Negatív és lebegőpontos számok ábrázolása kettes számrendszerben

A számítógép memóriája biteket tárol, ezért a pozitív egész számok mellett a negatív és a törtrészt tartalmazó számokat is 0-k és 1-ek sorozatával kell ábrázolni. Ehhez előre meg kell határozni, hogy az egyes biteknek mi a jelentésük.

Fontos, hogy ugyanaz a bitsorozat önmagában még nem határozza meg a tárolt szám értékét. A `11111111` bitsorozat jelenthet például:

- előjel nélküli egész számként 255-öt;
- 8 bites kettes komplemens előjeles számként −1-et;
- egy hosszabb adat, karakter vagy utasítás részét.

Az értelmezéshez tehát ismernünk kell az adattípust és a felhasznált ábrázolási módot.

1. Előjel nélküli egész számok

Előjel nélküli ábrázolásnál minden bit a szám nagyságát adja meg. Egy 8 bites szám helyiértékei:

Helyiérték	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Érték	128	64	32	16	8	4	2	1

Például:

$$00101101_2 = 32 + 8 + 4 + 1 = 45_{10}$$

n biten összesen 2^n különböző bitminta tárolható. Az előjel nélküli számok értéktartománya:

$$0 \ldots 2^n - 1$$

8 bit esetén ez:

```
0 ... 255
```

2. Negatív egész számok ábrázolása

A negatív számok tárolására többféle módszert dolgoztak ki. A legismertebbek:

- előjel–nagyságos ábrázolás;
- egyes komplemens ábrázolás;
- kettes komplemens ábrázolás.

A modern számítógépek szinte mindig a *kettes komplemens* ábrázolást használják az előjeles egész számok tárolására.

2.1. Előjel–nagyságos ábrázolás

Ebben a módszerben a legmagasabb helyiértékű, bal szélső bit az előjelet jelöli:

- 0: pozitív szám;
- 1: negatív szám.

A többi bit a szám abszolút értékét tartalmazza.

8 biten például:

```
+5 = 00000101
−5 = 10000101
```

A módszer könnyen értelmezhető, de kétféle nulla tartozik hozzá:

```
+0 = 00000000
−0 = 10000000
```

Az összeadás és kivonás végrehajtása is körülményes, mert az előjelet külön kell kezelni. Emiatt egész számok tárolására ma már ritkán használják.

2.2. Egyes komplemens ábrázolás

Egy negatív szám egyes komplemensét úgy kapjuk meg, hogy a megfelelő pozitív szám minden bitjét megfordítjuk:

```
+5 = 00000101
−5 = 11111010
```

A bitenkénti megfordítás során minden 0-ból 1, minden 1-ből 0 lesz.

Ennél a módszernél is kétféle nulla létezik:

```
+0 = 00000000
−0 = 11111111
```

Ez a tulajdonság, valamint az aritmetikai műveletek bonyolultsága miatt az egyes komplemenst a mai általános célú számítógépek egész számok tárolására jellemzően nem használják.

2.3. Kettes komplemens ábrázolás

A kettes komplemens ábrázolás előnye, hogy csak egyetlen nullaérték létezik, és ugyanaz az összeadó áramkör használható pozitív és negatív számok összeadására.

Egy negatív szám kettes komplemens alakjának előállítása:

1. Írjuk fel a szám pozitív abszolút értékét a megadott bitszámon.
2. Fordítsuk meg az összes bitet.
3. Adjunk az eredményhez 1-et.

Például a −5 előállítása 8 biten:

```
  00000101   +5 bináris alakja
  11111010   a bitek megfordítása
+        1
----------
  11111011   −5 kettes komplemens alakja
```

Tehát:

```
+5 = 00000101
−5 = 11111011
```

Negatív kettes komplemens szám visszaalakítása

Ha egy kettes komplemens számban a bal szélső bit 1, akkor a szám negatív. Az abszolút érték meghatározásához:

1. fordítsuk meg az összes bitet;
2. adjunk hozzá 1-et;
3. az eredményt lássuk el negatív előjellel.

