Számábrázolások

Bérci Norbert

2011/2012 őszi félév
v1.0

Tartalomjegyzék

1. Számrendszerek **1**
1.1. A számrendszer alapja és a számjegyek . . . 1
1.2. Alaki- és helyiérték . . . 2
1.3. Egész számok leírása . . . 2
1.4. Nem egész számok leírása . . . 2
1.5. Átváltás számrendszerek között . . . 3
1.6. Feladatok . . . 3
1.7. Számrendszerek pontossága . . . 3

2. Mértékegységek **4**

3. Gépi számábrázolás **4**
3.1. Előjel nélküli egész számok ábrázolása . . . 4
3.2. Előjeles egész számok ábrázolása . . . 5
3.3. Egész számok ábrázolási határai . . . 5
3.4. Túlcsordulás . . . 6
3.5. Lebegőpontos számábrázolás . . . 6
3.6. Lebegőpontos számábrázolás határai és pontossága . . . 7
3.7. Végtelenek és a NaN . . . 7

1. Számrendszerek

A számrendszerek egy szám (mint matematikai fogalom) írott formában történő megjelenítésére alkalmas módszerek. Ebben a részben a helyiértéken (pozíción) alapuló számrendszereket tárgyaljuk. Léteznek nem pozíción alapuló számrendszerek is, ilyenek például a sorrendiségen alapuló római számok, de ezekkel a továbbiakban nem foglalkozunk.

1.1. A számrendszer alapja és a számjegyek

A helyiértéken alapuló számrendszerek két legfontosabb tulajdonsága *a számrendszer alapja* és *az egyes pozíciókba írható számjegyek*. Ez a két tulajdonság nem független egymástól: A számrendszer alapja meghatározza az egyes pozíciókba írható számjegyek maximumát is: ha a számrendszer A alapú, akkor a számjegyek a következők: $0, 1, \ldots, A-1$.

1.1. példa. A nyolcas számrendszerben a $0, 1, 2, 3, 4, 5, 6, 7$ számjegyek közül választhatunk, míg a kettesben a $0, 1$ közül.

Tíznél nagyobb alapú számrendszerek esetében a számjegyek halmazát 9 után az ABC betűivel egészítjük ki. A kis és nagybetűk között általában nem teszünk különbséget, bár egyes nagy alapú számrendszereknél erre mégis szükség lehet.

1.2. példa. A tizenhatos számrendszerben használható "számjegyek": 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f (vagy 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Ha az a szövegkörnyezetből nem egyértelmű, a számrendszer alapját szögletes zárójelben a jobb alsó indexbe téve jelölhetjük. Például: $5221_{[10]}$, $726_{[8]}$ vagy $8af0_{[16]}$.

1.2. Alaki- és helyiérték

Egy adott számrendszerben leírt szám esetében egy számjegy értéke egyenlő a számjegy *alaki értékének* és *helyiértékének* szorzatával. A számjegy alaki értéke a számjegyhez tartozó érték, a helyiérték pedig a számrendszer alapjának a pozíció szerinti hatványa. A $0, 1, \ldots, 9$ esetében az alaki érték egyértelmű, a betűkkel kiegészített esetben pedig: a=10, b=11, c=12, d=13 stb.

1.3. példa. A tízes számrendszerben felírt 32 szám esetében a 3 helyiértéke $10^1 = 10$, mivel az jobbról a második pozíción szerepel (és a helyiértékeket a nulladik hatványtól indítjuk), így értéke: $3 \cdot 10^1 = 3 \cdot 10 = 30$.

1.4. példa. A tízes számrendszerben felírt 32 szám esetében a 2 helyiértéke $10^0 = 1$, mivel az jobbról az első pozíción szerepel (és a helyiértékeket a nulladik hatványtól indítjuk), így értéke: $2 \cdot 10^0 = 2 \cdot 1 = 2$.

A jól ismert tízes alapú *decimális* számrendszeren kívül az informatikában a leggyakrabban használtak a következők: a kettes alapú *bináris*, a nyolcas alapú *oktális*, és a tizenhatos alapú *hexadecimális*. Oktális számrendszer jelölésére használatos a kezdő 0 szerepeltetése, hexadecimális számok esetén a 0x, 0X prefixek vagy a h postfix. Például: 065 (oktális), 0x243 (hexadecimális), 0X331 (hexadecimális), 22h (hexadecimális). Ha sem a szám előtt, sem utána, sem az indexében nincs jelölve, akkor decimális számrendszerben értelmezzük a számot.

