Decimális → Bináris konverzió

A **105** decimális szám bináris konverziója:

```
105 ÷ 2 = 52, maradék: 1
52 ÷ 2 = 26, maradék: 0
26 ÷ 2 = 13, maradék: 0
13 ÷ 2 = 6, maradék: 1
6 ÷ 2 = 3, maradék: 0
3 ÷ 2 = 1, maradék: 1
1 ÷ 2 = 0, maradék: 1
```

A maradékokat alulról felfelé olvasva: **1101001**

Viszont te **01101001**-et írtál, ami valójában 8 bites reprezentáció (egy byte), ahol az első bit a vezető nulla.

Ellenőrzés:

01101001 = 0×128 + 1×64 + 1×32 + 0×16 + 1×8 + 0×4 + 0×2 + 1×1 = 64+32+8+1 = **105** ✓

Kód és szó konverzió példák:

ASCII karakterkódok:

- 'A' = 65 decimális = 01000001 bináris
- 'a' = 97 decimális = 01100001 bináris

Szavak (16 bit):

- 105 decimális = 0000000001101001 bináris (16 bit)
- 1000 decimális = 0000001111101000 bináris (16 bit)

Bit tárolási kapacitások

Alapvető tárolási méretek:

1 bit: 0 vagy 1 (2 lehetőség) **4 bit (nibble):** 0-15 (16 lehetőség) **8 bit (byte):** 0-255 (256 lehetőség) **16 bit (word):** 0-65,535 (65,536 lehetőség) **32 bit (dword):** 0-4,294,967,295 (~4.3 milliárd) **64 bit (qword):** 0-18,446,744,073,709,551,615 (~18.4 trillió)

Előjeles számok esetén:

8 bit előjeles: -128 től +127-ig **16 bit előjeles:** -32,768 től +32,767-ig **32 bit előjeles:** -2,147,483,648 től +2,147,483,647-ig

Bit értékek számolása

Hatványozási módszer:

```
Bit pozíció:  7  6  5  4  3  2  1  0
Érték:      128 64 32 16  8  4  2  1
```

Gyors számolási tippek:

Kettő hatványai megjegyzése:

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$
- $2^{10} = 1024$ (1K)
- $2^{20} = 1,048,576$ (1M)

Praktikus számolás:

Balról jobbra haladva: 01101001 = 64 + 32 + 8 + 1 = 105

Gyors ellenőrzés:

- Legnagyobb bit (64) + kisebb bitek összege
- Körülbelüli becslés, majd finomítás

Hexadecimális segítség:

4 bit = 1 hex digit

- 0000 = 0
- 0001 = 1
- ...
- 1111 = F (15)

Példa: 01101001 = 0110|1001 = 6|9 = 69_{16} = $6\times16 + 9 = 105_{10}$

Igen, a **105** decimális szám **7 bitet** igényel tárolásra!

Teljes számolás minden bitre:

105 → bináris konverzió:

```
105 ÷ 2 = 52, maradék: 1  ← bit 0 (legkisebb helyiérték)
52 ÷ 2 = 26, maradék: 0   ← bit 1
26 ÷ 2 = 13, maradék: 0   ← bit 2
13 ÷ 2 = 6, maradék: 1    ← bit 3
6 ÷ 2 = 3, maradék: 0     ← bit 4
3 ÷ 2 = 1, maradék: 1     ← bit 5
1 ÷ 2 = 0, maradék: 1     ← bit 6 (legnagyobb helyiérték)
```

Eredmény: 1101001 (7 bit)

Bit pozíciók:

```
Bit sorszám:  6  5  4  3  2  1  0
Bit érték:    1  1  0  1  0  0  1
Helyiérték:  64 32 16  8  4  2  1
```

Ellenőrzés: 64 + 32 + 8 + 1 = 105 ✓

Miért 7 bit?

A **legnagyobb bit** a 6. pozícióban van ($2^6 = 64$), ezért **7 bitet** (0-tól 6-ig) használunk.

Általános szabály: n decimális szám tárolásához $\lfloor \log_2(n) \rfloor + 1$ bit szükséges.

105 esetén: $\lfloor \log_2(105) \rfloor + 1 = \lfloor 6.71 \rfloor + 1 = 6 + 1 =$ **7 bit**

Kiváló kérdések! Lépésről lépésre:

7 bit → 8 bit konverzió

105 decimális:

- Minimális tárolás: **1101001** (7 bit)
- 8 bites tároláshoz: egyszerűen **vezető nullát** adunk hozzá
- **01101001** (8 bit)

Miért tároljuk 8 biten?