Például:

```
  11110110   ismeretlen negatív szám
  00001001   a bitek megfordítása
+        1
----------
  00001010   10
```

Ezért:

$$11110110_2 = -10_{10}$$

Egy másik gyors módszer szerint a 8 bites negatív szám bitsorozatát először előjel nélküli számként értelmezzük, majd kivonjuk belőle $2^8 = 256$ értékét:

```
11110110₂ = 246 előjel nélküli számként
246 − 256 = −10
```

A kettes komplemens értéktartománya

n bites kettes komplemens szám értéktartománya:

$$-2^{n-1} \ldots 2^{n-1} - 1$$

Néhány gyakori eset:

Bitszám	Legkisebb érték	Legnagyobb érték
8 bit	−128	127
16 bit	−32 768	32 767
32 bit	−2 147 483 648	2 147 483 647
64 bit	-2^{63}	$2^{63} - 1$

A tartomány azért nem szimmetrikus, mert a nulla is a nemnegatív bitminták egyikét foglalja el. 8 biten például:

```
01111111 =  127
00000000 =    0
11111111 =   −1
10000000 = −128
```

2.4. Összeadás kettes komplemens számokkal

Kettes komplemens ábrázolásnál a pozitív és negatív számok a szokásos bináris összeadással összeadhatók.

Számítsuk ki 8 biten az $5 + (-3)$ műveletet:

```
  00000101    +5
+ 11111101    −3
----------
1 00000010
```

A kilencedik bitre kerülő átvitelt elhagyjuk, mert a számábrázolás csak 8 bites. Az eredmény:

$$00000010 = 2$$

Valóban:

$$5 + (-3) = 2$$

Kivonás összeadással

Az $A - B$ kivonás elvégezhető úgy, hogy A értékéhez hozzáadjuk B kettes komplemensét:

$$A - B = A + (-B)$$

Például $7 - 5$:

```
  00000111    +7
+ 11111011    −5
----------
1 00000010     2
```

2.5. Túlcsordulás

Túlcsordulás akkor történik, ha a matematikai eredmény nem fér el a rendelkezésre álló bitszámon.

8 bites előjeles szám esetén a legnagyobb tárolható érték 127. Ha ehhez 1-et adunk:

```
  01111111    127
+ 00000001      1
----------
  10000000   −128-ként értelmezhető bitminta
```

A matematikai eredmény 128 lenne, de ez 8 bites előjeles számban nem ábrázolható.

Előjeles összeadásnál túlcsordulás történt, ha:

- két pozitív szám összege negatívnak látszik; vagy
- két negatív szám összege pozitívnak látszik.

Különböző előjelű számok összeadásakor nem keletkezhet előjeles túlcsordulás. Az utolsó bitről kilépő átvitel önmagában nem azonos az előjeles túlcsordulással.

2.6. Előjelkiterjesztés

Ha egy előjeles számot több biten szeretnénk tárolni, a bal szélső előjelbitet kell megismételni.

Például a −5 átalakítása 8 bitesről 16 bitesre:

```
8 biten:  11111011
16 biten: 11111111 11111011
```

Pozitív szám esetén balról 0-kat írunk hozzá:

```
8 biten:  00000101
16 biten: 00000000 00000101
```

Ez az *előjelkiterjesztés* megőrzi a szám eredeti értékét.

3. Törtszámok kettes számrendszerben

A bináris törtek a tízes számrendszer törtrészeihez hasonlóan helyiértékeket használnak, de a tizedesvesszőtől jobbra 2 negatív kitevőjű hatványai állnak.

Bináris helyiérték	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Tízes számrendszerben	1/2	1/4	1/8	1/16
Érték	0,5	0,25	0,125	0,0625

Például:

$$0{,}101_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0{,}5 + 0 + 0{,}125 = 0{,}625_{10}$$

Egy egész- és törtrészt egyaránt tartalmazó szám:

$$1101{,}01_2 = 8 + 4 + 1 + 0{,}25 = 13{,}25_{10}$$

Tízes tört átalakítása binárissá

A törtrészt ismételten megszorozzuk 2-vel. Minden lépésben az eredmény egész része adja a következő bináris számjegyet.

Alakítsuk át a 0,625 értéket:

Művelet	**Egész rész**	**Megmaradó törtrész**
0,625 · 2 = 1,25	1	0,25
0,25 · 2 = 0,5	0	0,5
0,5 · 2 = 1,0	1	0

Az egész részeket felülről lefelé összeolvasva:

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

Nem minden tízes számrendszerbeli tört írható fel véges bináris törtként. A 0,1 például végtelen, ismétlődő bináris tört:

$$0{,}1_{10} = 0{,}00011001100110011..._2$$

Emiatt a számítógép a 0,1 értéknek csak egy közeli közelítését tudja eltárolni.