1.3. Egész számok leírása

Általánosan tehát az A alapú számrendszerben felírt, $a_n a_{n-1} \ldots a_1 a_0$ számjegyekből álló szám értéke:

$$(a_n \cdot A^n) + (a_{n-1} \cdot A^{n-1}) + \cdots + (a_1 \cdot A^1) + (a_0 \cdot A^0)$$

ami nem más, mint a leírt számjegyek az előzőekben megismert módon kiszámolt értékeinek (alaki érték szorozva a helyiértékkel) összege.

1.5. példa. Triviális példa: $405_{[10]} = 4 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0 = 400 + 5$

1.6. példa. $405_{[8]} = 4 \cdot 8^2 + 0 \cdot 8^1 + 5 \cdot 8^0 = 256 + 5 = 261_{[10]}$

1.7. példa. $1001101_{[2]} = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 8 + 4 + 1 = 77_{[10]}$

1.8. példa. $0\text{xA3} = 10 \cdot 16^1 + 3 \cdot 16^0 = 10 \cdot 16 + 3 \cdot 1 = 163_{[10]}$

1.4. Nem egész számok leírása

Az előző részben megismert felírási módszert kiterjeszthetjük úgy, hogy a helyiértékek megadásánál nem állunk meg a nulladik hatványnál, hanem folytatjuk azt a negatív hatványokra is, így lehetőségünk adódik nem egész számok leírására. Általánosan tehát az A alapú számrendszerben felírt, $a_n a_{n-1} \ldots a_1 a_0 a_{-1} \ldots a_{-k}$ számjegyekből álló szám értéke:

$$a_n \cdot A^n + a_{n-1} \cdot A^{n-1} + \cdots + a_1 \cdot A^1 + a_0 \cdot A^0 + a_{-1} \cdot A^{-1} + \cdots + a_{-k} \cdot A^{-k}$$

Annak érdekében, hogy a mindkét végén (egész illetve törtrész) tetszőlegesen bővíthető felírás egyértelmű legyen, ennek a két résznek a határát jelöljük tizedesponttal (illetve a magyar helyesírás szerint tizedesvesszővel).[1] Mi a továbbiakban a tizedespontos jelölést fogjuk alkalmazni.

1.9. példa. Triviális példa: $405.23_{[10]} = 4 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0 + 2 \cdot 10^{-1} + 3 \cdot 10^{-3} = 4 \cdot 100 + 5 \cdot 1 + 2 \cdot \frac{1}{10} + 3 \cdot \frac{1}{100}$

[1] Ha nagyon pontosak akarunk lenni, akkor tizedespontról csak a tízes számrendszer használata esetén beszélhetnénk, bináris esetben inkább bináris pontról van szó (és hasonlóan oktális, hexadecimális stb. esetben).

1.10. példa. $405.23_{[8]} = 4 \cdot 8^2 + 0 \cdot 8^1 + 5 \cdot 8^0 + 2 \cdot 8^{-1} + 3 \cdot 8^{-2} = 4 \cdot 64 + 5 \cdot 1 + 2 \cdot \frac{1}{8} + 3 \cdot \frac{1}{8^2} = 256 + 5 + \frac{2}{8} + \frac{3}{64} = 261.296875_{[10]}$

1.11. példa. $1001101.01_{[2]} = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 64 + 8 + 4 + 1 + \frac{1}{4} = 77.25_{[10]}$

1.5. Átváltás számrendszerek között

Az adott számrendszerből tízes számrendszerbe váltást az előzőekben hallgatólagosan már bemutattuk. A fordított átváltásra nem térünk ki (a módszer könnyen kitalálható).

Az átváltás nagymértékben egyszerűsödik, ha a binárisból oktális vagy hexadecimális számrendszerbe kell átváltani: egyszerűen hármasával (oktális esetben) vagy négyesével (hexadecimális esetben) kell a bináris számjegyeket csoportosítani, és az így képzett csoportokat átváltani. Az átváltás fordított irányban is hasonlóan egyszerű: az egyes oktális vagy hexadecimális számjegyeket kell átváltani és az így kapott hármas illetve négyes bináris csoportokat egymás után írni. Oktálisból hexadecimálisba vagy decimálisból hexadecimálisba illetve fordítva a bináris számrendszert közbeiktatva egyszerűen átválthatunk.

