Negatív és lebegőpontos számok ábrázolása

Egy mérési eredmény lehet negatív, például −5 °C, és lehet tört érték, például 13,25 mm. A számítógép memóriájában azonban csak bitek, vagyis nullák és egyesek szerepelnek. Hogyan jelöljük ezekkel a mínuszjelet és a törtrészt?

Ehhez meg kell állapodnunk a bitek értelmezésében. Először a negatív egész számok tárolását vizsgáljuk meg, majd a törtekből kiindulva eljutunk a lebegőpontos számábrázolásig és az **IEEE 754** szabvány szerinti tárolásig.

1. Kiindulás: mit jelent egy bitsorozat?

A kettes számrendszerben a helyi értékek jobbról balra kétszereződnek: 1, 2, 4, 8, 16 és így tovább. Egy bit helyén álló 1 azt jelenti, hogy az adott helyi értéket hozzáadjuk a számhoz; a 0 azt, hogy nem adjuk hozzá.

Például a `00001101` bitsorozat értéke:

Helyi érték	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Bit	0	0	0	0	1	1	0	1

A szám tehát **8 + 4 + 1 = 13**.

Nyolc bit együtt egy **bájt**. Nyolc bitből 256 különböző bitsorozat készíthető. Ha mindegyiket nemnegatív egész számként értelmezzük, 0-tól 255-ig tudunk számokat ábrázolni. Ezt nevezzük **előjel nélküli egész számábrázolásnak**.

Ha negatív számokat is szeretnénk tárolni, akkor ugyanezt a 256 bitsorozatot kell másképpen felosztanunk.

A bitsorozat önmagában nem mondja meg, hogyan kell értelmezni. Tudnunk kell, milyen számábrázolási szabály tartozik hozzá.

2. Első ötlet: külön bit jelölje az előjelet

Kézenfekvő megoldás, hogy az első bit az előjelet adja meg, a többi pedig a szám abszolút értékét, vagyis az előjel nélküli nagyságát.

Állapodjunk meg abban, hogy az első bit 0 esetén pozitív, 1 esetén negatív előjelet jelent. Nyolc bitből így egy az előjelé, hét pedig a nagyságé:

Szám	**Előjelbit**	**A nagyság hét biten**	**Teljes bitsorozat**
+5	0	`0000101`	`00000101`
−5	1	`0000101`	`10000101`

Ezt **előjeles abszolútértékes ábrázolásnak** nevezzük. Könnyű megérteni, de az egész számokkal végzett műveleteknél kényelmetlen:

- A nullának két alakja van: `00000000` és `10000000`, azaz +0 és −0.
- A bitsorozatok közönséges bináris összeadása önmagában nem adja meg helyesen az előjeles összeget.

Például +5 és −5 matematikai összege 0. A fenti két bitsorozat szokásos bináris összeadása viszont `10001010`, ami ebben az ábrázolásban −10 lenne.

Az egész számokhoz ezért általában más megoldást használunk: a **kettes komplemens ábrázolást**.

3. Kettes komplemens

Olyan bitsorozatot keresünk −5 számára, amelyet +5 bitsorozatához hozzáadva az alsó nyolc biten nullát kapunk.

Nyolc biten legfeljebb 255 ábrázolható előjel nélkül. A következő szám, 256, már kilenc bitet igényel: `100000000`. Ha egy nyolcbites eredményből csak az alsó nyolc bitet tartjuk meg, ebből `00000000` marad.

Melyik nemnegatív számmal kell az 5-öt kiegészíteni 256-ra? A válasz: **256 − 5 = 251**.

Ezért nyolcbites kettes komplemensben a −5 tárolt bitmintája megegyezik az előjel nélkül 251-et jelentő bitmintával: **11111011**.

Hogyan állítjuk elő a negatív szám bitmintáját?

Az eljárás két egyszerű lépésből áll, de **előtte rögzíteni kell a bitszámot**. Most nyolc bittel dolgozunk.

1. Felírjuk a pozitív számot nyolc biten.
2. Minden bitet megfordítunk: a 0-ból 1, az 1-ből 0 lesz.
3. Az eredményhez hozzáadunk 1-et, és nyolc bitet tartunk meg.

Példa: −5 előállítása.

```
+5 nyolc biten:       00000101
Bitek megfordítása:  11111010
Egy hozzáadása:      11111011
```

Az eredmény tehát **−5 = 11111011** nyolcbites kettes komplemensben.