Számítógép architektúra:

- **Byte = 8 bit** (ez a legkisebb címezhető egység)
- Memória mindig byte-okban szervezett
- Processzorok byte-okat olvasnak/írnak egyszerre

Gyakorlati oka:

- 7 bites szám nem fér bele egy memóriacímbe
- Pazarló lenne külön kezelni a 7 bites értékeket

Maximális értékek bitszámonként:

```
1 bit:   2¹ - 1 = 1
2 bit:   2² - 1 = 3
3 bit:   2³ - 1 = 7
4 bit:   2⁴ - 1 = 15
5 bit:   2⁵ - 1 = 31
6 bit:   2⁶ - 1 = 63
7 bit:   2⁷ - 1 = 127  ← 105 belefér!
8 bit:   2⁸ - 1 = 255
16 bit:  2¹⁶ - 1 = 65,535
32 bit:  2³² - 1 = 4,294,967,295
64 bit:  2⁶⁴ - 1 = 18,446,744,073,709,551,615
```

Standard méretek:

Nem létezik 128, 256, 512 bit standard típus!

Valódi standard méretek:

- 8 bit (byte)
- 16 bit (word)
- 32 bit (dword)
- 64 bit (qword)
- 128 bit (csak speciális esetekben, pl. UUID-k)

Összefoglalva: 105 → minimum 7 bit, de **8 bitre kiegészítjük** a számítógép miatt!

1. Vezető nullák hozzáadása

NEM minden bithez adunk nullát! Csak **elöl** adjuk hozzá:

105 decimális példa:

```
7 bit:  1101001          (minimális)
8 bit:  01101001         (1 vezető nulla)
16 bit: 00000000 01101001 (9 vezető nulla)
32 bit: 00000000 00000000 00000000 01101001 (25 vezető nulla)
```

Szabály: Balról (elölről) nullákat adunk, amíg el nem érjük a kívánt bit hosszt.

2. Standard bit méretek - IGEN, ez így van!

Példák:

- **12 bit adat** → **16 bit**-en tárolva (4 vezető nulla)
- **13 bit adat** → **16 bit**-en tárolva (3 vezető nulla)
- **20 bit adat** → **32 bit**-en tárolva (12 vezető nulla)
- **50 bit adat** → **64 bit**-en tárolva (14 vezető nulla)

Standard méretek:

```
1-8 bit    → 8 bit (byte)
9-16 bit   → 16 bit (word)
17-32 bit  → 32 bit (dword)
33-64 bit  → 64 bit (qword)
```

Gyakorlati példa:

1000 decimális = **1111101000** (10 bit)

- Nem tárolhatjuk 10 biten
- **16 bitre** kell kiegészíteni: **000000** 1111101000

Miért? Mert a memória és processzorok csak ezekben a standard méretekben dolgoznak hatékonyan!

Tehát **igen**, minden "nem standard" méretet a következő nagyobb standard méretre kerekítünk fel!

Whitespace a bináris számban

A **000000** 1111101000-ben a szóköz csak **vizuális segítség**!

Célja:

- Könnyebb olvashatóság
- Elválasztja a vezető nullákat az eredeti bitoktól
- Mint ahogy a telefonszámokban: +36 30 123-4567

Valójában: 0000001111101000 (egybeírva)

Ez tényleg **cool** és praktikus a dokumentációban!

C++ vs Java sebesség

Natív C++ (bitwise nélkül)

```
int decimal = 1000;
string binary = "";
while (decimal > 0) {
    binary = (decimal % 2 ? "1" : "0") + binary;
    decimal /= 2;
}
```

Java (bitwise-zal)

```
int decimal = 1000;
String binary = Integer.toBinaryString(decimal);
```

Sebesség összehasonlítás:

Java bitwise GYORSABB lesz!

Miért?

1. **Integer.toBinaryString()** optimalizált bitwise műveleteket használ
2. C++ ciklus

   lassabb:

   - String concatenation minden iterációban
   - Division és modulo műveletek
   - Többszöri memóriaallokáció

Meglepő, de igaz: A Java beépített bitwise megoldása legyőzi a natív C++ ciklusos megközelítést!

Ha C++-ban is bitwise-t használnánk:

```
bitset<32> binary(decimal);
```

Akkor a C++ lenne gyorsabb.

Tanulság: Az algoritmus fontosabb, mint a nyelv sebessége!