1.12. példa. $110111101_{[2]} = 110\ 111\ 101_{[2]} = 675_{[8]}$

1.13. példa. $10111101_{[2]} = 1011\ 1101_{[2]} =$0xbd

1.14. példa. $\mathrm{c6d}_{[16]} = 1100\ 0110\ 1101_{[2]} = 110\ 001\ 101\ 101_{[2]} = 6155_{[8]}$

1.6. Feladatok

1.1. feladat. $1010111001_{[2]} = ?_{[8]} = ?_{[16]}$

1.2. feladat. $54_{[8]} = ?_{[16]}$

1.3. feladat. $962_{[10]} = ?_{[8]} = ?_{[16]}$

1.4. feladat. $\mathrm{9a2d}_{[16]} = ?_{[2]} = ?_{[8]} = ?_{[16]}$

1.5. feladat. Adjunk algoritmust (módszert) decimálisból a) oktális-, b) hexadecimális számrendszerbe történő közvetlen (tehát nem a bináris számrendszer közbeiktatásával történő) átváltásra!

1.6. feladat. Minden racionális szám (tört) leírható bármilyen alapú számrendszerben véges számjegy felhasználásával?

1.7. feladat. A http://www.digitconvert.com/ oldalon kipróbálhatók, ellenőrizhetők az átváltások.

1.7. Számrendszerek pontossága

Fontos kiemelni, hogy nem egész számok felírása esetén nem biztos, hogy az átváltás hibamentesen lehetséges (nem írható le véges számjeggyel)! Máshogyan megfogalmazva, a nem egész számok ábrázolásának pontossága függ a számrendszer alapjától: például az $\frac{1}{3}$ tízes számrendszerben nem írható fel véges számjeggyel, ugyanakkor hármas számrendszerben pontosan felírható: $\frac{1}{3} = 0.1_{[3]} = 1.33333\ldots_{[10]}$

1.8. feladat. Adjunk meg néhány példát arra, amikor az egyik számrendszerben véges számjeggyel felírható szám a másik számrendszerben nem írható fel véges számjeggyel!

1.9. feladat. Kiválasztható olyan alapú számrendszer, amiben minden racionális szám pontosan ábrázolható? Indokoljuk meg!

2. Mértékegységek

Az informatikában használatos legkisebb egység a bit (sok esetben b-vel rövidítik, de a legfrissebb szabvány[2] a rövidítés nélküli formát ajánlja). Értéke 0 vagy 1 lehet. Használhatjuk tárolókapacitás vagy információ jelölésére. Az utóbbi egy felsőbb éves tárgy, az *Információ és kódelmélet* témája, mi itt csak a tárolási vonatkozásával foglalkozunk.

A bájt (byte) az informatika másik legfontosabb egysége, jele: B. Mi az általánosan elfogadott, a gyakorlatban majdnem kizárólagosan használt 1 B = 8 bit átváltást használjuk, bár egyes (egzotikus) architektúrák esetében ennél több vagy kevesebb bit is alkothat egy bájtot.

Az SI mértékegységrendszerben használatos k (kilo), M (mega), G (giga), T (tera), P (peta) stb. prefixek mellett a bit és a bájt esetében használatosak a Ki (kibi), Mi (mebi), Gi (gibi), Ti (tebi), Pi (pebi) stb. *bináris prefixek* is (lásd az 1. ábrán). Fontos kiemelni, hogy az egyre nagyobb prefixek esetében egyre nagyobb a különbség az SI és az informatikai/bináris prefixek között. Például a G (1000^3) és Gi (1024^3) között a különbség kb. 7%, a T (1000^4) és Ti (1024^4) között már kb. 10%.[3]

A kapcsolat a prefixek és a számrendszerek között ott fedezhető fel, hogy a használt prefixek mindig a számrendszer alapjának valamely hatványai. Az SI esetben ez a tíz harmadik hatványa (illetve ennek további hatványai), de ugyanez igaz a bináris prefixekre is, amikor is ez a kettő tizedik hatványa (illetve ennek további hatványai).

prefix	szorzó	prefix	szorzó
k (kilo)	1000	Ki (kibi)	1024
M (mega)	1000^2	Mi (mebi)	1024^2
G (giga)	1000^3	Gi (gibi)	1024^3
T (tera)	1000^4	Ti (tebi)	1024^4
P (peta)	1000^5	Pi (pebi)	1024^5

1. ábra. SI és bináris prefixek

3. Gépi számábrázolás

A gépi számábrázolás a számok (számító)gépek memóriájában történő tárolását jelenti.