Miért működik a megfordítás és az egy hozzáadása? A nyolc egyesből álló szám értéke 255. Egy pozitív x szám bitjeinek megfordítása 255 − x-et ad, az ezt követő +1 pedig 256 − x-et. Pontosan ezt kerestük.

Hogyan olvassuk vissza a számot?

Nyolcbites kettes komplemensben az első bit helyi értéke **−128**, a többi bité változatlanul pozitív:

Helyi érték	**−128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
−5 bitmintája	1	1	1	1	1	0	1	1

Az érték: **−128 + 64 + 32 + 16 + 8 + 2 + 1 = −5**.

Ezért az első bit valóban jelzi az előjelet: 0 esetén nemnegatív, 1 esetén negatív a szám. De **nem egyszerűen külön mínuszjel a többi bit elé**. A maradék hét bit itt nem a negatív szám abszolút értékét tárolja.

Másik visszaolvasási módszer: az `11111011` előjel nélkül 251. Mivel az első bit 1, kivonunk 256-ot: **251 − 256 = −5**.

4. Miért hasznos a kettes komplemens?

A pozitív és negatív számok bitmintáit ugyanazzal a bináris összeadással kezelhetjük. Az összeadásban 1 + 1 eredménye 0 az adott helyen, és 1-et továbbviszünk a következő helyi értékre. Ezt **átvitelnek** nevezzük.

Vizsgáljuk meg az 5 + (−5) összeget:

```
  00000101     +5
+ 11111011     -5
----------
1 00000000
```

A bal oldali kilencedik bit nem fér bele a nyolcbites eredménybe. Az alsó nyolc bit `00000000`, vagyis **0**, ahogyan vártuk.

A kivonás is összeadássá alakítható: **7 − 5 = 7 + (−5)**.

```
  00000111     +7
+ 11111011     -5
----------
1 00000010
```

Az alsó nyolc bit értéke **2**.

Fontos: A helyes előjeles eredményhez az is szükséges, hogy a matematikai eredmény beleférjen az adott bitszám értéktartományába. A kilencedik bit elhagyása önmagában nem oldja meg a túl nagy eredmények problémáját.

5. Mekkora számok férnek el?

Nyolcbites kettes komplemensben:

- a legnagyobb pozitív érték `01111111`, vagyis **127**;
- a legkisebb érték `10000000`, vagyis **−128**;
- a nulla egyetlen alakja `00000000`.

A tartomány azért nem szimmetrikus, mert az első bit 0-val kezdődő 128 lehetősége közé a nulla is beletartozik: 0–127. Az 1-gyel kezdődő 128 lehetőség a −128 és −1 közötti számoké.

Bitszám	Előjel nélküli egész	Kettes komplemens egész
8	0 … 255	−128 … 127
16	0 … 65 535	−32 768 … 32 767
32	0 … 4 294 967 295	−2 147 483 648 … 2 147 483 647

Általánosan n bit esetén a kettes komplemens tartománya $\mathbf{-2^{n-1} \ldots 2^{n-1} - 1}$. A felső index hatványozást jelent.

Ugyanaz a bitsorozat eltérő szabállyal mást jelent:

Bitsorozat	Előjel nélkül	Kettes komplemens szerint
01111111	127	127
10000000	128	−128
11111111	255	−1

A −128 pozitív párja, +128 nem fér el nyolcbites kettes komplemensben. Ezért a legkisebb negatív érték előjelének megfordítása is túlcsordulhat.

Mi történik, ha több bitet használunk?

Ha egy kettes komplemens számot változatlan értékkel több bitre bővítünk, az új bal oldali biteket a régi első bit értékével töltjük ki. Ezt **előjelkiterjesztésnek** nevezzük.

Szám	Nyolc biten	Tizenhat biten
+5	00000101	00000000 00000101
−5	11111011	11111111 11111011

A szóköz csak az olvashatóságot segíti. Negatív szám elé tehát nem nullákat írunk, mert az megváltoztatná az értékét.