4. Fixpontos és lebegőpontos ábrázolás

Törtszámok tárolására többféle megoldás használható.

Fixpontos ábrázolásnál előre meghatározzuk, hogy a bitsorozatban hol található a bináris pont. Például egy 8 bites formátumban kijelölhetünk 4 bitet az egészrész és 4 bitet a törtrész számára. A `00110100` bitsorozat ekkor így értelmezhető:

$$0011{,}0100_2 = 3{,}25_{10}$$

A fixpontos számok egyszerűek és kiszámítható pontosságúak, de az ábrázolható tartományuk korlátozott.

Lebegőpontos ábrázolásnál a bináris pont helye nem rögzített. A számot a tudományos számábrázoláshoz hasonló alakban tároljuk:

```
előjel · mantissza · 2kitevő
```

Például:

$$1101{,}01_2 = 1{,}10101_2 \cdot 2^3$$

A bináris pontot három hellyel mozgattuk balra, ezért a kitevő értéke 3.

5. Az IEEE 754 lebegőpontos szabvány

A legtöbb mai számítógép az IEEE 754 szabvány szerint tárolja a lebegőpontos számokat. A bitsorozat három mezőből áll:

- S – előjelbit (sign);
- E – eltolt kitevő (exponent);
- F – törtrészmező (fraction), amelyet gyakran mantisszának is neveznek.

A két leggyakoribb formátum:

Formátum	Teljes méret	Előjel	Kitevő	Törtrész	Kitevő eltolása	Közelítő decimális pontosság
Egyszeres pontosság (float)	32 bit	1 bit	8 bit	23 bit	127	kb. 7 számjegy
Kétszeres pontosság (double)	64 bit	1 bit	11 bit	52 bit	1023	kb. 15–16 számjegy

Egy normál, 32 bites lebegőpontos szám értéke:

$$(-1)^S \cdot 1{,}F_2 \cdot 2^{(E - 127)}$$

ahol:

- S az előjelbit;
- E a kitevőmező előjel nélküli értéke;
- F a törtrészmező;
- 127 a kitevő eltolási értéke, más néven *bias*.

Az előjelbit értelmezése:

```
S = 0: pozitív szám
S = 1: negatív szám
```

A kitevőt eltolva tároljuk, ezért negatív kitevőhöz sincs szükség külön előjelbitre. Például a 3 kitevőt 32 bites formátumban így tároljuk:

$$3 + 127 = 130 = 10000010_2$$

Normál számok esetében a mantissza mindig `1,` kezdetű. A kezdő 1-est nem kell eltárolni, mert annak értéke ismert. Ezt *rejtett egyesnek* nevezzük. Így 23 tárolt törtrészbit valójában 24 bites bináris pontosságot biztosít.

5.1. A 13,25 ábrázolása 32 bites IEEE 754 formátumban

1. Átalakítás kettes számrendszerbe

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

2. Normalizálás

$$1101,01_2 = 1,10101_2 \cdot 2^3$$

3. Az előjelbit meghatározása

A szám pozitív:

$$S = 0$$

4. A kitevő tárolása

$$\text{kitevő} = 3$$
$$\text{eltolt kitevő} = 3 + 127 = 130 = 10000010_2$$

5. A törtrész tárolása

A normalizált alak kezdő 1-esét nem tároljuk. Az utána következő bitek:

$$10101000000000000000000$$

6. A mezők összeállítása

S	E	F
0	10000010	10101000000000000000000

Egybefüggően:

```
01000001010101000000000000000000
```

Hexadecimális alakban:

```
0x41540000
```

A −13,25 ábrázolásánál csak az előjelbit változik 1-re:

```
11000001010101000000000000000000 = 0xC1540000
```

5.2. IEEE 754 bitsorozat visszaalakítása

Határozzuk meg a következő 32 bites lebegőpontos szám értékét:

```
1 | 10000001 | 01100000000000000000000
```

Az előjelbit:

```
S = 1, ezért a szám negatív.
```

A kitevőmező:

```
10000001₂ = 129
valódi kitevő = 129 − 127 = 2
```

A rejtett 1-essel kiegészített mantissza:

```
1,011₂ = 1 + 1/4 + 1/8 = 1,375
```

A szám értéke:

```
(−1) · 1,375 · 2² = −1,375 · 4 = −5,5
```

5.3. Különleges értékek

Az IEEE 754 szabvány bizonyos kitevőmintákat különleges értékek jelölésére tart fenn.