3.1. Előjel nélküli egész számok ábrázolása

Az előjel nélküli (nem negatív) egész számok ábrázolása megegyezik a bináris számrendszernél megismert leírással: $86_{[10]} = 01010110_{[2]}$, azaz egy nem negatív egész számot a kettes számrendszerbe átváltott formájában tárolunk. A tömörebb írásmód miatt ugyanakkor ezt legtöbbször nem bináris, hanem hexadecimális formában írjuk le. (Ne feledjük, hogy a bináris - hexadecimális átváltás nem más, mint négy bitesével csoportosítás, ahogy azt az előzőekben láthattuk.)

Mivel a gyakorlatban általában csak egész byte méretű ábrázolásokat használunk, az előjel nélküli egészek is legtöbbször 1, 2, 4, 8, 16 byte (8, 16, 32, 64, 128 bit) hosszúak lehetnek. Így is hívjuk ezeket: 8 bites előjel nélküli egész, 16 bites előjel nélküli egész stb.

3.1. példa. A $46_{[10]}$ számot a memóriában a következőképpen tároljuk 1 bájton: 00101110.

3.1. feladat. Az összeadás művelet hogyan végezhető el az előjel nélküli egész számok bináris tárolása esetén? Adjunk erre módszert (algoritmust)!

[2] ISO/IEC 80000, Part 13 - Information science and technology

[3] Különösen fontos ez a háttértárak esetében, ahol a gyártók inkább az SI prefixeket használják, mert így egy 1000000000000 B méretű lemezegység esetében 1 TB-ot tüntethetnek fel, míg ugyanez a bináris prefixekkel csupán 0.9 TiB

3.2. Előjeles egész számok ábrázolása

Első gondolatunk az lehet, hogy egy előjelet jelentő bit hozzáadásával (ami például 0, ha pozitív az előjel és 1, ha negatív az előjel) egyszerűen meg lehet oldani a problémát. Sajnos azonban ez a megoldás sok szempontból nem megfelelő: a legkézenfekvőbb probléma, hogy ezzel a módszerrel lehetséges a $+0$ és a -0 ábrázolása is, ami zavarhoz vezet (például a „nulla-e" vizsgálatot így két különböző értékre kell megtenni), továbbá az ilyen módon felírt számokkal végzett műveletek bonyolultabbak, mint amennyire az feltétlenül szükséges lenne.

3.2. feladat. A 3.1. feladatban kitalált összeadás művelet elvégezhető-e *módosítás nélkül* az előjelbites számábrázolási módszer használatával?

Sokkal jobb eredményre vezet a *kettes komplemens* ábrázolás: ahelyett, hogy egy előjelbittel jelölnénk az előjelet, a negatív számokat úgy ábrázoljuk, hogy hozzáadjuk őket egy nagy pozitív számhoz, és az eredményt ábrázoljuk, mint egy előjel nélküli egész számot. Ahhoz, hogy az eredmény biztosan pozitív legyen (hogy előjel nélküli egészként felírhassuk) de ugyanakkor ne is legyen túl nagy (hogy minél kevesebb biten felírható legyen az eredmény), a következő módszert alkalmazzuk: ha N biten akarjuk tárolni a kettes komplemens számot, akkor a negatív számhoz hozzáadandó nagy pozitív szám legyen a következő: 2^N.

3.2. példa. A -2 kettes komplemens ábrázolása 8 biten: $2^8 + (-2) = 256 - 2 = 254$, azaz: 11111110.

3.3. példa. A -17 kettes komplemens ábrázolása 8 biten: $2^8 + (-17) = 256 - 17 = 239$, azaz: 11101111.

Fontos tudnivalók:

- Kettes komplemens ábrázolásban is lehetséges nem negatív számok ábrázolása, aminek módja megegyezik az előjel nélküli egészek tárolási módjával. (Azaz ebben az esteben nem kell a számot hozzáadni a 2^N-hez.)
- A kettes komplemens ábrázolásban már csak egyetlen ábrázolási módja van a nullának.

3.3. feladat. Adjuk meg a 0 kettes komplemens ábrázolását 8, 16, 32, 64 biten!

3.4. feladat. Adjuk meg a -1 kettes komplemens ábrázolását 8, 16, 32, 64 biten!

3.5. feladat. Adjuk meg az 1 kettes komplemens ábrázolását 8 biten!

3.6. feladat. A 3.1. feladatban kitalált összeadás művelet elvégezhető-e *módosítás nélkül* a kettes komplemens számábrázolási módszer használatával? Adjuk össze az előző két feladatokban kiszámolt, 8 bites -1 és 1 értéket, és ellenőrizzük, hogy nullát kaptunk-e!