6. Hogyan írunk fel törteket kettes számrendszerben?

Tízes számrendszerben a vessző utáni helyi értékek: egy tized, egy század, egy ezred. Kettes számrendszerben ugyanígy folytatódik a helyiérték-rendszer, de minden új helyen kettővel osztunk:

Hely	A vessző előtt	Első törtjegy	Második törtjegy	Harmadik törtjegy	Negyedik törtjegy
Helyi érték	1	1/2	1/4	1/8	1/16
Tízes számrendszerben	1	0,5	0,25	0,125	0,0625

Például a bináris **0,101** értéke:

1 × 1/2 + 0 × 1/4 + 1 × 1/8 = 0,625.

A következőkben a szám mögötti alsó 2 a kettes, az alsó 10 a tízes számrendszert jelöli. Így az előző összefüggés:

$0{,}101_2 = 0{,}625_{10}$.

Ez nem azonos a tízes számrendszerbeli 0,101-gyel. A vessző mindkét esetben az egész és a tört részt választja el, de a helyi értékek különböznek.

Példa: 13,25 felírása binárisan

Az egész részt és a tört részt külön vizsgáljuk:

- 13 = 8 + 4 + 1, ezért az egész rész `1101`.
- 0,25 = 1/4, ezért a tört rész `01`.

Tehát **$13{,}25_{10} = 1101{,}01_2$**.

Ellenőrzés: 8 + 4 + 1 + 1/4 = 13,25.

Általános módszer: a törtrész szorzása kettővel

A törtjegyeket ismételt kettővel szorzással is meghatározhatjuk. Minden lépésben felírjuk a szorzat egész részét, majd a következő lépésben csak a megmaradt törtrésszel dolgozunk.

Például 0,625 esetén:

Lépés	Szorzás	Következő bináris jegy	Továbbvitt törtrész
1.	0,625 × 2 = 1,25	1	0,25
2.	0,25 × 2 = 0,5	0	0,5
3.	0,5 × 2 = 1,0	1	0

A jegyeket keletkezésük sorrendjében olvassuk: **$0{,}101_2$**. A maradék nulla lett, ezért a felírás véget ért.

7. Hol tároljuk a vesszőt?

A memóriában nincs a bitek közé rajzolt vessző. A helyét is értelmezési szabállyal kell megadnunk.

Az egyik lehetőség, hogy előre rögzítjük, hány bit tartozik a törtrészhez. Ezt **fixpontos számábrázolásnak** nevezzük.

Például használjunk nyolc előjel nélküli bitet, és állapodjunk meg abban, hogy az utolsó négy bit a törtrész. A tárolt `00110100` bitmintát ilyenkor `0011,0100` alakban értelmezzük:

2 + 1 + 1/4 = 3,25.

Ugyanez megfogalmazható úgy is, hogy a tárolt egész szám 52, és annak értékét 16-tal osztjuk: **52/16 = 3,25**.

Ebben az egyszerű formátumban két szomszédos szám távolsága mindig 1/16 = 0,0625. A legnagyobb szám 255/16 = 15,9375. Több törtbit finomabb felbontást adna, de ugyanennyi összes bit mellett kisebb egész rész férne el.

A fixpontos ábrázolás állandó lépésközt használ. Ez hasznos lehet, ha a mérési tartomány és a szükséges felbontás előre ismert. Mi történjen azonban, ha nagyon kicsi és nagyon nagy számokkal is szeretnénk dolgozni?

8. A lebegőpontos ábrázolás alapötlete

A mérnöki gyakorlatban gyakran használunk hatványalakot. Például:

- $13\,250 = 1{,}325 \times 10^{4}$;
- $0{,}001325 = 1{,}325 \times 10^{-3}$.

Mindkét számban az 1,325 hordozza az értékes számjegyeket. A tíz hatványa mondja meg, mekkora legyen a szám. A **kitevő** az a szám, amelyet a hatvány alapjának felső indexébe írunk: itt 4, illetve −3.

A negatív kitevő osztást jelent: $10^{-3} = 1/1000$. Ugyanígy $2^{-3} = 1/8$.

A **lebegőpontos számábrázolás** ezt az ötletet használja: külön tárolja a szám előjelét, értékes jegyeit és a nagyságrendet meghatározó kitevőt. A vessző helye a kitevőtől függ, ezért mondjuk, hogy „lebeg”.

A bináris lebegőpontos számoknál a hatvány alapja **2**, nem 10.