Kitevőmező	Törtrészmező	Jelentés
minden bit 0	minden bit 0	+0 vagy −0
minden bit 0	nem minden bit 0	szubnormális szám
1 és 254 között	tetszőleges	normál szám
minden bit 1	minden bit 0	+végtelen vagy −végtelen
minden bit 1	nem minden bit 0	NaN

Pozitív és negatív nulla

Az IEEE 754 formátumban a nullának két előjeles változata van:

```
+0: S = 0
−0: S = 1
```

A legtöbb műveletben egyenlőnek számítanak, de bizonyos műveleteknél az előjelüknek lehet jelentősége.

Szubnormális számok

A szubnormális számok segítségével a nullához nagyon közeli értékek is ábrázolhatók. Ezeknél nincs rejtett kezdő 1-es, ezért az értékük:

```
(−1)^S · 0,F₂ · 2^(1 − bias)
```

Pontosságuk kisebb, mint a normál számoké, de fokozatos átmenetet biztosítanak a legkisebb normál szám és a nulla között.

Végtelen

Pozitív vagy negatív végtelen keletkezhet például túl nagy eredmény vagy nem nulla érték nullával történő lebegőpontos osztása esetén:

```
1,0 / 0,0 = +végtelen
−1,0 / 0,0 = −végtelen
```

NaN

A NaN jelentése *Not a Number*, vagyis „nem szám”. Matematikailag nem értelmezett lebegőpontos műveletek eredménye lehet, például:

```
0,0 / 0,0
négyzetgyök(−1,0) a valós számok között
végtelen − végtelen
```

A NaN különleges tulajdonsága, hogy még önmagával sem tekinthető egyenlőnek:

```
if (x != x)
{
    /* x értéke NaN */
}
```

A gyakorlatban azonban erre a célra a programozási nyelvek beépített `isnan` függvényét célszerű használni.

6. A lebegőpontos ábrázolás pontatlansága

A lebegőpontos számok véges számú bitből állnak, ezért a valós számoknak csak egy véges részhalmazát tudják pontosan tárolni. A többi értéket a legközelebbi ábrázolható számra kell kerekíteni.

Mivel a 0,1 és a 0,2 nem írható fel véges bináris törtként, a következő összehasonlítás egyes programozási nyelvekben meglepő eredményt adhat:

```
double a = 0.1;
double b = 0.2;

if (a + b == 0.3)
{
    printf("Egyenlo");
}
else
{
    printf("Nem egyenlo");
}
```

Az eredmény rendszerint `Nem egyenlo`, mert a tárolt közelítő értékek összege nem pontosan egyezik meg a 0,3 tárolt közelítésével.

Lebegőpontos értékeket ezért általában egy kis hibahatár, úgynevezett *epsilon* segítségével hasonlítunk össze:

```
#include <math.h>

double epsilon = 1e-9;

if (fabs((a + b) - 0.3) < epsilon)
{
    printf("A ket ertek a megadott pontossagon belul egyenlo");
}
```

Az epsilon megfelelő értéke a számok nagyságrendjétől és az alkalmazás pontossági követelményeitől függ; nem minden feladathoz jó ugyanaz az állandó.

6.1. Pontosság és nagyságrend

A lebegőpontos számábrázolás relatív pontosságú. Minél nagyobb számokat tárolunk, annál nagyobb távolság lehet két egymást követő ábrázolható érték között.

A 32 bites `float` 24 bites bináris pontossága miatt minden egész számot pontosan képes ábrázolni $2^{24} = 16\ 777\ 216$ értékig. E fölött már nem minden egymást követő egész szám tárolható.

Például 32 bites float esetén:

```
16 777 216 még pontosan ábrázolható.
16 777 217 már nem ábrázolható pontosan.
```

Ezért lebegőpontos típust nem célszerű például nagy azonosítók vagy pénzösszegek pontos tárolására használni.

6.2. Túlcsordulás és alulcsordulás

Túlcsordulás akkor történik, ha az eredmény abszolút értéke nagyobb a formátumban tárolható legnagyobb véges számnál. Az eredmény ilyenkor gyakran pozitív vagy negatív végtelen lesz.

Alulcsordulás akkor történik, ha egy nem nulla eredmény abszolút értéke túl közel kerül a nullához. Az eredmény először szubnormális számmá, végül nullává kerekedhet.

