a'` . Bellekteki bir bayt sayı mı, harf mi, komut mu? Bunu **programın onu nasıl kullandığı** belirler.

ÖZET

Özet

Konu	Hatırlanacaklar
Parçalar	CPU hesaplar (ALU + kontrol birimi + yazmaçlar); RAM geçici, disk kalıcı; veri yolu: adres, veri, kontrol
İkili sayılar	bit = 0/1, 8 bit = 1 bayt; d basamakla 0 … $b^d - 1$; çevirme: 2'ye böl, kalanları aşağıdan yukarı oku
Negatifler	ikiye tümleyen: ters çevir + 1; 8 bit: −128 … 127; sınır aşılırsa sessiz taşma
Kesirli sayılar	kesir × taban^{üs}; IEEE 754 `float` / `double`; 0.1 + 0.2 ≠ 0.3
Metin	ASCII 7 bit / 128 karakter; Unicode kod noktaları (U+0000–U+10FFFF); UTF-8 1–4 bayt; Java `char` = 16 bit UTF-16
Yapılandırılmış bilgi	resim = RGB piksel dizisi, ses = basınç örnekleri, film = kare dizisi
Saklanan program	komutlar da bellekte sayı; PC sıradaki adres; Getir–Çöz–Yürüt; atlama sırayı değiştirir
Düzeyler	yüksek düzeyli dil → assembly → makine kodu (bitler); assembly işlemciye özel, her satırı genellikle tek komut

ÖZET

İlgili Sayfalar

ASCII Tablosu

0–255 arası bütün karakterlerin ondalık, onaltılık, sekizlik ve ikilik kodları; karakter ya da kodla arama, ISO-8859-1 ve Windows-1254 görünümleri.

ÖZET

Sık Yapılan Hatalar

Aralık dışı sabit

`byte b = 200;` → `error: incompatible types: possible lossy conversion from int to byte`. `byte` en fazla 127'dir.

`L` ekini unutmak

`long x = 3000000000;` → `error: integer number too large`. Doğrusu: `3000000000L`.

Karakter ile dizgiyi karıştırmak

`char c = "A";` → `error: incompatible types: String cannot be converted to char`. Karakter tek tırnakla: `'A'`.

Başta sıfır

`int x = 010;` sekizlidir, değeri **8**! `int y = 08;` ise derlenmez (`error: ';' expected`): sekizlide 8 rakamı yoktur.

Taşmayı fark etmemek

`Integer.MAX_VALUE + 1` hata vermez, **−2147483648** olur. Büyük çarpımlarda `long` gerekir: `1920L * 1080`.

Kesirlerde `==`

`0.1 + 0.2 == 0.3` → **false**. Farkın küçüklüğüne bakılır. Ayrıca `"😀".length()` 1 değil 2'dir.

ÖZET

Örnek Problemler

1. 45 sayısını 8 bit ikili olarak yazın.
2. `10110110₂` işaretsiz olarak kaçtır? 8 bit ikiye tümleyen ile işaretli olarak kaçtır?
3. −20'yi 8 bit ikiye tümleyen ile yazın.
4. `0x3F` ondalık, ikili ve sekizli olarak nedir?
5. `01101101₂ + 00110110₂` toplamını bulun. Sonuç 8 bit işaretli `byte` olarak ne olur?
6. `System.out.println('a' + 'b');` ve `System.out.println("" + 'a' + 'b');` ne yazdırır?
7. Oyuncak bilgisayarda `10011100` komutu ne yapar?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. 16 bitlik işaretli bir tam sayının (Java `short`) aralığı nedir?
2. `"Java"` dizgisinin karakter kodlarını ondalık ve ikili olarak yazın.
3. 1024 × 768 boyutlu, piksel başına 3 bayt kullanan sıkıştırmasız bir resim kaç bayttır?
4. `"ağaç"` için `length()` ve UTF-8 bayt sayısı nedir?
5. 0'dan 15'e kadar sayıları 4 basamaklı ikili olarak yazdıran bir Java programı yazın.
6. 32 hatlı bir adres yolu kaç farklı bayt adresleyebilir?

▶ **Cevaplar / ipuçları**

Kaynaklar

- D. A. Patterson ve J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, RISC-V sürümü, 2. baskı, Morgan Kaufmann, 2020.
- C. Petzold, *Code: The Hidden Language of Computer Hardware and Software*, 2. baskı, Microsoft Press, 2022.
- A. B. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly, 2019 (Bölüm 1).
- Oracle, *The Java Tutorials*: "Primitive Data Types" ve "Characters".
- J. Gosling ve diğ., *The Java Language Specification, Java SE 21*, §3.10 (Sabitler) ve §4.2 (İlkel tipler).
- The Unicode Consortium, *The Unicode Standard*, unicode.org; IEEE 754-2019, *Standard for Floating-Point Arithmetic*.
- J. von Neumann, *First Draft of a Report on the EDVAC*, 1945.

Son

Bölüm 1.1: Bilgisayarlar

Sonraki bölüm: Bölüm 1.2: Hesaplama ve Hesaplamalı Düşünme · Terim sözlüğü