A szám rendezése egységes alakba

A 13,25 korábban kapott bináris alakja `1101,01`. A vesszőt három hellyel balra helyezve `1,10101`-et kapunk. Ez nyolcadakkora szám, ezért 2^{3}-nal, azaz nyolccal kell szoroznunk:

$$\mathbf{1101{,}01_2 = 1{,}10101_2 \times 2^{3}}.$$

A nem nulla szám abszolút értékét úgy rendezzük át, hogy a vessző előtt pontosan egy 1-es maradjon. Ezt az alakot **normalizált alaknak**, az átrendezést **normalizálásnak** nevezzük.

Az `1,10101` részt **szignifikansnak** nevezzük; sok tananyag a **mantissza** elnevezést használja rá. Ez hordozza a szám értékes bináris jegyeit. Értéke normalizált alakban legalább 1, de kisebb 2-nél.

Egy egynél kisebb számnál negatív kitevőt kapunk:

$$\mathbf{0{,}00101_2 = 1{,}01_2 \times 2^{-3} = 0{,}15625_{10}}.$$

Itt a vesszőt jobbra mozdítottuk három hellyel, ezért az érték megtartásához nyolccal osztunk.

9. IEEE 754: pontosan melyik bit mit tároljon?

Az alapötlet még nem határozza meg, hány bitet kapjon az előjel, a kitevő és a szignifikáns. Ha különböző számítógépek ugyanúgy szeretnék értelmezni az adatot, közös szabályokra van szükségük.

Az **IEEE 754** a lebegőpontos számábrázolás és aritmetika szabványa. Többek között tárolási formátumokat, kerekítési szabályokat és különleges értékeket határoz meg. Itt a **binary32** nevű, 32 bites bináris formátumot vezetjük le részletesen. Ezt egyszeres pontosságú ábrázolásnak is nevezik.

A 32 bitet három egymás utáni részre osztjuk. Az ilyen, meghatározott szerepű bitcsoportot **mezőnek** nevezzük.

Mező	Méret	Feladat
Előjel	1 bit	0: pozitív, 1: negatív előjel
Eltolt kitevő	8 bit	A kettes hatvány kitevőjének tárolása, az alább megadott eltolással
Törtrész	23 bit	A normalizált szignifikáns vessző utáni bitjei

Ez összesen **1 + 8 + 23 = 32 bit**, vagyis négy bájt. A következő levezetések a normál, véges, nem nulla számokra vonatkoznak. A nulla és más különleges esetek szabályait később külön tárgyaljuk.

Miért csak a vessző utáni biteket tároljuk?

A normalizált bináris alak mindig `1,`-gyel kezdődik. Ezt az első 1-est ezért nem szükséges külön eltárolni: visszaolvasáskor automatikusan eléírjuk a tárolt törtbiteknek.

Például az `1,10101` szignifikansból csak az `10101` rész kerül a törtrészmezőbe. A hiányzó helyeket jobbról nullákkal töltjük ki.

Az elhagyott első 1-est **rejtett vezető bitnek** nevezik. Emiatt a 23 tárolt törtbit a normál számoknál **24 értékes bináris jegyet** biztosít. A szignifikans és a tárolt törtrészmező tehát nem pontosan ugyanaz.

Hogyan tárolunk negatív kitevőt?

A kitevő lehet pozitív és negatív is. A binary32 kitevőmezőjét azonban **nem kettes komplemens szerint** értelmezzük. Ehelyett a valódi kitevőhöz hozzáadunk **127-et**, és ezt a nemnegatív értéket tároljuk.

Ezt az állandó hozzáadást **eltolásnak** nevezzük; angol neve bias.

- **Tárolás:** tárolt kitevő = valódi kitevő + 127.
- **Visszaolvasás:** valódi kitevő = tárolt kitevő − 127.

Valódi kitevő	Tárolt érték	A tárolt érték nyolc biten
−3	124	`01111100`
0	127	`01111111`
+3	130	`10000010`

A normál számok tárolt kitevője 1 és 254 közötti lehet, ami −126 és +127 közötti valódi kitevőt jelent. A 0 és a 255 mezőérték különleges esetek számára van fenntartva.

10. Teljes példa: −13,25 tárolása binary32 formátumban

1. lépés: az előjel

A szám negatív, ezért az **előjelbit 1**. A továbbiakban a 13,25 abszolút értékével dolgozunk.