3.3. Egész számok ábrázolási határai

Ha előjel nélküli bináris módon ábrázoljuk a nem negatív számokat és ehhez N bit áll rendelkezésre, akkor a tárolható legkisebb érték: 0, a tárolható legnagyobb érték: $2^N - 1$.

3.4. példa. Ha 8 bites előjel nélküli egész ábrázolást használunk, akkor a legkisebb ábrázolható szám a 00000000 (értéke 0), a legnagyobb ábrázolható szám az 11111111 (értéke $2^8 - 1 = 255$).

3.7. feladat. Mennyi a legnagyobb tárolható érték 8, 16, 32, 64 bites előjel nélküli egész esetében?

3.8. feladat. Összesen hány különböző érték tárolható 8, 16, 32, 64 biten, előjel nélküli egész számábrázolás esetében?

Ha kettes komplemens módon ábrázolunk egy egész számot és ehhez N bit áll rendelkezésre, akkor a tárolható legkisebb érték: -2^{N-1}, a tárolható legnagyobb érték: $2^{N-1} - 1$.

3.5. példa. Ha 8 bites kettes komplemens ábrázolást használunk, akkor a legkisebb ábrázolható szám az 10000000 (értéke -128), a legnagyobb ábrázolható szám a 01111111 (értéke 127).

3.9. feladat. Kettes komplemens ábrázolás esetén miért nem ugyanannyi szám tárolható a pozitív és a negatív tartományban? (Azaz miért nem -127 és 127 illetve -128 és 128 a két határ?)

3.10. feladat. Mennyi az értéke a kettes komplemens ábrázolással, 8 biten tárolt 11111111 illetve a 00000000 számoknak?

3.11. feladat. Eldönthető egyszerűen (ránézésre) egy kettes komplemens módon ábrázolt számról, hogy az negatív vagy pozitív?

3.12. feladat. Összesen hány különböző érték tárolható 8, 16, 32, 64 biten, kettes komplemens számábrázolás esetében?

3.13. feladat. Mi a kapcsolat a 3.8. feladat és a 3.12. feladatban kapott eredmények között?

3.4. Túlcsordulás

Az egész számok véges biten történő ábrázolása miatt mindig van legkisebb és legnagyobb ábrázolható szám. Amikor műveletet végzünk, elképzelhető, hogy a művelet eredménye már nem ábrázolható az operandusokkal megegyező méretben. Ezt a jelenséget túlcsordulásnak nevezzük.

3.6. példa. Ha 8 bites előjel nélküli egészekkel dolgozunk, a 156+172=328 összeget már nem tudjuk 8 biten tárolni (mert a legnagyobb tárolható érték a 255).

3.7. példa. Ha 8 bites előjeles egészekkel dolgozunk, a -84+(-79)=-163 összeget már nem tudjuk 8 biten tárolni (mert a legkisebb tárolható érték a -127).

Túlcsordulás esetén – megvalósítástól függően – lehetséges

- *levágás*: a túlcsordult eredmény még ábrázolható részét tároljuk (a számláló kvázi körbefordul, mint például egy gázóránál vagy kilométeróránál, aminél fix helyiértéken történik a mérés),
- *szaturáció*: a túlcsordult eredmény helyett a legnagyobb illetve legkisebb ábrázolható értéket tároljuk.

3.8. példa. Levágás: ha 8 bites előjel nélküli egészekkel dolgozunk, a $156_{[10]} = 10011100_{[2]}$ és a $172_{[10]} = 10101100_{[2]}$ valódi összege ($328_{[10]} = 101001000_{[2]}$) helyett annak a 8 utolsó bitjét tároljuk: 01001000.

3.9. példa. Szaturáció: ha 8 bites előjel nélküli egészekkel dolgozunk, a $156_{[10]} = 10011100_{[2]}$ és a $172_{[10]} = 10101100_{[2]}$ valódi összege ($328_{[10]} = 101001000_{[2]}$) helyett az ábrázolható legnagyobb számot tároljuk: 11111111.

3.5. Lebegőpontos számábrázolás

Nem egész számok gépi ábrázolására a lebegőpontos ábrázolást használjuk: a számot először átalakítjuk normalizált alakba, és az így kapott alak különböző részeit külön-külön tároljuk.