6.3. Kerekítési hibák felhalmozódása

Egyetlen lebegőpontos művelet hibája rendszerint nagyon kicsi, de sok egymás utáni művelet során a hibák felhalmozódhatnak. A műveletek sorrendje is befolyásolhatja az eredményt.

Például nagyon nagy és nagyon kicsi szám összeadásakor a kis szám hatása elveszhet:

```
nagy_szam + kis_szam = nagy_szam
```

Ez nem matematikai azonosság, hanem a véges pontosság következménye.

7. Egész és lebegőpontos számábrázolás összehasonlítása

Tulajdonság	Kettes komplemens egész szám	IEEE 754 lebegőpontos szám
Tárolható értékek	egészek	törtrészes és nagyon kis vagy nagy számok
Pontosság	a tartományon belül pontos	gyakran közelítő
Előjel kezelése	a legmagasabb bit helyiértékének része	külön előjelbit
Túlcsordulás	a bitsorozat hibás előjelű értékké fordulhat vagy a nyelv hibát jelezhet	gyakran végtelen keletkezik
Különleges értékek	általában nincsenek	±0, ±végtelen, NaN, szubnormális számok
Jellemző alkalmazás	számlálók, indexek, darabszámok	mérések, geometriai és tudományos számítások

8. Összefoglalás

- Az előjel nélküli n bites egész számok tartománya $0 \ldots 2^n - 1$.
- Az előjeles egész számokat a modern számítógépek jellemzően kettes komplemens alakban tárolják.
- Egy negatív kettes komplemens számot a pozitív alak bitjeinek megfordításával, majd 1 hozzáadásával kapunk meg.

- Az n bites kettes komplemens számok tartománya -2^{n-1} … $2^{n-1} - 1$.
- A bináris törtrész helyiértékei 2^{-1}, 2^{-2}, 2^{-3}, … értékek.
- Az IEEE 754 lebegőpontos szám előjelből, eltolt kitevőből és törtrészből áll.
- Sok tízes tört, például a 0,1, binárisan csak közelítőleg tárolható.
- Lebegőpontos számokat általában nem közvetlen egyenlőségvizsgálattal, hanem megfelelő hibahatár figyelembevételével hasonlítunk össze.
- Pénzügyi és más, pontos tizedes számítást igénylő feladatokhoz célszerű decimális vagy skálázott egész számábrázolást használni.

9. Ellenőrző kérdések és feladatok

1. Miért nem határozható meg egy bitsorozat értéke az adattípus ismerete nélkül?
2. Mi a 45 előjel nélküli 8 bites bináris alakja?
3. Ábrázolja a −18 számot 8 bites kettes komplemens formában!
4. Milyen értéket jelent a `11101100` bitsorozat 8 bites kettes komplemens számként?
5. Mi a 8 bites előjeles egészek legkisebb és legnagyobb értéke?
6. Végezze el 8 biten a `12 + (−7)` összeadást!
7. Miért okoz túlcsordulást 8 biten a `100 + 40` összeadás?
8. Alakítsa át a $101{,}101_2$ számot tízes számrendszerbe!
9. Alakítsa át a $10{,}625_{10}$ számot kettes számrendszerbe!
10. Milyen három fő mezőből áll egy IEEE 754 lebegőpontos szám?
11. Miért nem tárolható pontosan a 0,1 a szokásos bináris lebegőpontos formátumokban?
12. Mit jelent a NaN érték?

10. A feladatok rövid megoldása

1. A bitsorozat jelentése az ábrázolási módtól és az adattípustól függ.
2. $45_{10} = 00101101_2$.
3. `+18 = 00010010`, bitfordítás után `11101101`, majd 1 hozzáadásával `−18 = 11101110`.
4. $11101100_2 = -20_{10}$.
5. A tartomány −128-tól 127-ig tart.
6. `00001100 + 11111001 = 00000101`, tehát az eredmény 5.
7. Az eredmény 140 lenne, amely nagyobb a legnagyobb 8 bites előjeles értéknél, vagyis 127-nél.
8. $101{,}101_2 = 5{,}625_{10}$.
9. $10{,}625_{10} = 1010{,}101_2$.
10. Előjelbitből, kitevőmezőből és törtrészmezőből.
11. Mert a 0,1 bináris alakja végtelen, ismétlődő tört, miközben a tárolásra véges számú bit áll rendelkezésre.
12. A NaN matematikailag nem értelmezhető lebegőpontos művelet eredményét jelöli.