2. lépés: bináris alak és normalizálás

A korábbi levezetés szerint:

$13{,}25_{10} = 1101{,}01_2 = 1{,}10101_2 \times 2^3$.

A valódi kitevő tehát **3**, a szignifikáns **1,10101**.

3. lépés: a kitevőmező

A 3-hoz hozzáadjuk az eltolást: **3 + 127 = 130**.

Mivel 130 = 128 + 2, a kitevőmező nyolc bitje **10000010**.

4. lépés: a törtrészmező

Az `1,10101` elejéről elhagyjuk a rejtett 1-est. A maradó `10101` öt bit hosszú; jobbról 18 nullával egészítjük ki 23 bitre:

10101000000000000000000.

Itt nincs szükség kerekítésre, mert az összes értékes jegy elfér.

5. lépés: a mezők összeillesztése

Előjel: 1 bit	Kitevő: 8 bit	Törtrész: 23 bit
1	10000010	10101000000000000000000

A teljes 32 bites minta:

```
1 10000010 10101000000000000000000
```

A szóközök csak a mezőhatárokat mutatják, nem tárolt jelek. A bitmintát a legfelső előjelbittől a legalsó törtbit felé írtuk fel; a memóriabeli bájtsorrend külön kérdés.

Fontos különbség: itt a negatív előjelet egy külön bit tárolja. A −13,25 előállításához nem fordítjuk meg a +13,25 teljes bitmintáját, és nem adunk hozzá 1-et. A kettes komplemens az egész számoknál tanult másik ábrázolás.

Visszaellenőrzés: mit jelent ez a 32 bit?

1. Az előjelbit 1, ezért az eredmény negatív.
2. A kitevőmező `10000010`, vagyis 130. A valódi kitevő 130 − 127 = 3.
3. A törtmező elé visszaírjuk a rejtett 1-est: `1,10101`.
4. A szignifikáns értéke 1 + 1/2 + 1/8 + 1/32 = 1,65625.
5. Megszorozzuk 2^3-nal: 1,65625 × 8 = 13,25.
6. Alkalmazzuk a negatív előjelet: **−13,25**.

A szabály tömör képlettel

Jelöljük az előjelbitet s-sel, a kitevőmező előjel nélküli egész értékét E-vel, a 23 bites törtrészmező előjel nélküli egész értékét pedig F-fel. Normál binary32 számnál:

érték = $(-1)^s \times (1 + F / 2^{23}) \times 2^{E-127}$.

A képlet ugyanazokat a lépéseket foglalja össze, amelyeket az előbb külön elvégeztünk. Csak akkor alkalmazható ebben az alakban, ha **1 ≤ E ≤ 254**.

11. Második példa: egy kicsi pozitív szám

Ábrázoljuk a **0,15625** számot! Mivel 0,15625 = 1/8 + 1/32, bináris alakja `0,00101`.

Normalizálva: **$1{,}01_2 \times 2^{-3}$**.

- Az előjelbit **0**, mert a szám pozitív.
- A valódi kitevő **−3**, így a tárolt kitevő −3 + 127 = **124**, azaz `01111100`.
- A rejtett 1-es utáni törtbitek `01`, amelyeket 23 bitre egészítünk ki.

Előjel	Kitevő	Törtrész
0	01111100	01000000000000000000000

Visszaolvasva a szignifikáns 1,25, a kettes hatvány 1/8, ezért az eredmény **1,25 / 8 = 0,15625**.

12. Miért nem minden szám tárolható pontosan?

A tízes számrendszerben az 1/3 felírása végtelen: 0,3333… Kettes számrendszerben is vannak végtelen felírású törtek, de nem pontosan ugyanazok.

Például a **$0{,}1_{10}$ bináris alakja $0{,}0001100110011\ldots_2$**. A `0011` rész az első törtjegy után ismétlődik.

Az ismételt kettővel szorzásnál is látszik az ismétlődés:

Lépés	Szorzás	Kiírt bit	Maradó törtrész
1.	0,1 × 2 = 0,2	0	0,2
2.	0,2 × 2 = 0,4	0	0,4
3.	0,4 × 2 = 0,8	0	0,8
4.	0,8 × 2 = 1,6	1	0,6
5.	0,6 × 2 = 1,2	1	0,2

Az ötödik lépésben újra 0,2 maradt, ezért innentől ugyanaz a folyamat ismétlődik. A szám felírása nem ér véget.