Egy szám normalizált alakján olyan szorzatra bontását értjük, ahol a második tag a számrendszer alapjának valamely hatványa (amit a szám nagyságrendjének is nevezünk), az első tag értéke pedig annyi, hogy a második taggal megszorozva az eredeti számot kapjuk, továbbá az első tag legalább 1, de a számrendszer alapjánál kisebb értékű.

3.10. példa. Tízes számrendszerben a 382 normalizált alakja: $3.82 \cdot 10^2$, a 3.875 normalizált alakja $3.875 \cdot 10^0$, a 0.00000651 normalizált alakja $6.51 \cdot 10^{-6}$.

Az előzőekben definiált első tagot a szám *mantisszájának*, a hatványkitevőt (a nagyságrendet) a szám *karakterisztikájának* nevezzük. Negatív számok tárolásához szükség van még az előjelre (ami 1 bit méretű, értéke 0, ha pozitív a szám és 1, ha negatív a szám). Egy szám ábrázolásához tehát ezeket az értékeket kell eltárolni.

Kettes számrendszerben ábrázolva a számot, a normalizált alak tovább egyszerűsödik, hiszen a mantissza tizedespontja előtt mindig 1 áll, amit így nem kell eltárolni, és ezt a megtakarított bitet a mantissza pontosabb tárolására lehet fordítani.

A karakterisztika is lehet pozitív vagy negatív, de a karakterisztika esetében nem az egész számoknál megismert előjeles (kettes komplemens) tárolást használjuk, hanem az *eltolt* tárolást: a tárolandó karakterisztikához hozzáadunk egy fix értéket, és az így kapott eredményt tároljuk el.

3.11. példa. A $382_{[10]} = 101111110_{[2]}$ normalizált alakja: $1.01111110_{[2]} \cdot 2^{8_{[10]}} = 1.01111110_{[2]} \cdot 2^{1000_{[2]}}$, azaz tárolandó a 0 előjelbit, a 01111110 mantissza és az 1000 karakterisztika (a megfelelő eltolással).

3.12. példa. A $-3.375_{[10]} = -11.011_{[2]}$ normalizált alakja $-1.1011_{[2]} \cdot 2^1$, azaz tárolandó az 1 előjelbit, a 1011 mantissza és az 1 karakterisztika (a megfelelő eltolással).

3.13. példa. A $-3.375_{[10]} = -11.011_{[2]}$ normalizált alakja $-1.1011_{[2]} \cdot 2^1$, azaz tárolandó az 1 előjelbit, a 1011 mantissza és az 1 karakterisztika (a megfelelő eltolással).

3.6. Lebegőpontos számábrázolás határai és pontossága

Az egész számok tárolásakor az ábrázolásnak csak a két határát kellett megadni, azokon belül minden egész szám tárolására lehetőség volt. A lebegőpontos számábrázolás esetében ez nincs így, véges hosszon nyilván nem ábrázolható minden szám (még adott intervallum rögzítése esetén sem). Ezek miatt az ábrázolási határok mellett a pontosság is jellemez egy-egy konkrét lebegőpontos számábrázolást, ami megadja, hogy egy adott szám tárolása esetén a tárolt szám és az eredeti szám értéke milyen távol áll egymástól. A lebegőpontos számok normalizált alakú tárolásából következik, hogy a pontosságot a mantissza tárolási mérete határozza meg, az ábrázolási határok pedig a elsődlegesen a karakterisztikából következnek.

A lebegőpontos számábrázolás határainak és pontosságának megadásához az IEEE 754 = IEC 599 = ISO/IEC 60559 szabványban definiált konkrét bináris lebegőpontos ábrázolásokat vizsgáljuk, amit a 2. ábrán foglaltunk össze.

elnevezés	mantissza méret	karakterisztika méret	karakterisztika eltolás
binary16	10	5	15
binary32	23	8	127
binary64	52	11	1023
binary128	112	15	16383

2. ábra. IEEE 754 = IEC 599 = ISO/IEC 60559 bináris lebegőpontos típusok jellemzői

3.14. feladat. Adjuk meg a legnagyobb binary16-ban ábrázolható számot!

3.7. Végtelenek és a NaN

Fontos kiemelni, hogy a szabványos lebegőpontos számábrázolások a valós számokon kívül képesek tárolni a ∞-t és $-\infty$-t, továbbá a speciális NaN (Not a Number) értéket. Ez utóbbit kapjuk eredményül (többek között) akkor, ha nullát nullával osztunk vagy ha negatív számból vonunk négyzetgyököt.