Véges számú biten csak véges számú jegyet tárolhatunk, ezért **közelítő értéket** választunk. A binary32 formátumban a 0,1 legközelebbi tárolható értéke körülbelül **0,10000000149**.

Ez nem meghibásodás. Hasonló ahhoz, amikor az 1/3 helyett csak 0,333-at írunk le.

A kerekítés alapelve

Az IEEE 754 alapértelmezett bináris kerekítése a **legközelebbi ábrázolható értéket** választja. Pontos félútnál azt az értéket részesíti előnyben, amelynek utolsó megtartott bináris jegye 0. Ezt „legközelebbire, döntetlennél párosra” kerekítésnek nevezik.

Ezért a tárolás nem minden esetben puszta levágás: a megtartott utolsó jegyek a kerekítés hatására megváltozhatnak.

A pontosság nem a tizedesjegyek állandó számát jelenti

A binary32 normál számoknak 24 értékes bináris jegyük van. Amikor a kitevő nő, ugyanezek a jegyek nagyobb helyi értékeket jelölnek. Ezért a nagyobb számok között általában nagyobb a távolság.

Például binary32 esetén:

- 1 és 2 között a szomszédos számok távolsága 2^{-23}, körülbelül 0,0000001192.
- 2^{24} = 16 777 216 és 2^{25} között a lépésköz már **2**.

A **16 777 217** egész szám ezért nem tárolható pontosan binary32-ben, pedig nem közelíti meg a formátum legnagyobb értékét. A szomszédai 16 777 216 és 16 777 218. Az alapértelmezett kerekítés itt a 16 777 216-ot választja.

Az ábrázolható tartomány és az ábrázolás pontossága külön tulajdonság. Attól, hogy egy érték a tartományon belül van, még nem feltétlenül tárolható pontosan.

13. Nulla, nagyon kicsi számok és különleges értékek

A normál alak mindig rejtett 1-essel kezdődik, ezért ezzel a szabállyal a nullát nem tudnánk leírni. A szabvány ezért bizonyos kitevőmintákhoz külön értelmezést rendel.

A következő táblázatban E a kitevőmező egész értéke, F pedig a törtrészmezőé:

Tárolt kitevő, E	Törtrész, F	Jelentés
1–254	Tetszőleges	Normál, véges, nem nulla szám; rejtett vezető bitje 1.
0	0	Nulla; az előjelbittől függően +0 vagy −0.
0	Nem nulla	Szubnormális, azaz a normál tartománynál kisebb abszolút értékű, nem nulla szám.
255	0	Pozitív vagy negatív végtelen az előjelbit szerint.
255	Nem nulla	NaN: különleges „nem szám” érték.

A **szubnormális számoknál** a vezető bit 0, a kettes hatvány kitevője pedig rögzítetten −126. Így a normál tartomány alatt is tárolhatók nullától különböző értékek, de kevesebb értékes jeggyel. A legkisebb pozitív binary32 érték 2^{-149}, körülbelül $1{,}4 \times 10^{-45}$.

A szubnormális érték képlete: $\mathbf{(-1)^s \times (F / 2^{23}) \times 2^{-126}}$. Itt nem írunk vissza rejtett 1-est.

A **végtelen** jelölés például túl nagy eredménynél jelenhet meg. Nem egy rendkívül nagy véges számot jelent, hanem különleges érték.

A **NaN** az angol Not a Number rövidítése. Olyan eredményt jelölhet, amelyhez nem rendelhető szokásos számérték, például a 0/0 műveletnél. A program konkrét szabályai szerint ilyen helyzetben különleges érték vagy hibajelzés is keletkezhet.

A +0 és −0 a szokásos numerikus egyenlőségvizsgálatban egyenlő, de bitmintájuk eltér, és egyes műveletek különbséget tesznek közöttük. Ez az egész számok kettes komplemens ábrázolásának egyetlen nullájától eltérő szabály.

14. Mi változik 64 bites lebegőpontos ábrázolásnál?

Az IEEE 754 másik gyakori formátuma a **binary64**, más néven kétszeres pontosságú ábrázolás. Ugyanarra az elvre épül, de több bitet használ a kitevőhöz és a törtrészhez is.

Tulajdonság	binary32	binary64
Teljes méret	32 bit, 4 bájt	64 bit, 8 bájt
Előjel	1 bit	1 bit
Kitevőmező	8 bit	11 bit
Tárolt törtbitek	23 bit	52 bit
Értékes bináris jegyek normál számnál	24	53
Kitevő eltolása	127	1023
Tájékoztató pontosság tízes számrendszerben	Körülbelül 7 értékes jegy	Körülbelül 16 értékes jegy
Legnagyobb pozitív véges érték, közelítőleg	$3{,}4 \times 10^{38}$	$1{,}8 \times 10^{308}$

Az „értékes jegy” nem ugyanaz, mint a „vessző utáni jegy”. Például az 1,234567 és az 1 234 567 egyaránt hét értékes decimális jegyet tartalmaz. A táblázat kerekített tájékoztatást ad, nem minden decimális számra érvényes pontos tárolási garanciát.

A binary64 lényegesen pontosabb, de **a 0,1-et ez sem tárolja pontosan**, mert annak bináris felírása végtelen.

15. Mire figyeljünk mérnöki számításoknál?

- **Egész számoknál ellenőrizzük a tartományt.** Például egy számláló tárolásakor fontos, elfér-e benne a legnagyobb várható érték.
- **A lebegőpontos eredményben lehet kerekítési eltérés.** Ez nem azonos a műszer mérési hibájával, bár a számítás során mindkettő jelentős lehet.
- **Nagy érték mellett egy kis változás elveszhet.** Ha a változás kisebb a helyi lépésköznél, kerekítés után ugyanaz a szám maradhat.
- **A számítás sorrendje is számíthat.** A részeredmények kerekítése miatt az összeadások átcsoportosítása eltérő eredményt adhat.
- **Számított vagy mért értékek egyezését gyakran tűréssel vizsgáljuk.** A tűrést a mértékegységhez, a nagyságrendhez és a feladat pontossági követelményéhez kell választani.

Például ha két hosszúságeredmény eltérését legfeljebb 0,001 mm-ig tekintjük elfogadhatónak, azt vizsgáljuk, hogy különbségük abszolút értéke legfeljebb 0,001 mm-e. Más nagyságrendű feladatnál relatív, az érték nagyságához viszonyított tűrésre is szükség lehet.

A több megjelenített tizedesjegy nem teszi pontosabbá sem az eredeti mérést, sem a már elvégzett számítást.

Ellenőrző kérdések és feladatok

1. Írja fel a −6 értéket nyolcbites kettes komplemens ábrázolással!
2. Mit jelent az `11110110` nyolcbites kettes komplemens szerint?
3. Miért okoz problémát a 100 + 40 művelet nyolcbites kettes komplemensben?
4. Írja fel a 0,375 számot bináris tört alakban!
5. Írja fel a 6,5 számot normalizált bináris alakban!
6. Adja meg a +6,5 három mezőjét IEEE 754 binary32 formátumban!
7. Az előző bitmintában mely bitek változnak meg, ha +6,5 helyett −6,5-et tárolunk?
8. Miért nem tárolható pontosan a 0,1 sem binary32, sem binary64 formátumban?
9. Hogyan lehetséges, hogy egy tartományon belüli egész szám sem tárolható pontosan lebegőpontos formában?

Megoldások

1. +6: `00000110`; megfordítva: `11111001`; egyet hozzáadva: **11111010**.
2. Előjel nélkül 246, ezért kettes komplemens szerint 246 − 256 = **−10**.
3. A matematikai eredmény 140, de a legnagyobb ábrázolható érték 127. Túlcsordulás történik.
4. 0,375 = 1/4 + 1/8, ezért $\mathbf{0{,}011_2}$.
5. $6{,}5 = 110{,}1_2 = \mathbf{1{,}101_2 \times 2^2}$.
6. Előjel: `0`; kitevő: 2 + 127 = 129, azaz `10000001`; törtrész: `10100000000000000000000`.
7. Csak az előjelbit változik 0-ról 1-re.
8. A 0,1 bináris alakja végtelen szakaszos tört, a tárolható bitek száma pedig véges.
9. A tartomány a nagyságrendet korlátozza, a szignifikáns hossza pedig a pontosságot. Nagy értékeknél a szomszédos tárolható számok között egynél nagyobb távolság is lehet.

Források és további olvasnivaló

- Oracle Numerical Computation Guide: IEEE-formátumok és különleges értékek
- David Goldberg: What Every Computer Scientist Should Know About Floating-Point Arithmetic

#szam-demo { max-width: 1000px; margin: 20px auto; padding: 22px; box-sizing: border-box; border: 1px solid #cbd5e1; border-radius: 14px; background: #f8fafc; color: #172033; font: 16px/1.6 Arial, sans-serif; } #szam-demo * { box-sizing: border-box; } #szam-demo h3 { margin: 0 0 12px; color: #172033; } #szam-demo p { margin: 10px 0; } #szam-demo button, #szam-demo input { font: inherit; } #szam-demo button { padding: 8px 13px; border: 1px solid #94a3b8; border-radius: 7px; background: white; color: #172033; cursor: pointer; } #szam-demo button:hover { background: #e2e8f0; } #szam-demo button:focus-visible, #szam-demo input:focus-visible { outline: 3px solid #2563eb; outline-offset: 2px; } #szam-demo .navigation, #szam-demo .controls, #szam-demo .presets { display: flex; flex-wrap: wrap; gap: 8px; align-items: center; margin-bottom: 15px; } #szam-demo .navigation button[aria-pressed="true"] { background: #1e40af; border-color: #1e40af; color: white; } #szam-demo input[type="text"], #szam-demo input[type="number"] { width: 180px; max-width: 100%; padding: 8px; border: 1px solid #94a3b8; border-radius: 6px; background: white; color: #172033; } #szam-demo input[type="range"] { display: block; width: 100%; margin: 18px 0; accent-color: #1e40af; } #szam-demo .bits { display: flex; flex-wrap: wrap; gap: 5px; margin: 10px 0; } #szam-demo .bit { width: 44px; min-height: 48px; padding: 5px; font: bold 20px monospace; } #szam-demo .bit[aria-pressed="true"] { background: #1e40af; color: white; border-color: #1e40af; } #szam-demo .bit small { display: block; font: 11px Arial, sans-serif; } #szam-demo .result { margin: 14px 0; padding: 14px; border-left: 4px solid #2563eb; border-radius: 5px; background: #eaf1ff; overflow-wrap: anywhere; } #szam-demo .mono { font-family: Consolas, monospace; overflow-wrap: anywhere; } #szam-demo .error { color: #a61b1b; font-weight: bold; } #szam-demo .hint { color: #475569; font-size: 14px; } #szam-demo .field { margin-top: 16px; padding: 12px; border-radius: 8px; background: white; border: 1px solid #dbe3ed; } #szam-demo .field-title { font-weight: bold; } #szam-demo [hidden] { display: none !important; } @media (max-width: 500px) { #szam-demo { padding: 12px; } #szam-demo .bit { width: 38px; } }
Negatív egész számok IEEE 754 lebegőpontos számok

Nyolc bit – kétféle értelmezés

Kattints a bitekre! Ugyanazt a bitsorozatot előjel nélküli egészként és kettes komplemens szerint is értelmezzük.

Előjeles érték: -5 Beállítás

-5

−128 −5 −1 0 +5 +127

Előjel nélkül:

Kettes komplemens szerint:

Adjunk hozzá 1-et! Bitek megfordítása és +1

Próbáld ki: állíts be 127-et, majd adj hozzá 1-et! A demonstráció nyolc bitet tart meg.

Hogyan lesz egy számból 32 bit?

Írj be egy számot, vagy válassz példát! A program IEEE 754 binary32 formátumra alakítja. A kapott biteket külön is átkapcsolhatod.

Szám: -13,25 Átalakítás
−13,25 0,15625 0,1 16 777 217 +0 −0 +∞ NaN

1. Előjel – 1 bit
2. Kitevő – 8 bit
3. Törtrész – 23 bit

Típus:
Tárolt érték:
A véges, nem nulla értéket 17 értékes decimális jegyre írjuk ki.

A 0,1 példán figyeld meg a kerekítést! Normál számnál a tárolt törtbitek elé egy rejtett 1-es kerül. Nulla és szubnormális szám esetén a vezető bit 0.

A demonstráció használatához engedélyezett JavaScript szükséges.

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:infrendalapjai_architekturak:logika_alapjai:szamabrazolas

Last update: **2026/10/03 18:17**