Lebegőpontos számok ábrázolása (jegyzet)

Bérci Norbert

2015. szeptember 17-i óra anyaga

Tartalomjegyzék

1. A lebegőpontos számábrázolás

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

1.1. Normalizált alak

Egy szám normalizált alakján olyan szorzatra bontását értjük (lásd 1. ábra), 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. További feltétel, hogy az első tag egyetlen nem nulla számjegyet tartalmazzon a tizedespont előtt, ami garantálja, hogy a normalizált alakban történő felírás egyértelmű legyen.

1.1.1. példa. Tízes számrendszerben a 380 normalizált alakja: $3.8 \cdot 10^2$, a 3.875 normalizált alakja $3.875 \cdot 10^0$, a 0.00000651 normalizált alakja $6.51 \cdot 10^{-6}$, a -53.75 normalizált alakja: $-5.375 \cdot 10^1$.

Az előzőekben definiált első tagot a szám *mantisszájának* [mantissa, significand, coefficient], a hatványkitevőt (a nagyságrendet) a szám *karakterisztikájának* vagy *exponens*ének [exponent] nevezzük. Negatív számok tárolásához szükség van még az előjelre is (lásd 1. ábra).

A lebegőpontos elnevezés abból adódik, hogy az ábrázolható számok nem fix helyiértékű tizedesjegyekkel kerülnek tárolásra[1], hanem az exponens alapján a mantissza tizedespontja változik („lebeg").

1. ábra. A -37.75 normalizált alakja tízes számrendszerben és ennek elemei (bekarikázva).

1.1.2. feladat. Adjunk arra példát, hogy az „első tag egyetlen nem nulla számjegyet tartalmazzon a tizedespont előtt" feltétel hiányában egy számot többféleképpen is fel lehet írni normalizált alakban!

1.1.3. feladat. Adjuk meg a nulla normalizált alakját!

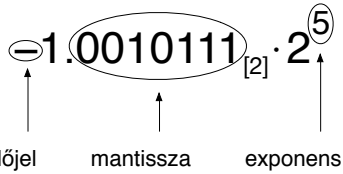
1.2. Bináris normalizált alak

1.2.1. A mantissza

Kettes számrendszerben ábrázolva a számot, a normalizált alak tovább egyszerűsödik, hiszen a mantissza tizedespontja előtt mindig 1 áll („az első tag egyetlen nem nulla számjegyet tartalmazzon a tizedespont előtt" feltétel miatt), amit így nem kell eltárolni (lásd 2. ábra), és ezt a megtakarított bitet a mantissza pontosabb tárolására lehet fordítani.

Fontos, hogy a lebegőpontos szám értelmezésekor ezt az el nem tárolt számjegyet is figyelembe vegyük, továbbá, hogy az első (nem tárolt) egyes után már fontosak az azt követő nullák, azaz például a 0010111 mantissza helyett nem tárolhatjuk el az 10111 értéket!

[1] ezt a módot fixpontos ábrázolásnak nevezzük



2. ábra. A -37.75 normalizált alakja kettes számrendszerben és ennek *tárolandó* elemei (bekarikázva).

1.2.2. Az exponens

A normalizálásból adódóan az exponens is lehet pozitív vagy negatív. Ennek tárolásához az eltolt tárolási módszert használjuk.

1.2.3. Az előjel

Az előjelet 1 biten tároljuk; ha értéke 1: a szám negatív, ha 0: a szám pozitív.

1.2.1. példa. A $380_{[10]} = 101111100_{[2]}$ normalizált alakja $1.011111_{[2]} \cdot 2^8$, azaz tárolandó a 0 előjelbit, a 011111 mantissza és a 8 karakterisztika (a megfelelő eltolással).

1.2.2. 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).

1.3. A lebegőpontos szám elemeinek tárolási mérete

Az előjelet mindig egy biten, a mantisszát és az exponenst megadott számú biten tároljuk. Ha az előzőekben kiszámolt mantissza vagy exponens mérete nem egyezik meg a tárolási mérettel, akkor az exponenst balról, a mantisszát jobbról egészíthetjük ki nullákkal (ha szükséges)!

1.3.1. példa. A mantissza 10 biten, az exponens 5 biten történő excess-15 ábrázolása esetén a tízezer ábrázolása: $10000 = 10011100010000_{[2]} = 1.001110001_{[2]} \cdot 2^{13}$, azaz tárolandó a 0 előjel, a 0011100010 mantissza (figyelem, kiegészítettük 10 bites hosszra nullákkal jobbról!) és a 13 exponens, utóbbi az excess-15 ábrázolás miatt 11100 formában.

1.3.2. példa. A mantissza 10 biten, az exponens 5 biten történő excess-15 ábrázolása esetén a -0.078125 ábrázolása: $-0.078125 = -0.000101_{[2]} = -1.01_{[2]} \cdot 2^{-4}$, azaz tárolandó az 1 előjel, a 0100000000 mantissza (figyelem, kiegészítettük 10 bites hosszra nullákkal jobbról!) és a -4 exponens, utóbbi az excess-15 ábrázolás miatt 01011 formában (figyelem, kiegészítettük 5 bites hosszra egy nullával balról!).

1.3.3. feladat. A mantisszát miért jobbról, az exponenst miért balról egészíthetjük csak ki nullákkal?

1.4. Lebegőpontos számábrázolás határai és pontossága

Az előjeles vagy előjel nélküli egész és a lebegőpontos számábrázolás esetében is csak fix értékek tárolhatók, de míg ezek az egészek esetében pontosan megegyeznek a tárolni kívánt egész számokkal, a lebegőpontos számábrázolás esetében ez nincs így, mivel bárhogy válasszuk is meg a számábrázolás határait, az ezek között lévő végtelen sok valós szám nyilván nem ábrázolható véges helyen. Ezt úgy is értelmezhetjük, hogy a lebegőpontos tárolás során kényszerű kerekítés történik. Mindezek miatt az ábrázolási határok mellett a pontosság is jellemez egy-egy konkrét lebegőpontos számábrázolást, ami megadja, hogy egy adott szám tárolása esetén a tárolni kívánt és a tárolt szám értéke legfeljebb milyen távol lehet egymástól. A lebegőpontos számok normalizált alakú tárolásából következik, hogy a pontosságot a mantissza tárolási mérete határozza meg, az ábrázolási határok pedig elsődlegesen a karakterisztika ábrázolási méretéből adódnak. Fontos azt is kiemelni, hogy a pontosság az ábrázolási tartományban abszolút értelemben nem egyenletes, azaz függ az ábrázolni kívánt számtól (lásd a 3. ábra):

3. ábra. Lebegőpontos számábrázolás határai és a pontosan ábrázolható számok (2 bites mantissza, 3 bites exponens excess-4 módon tárolva).

1.4.1. példa. A mantissza 10 biten, az exponens 5 biten történő excess-15 ábrázolása esetén

- a tízezernél nagyobb, pontosan ábrázolható számok közül a legkisebb (lásd az 1.3.1. feladatot a 10000 ábrázolásához) a 0011100011 tárolt mantisszájú és a 11100 tárolt exponensű szám: $1.0011100011_{[2]} \cdot 2^{13} = 1.2216796875 \cdot 8192 = 10008$
- a tízezernél kisebb, pontosan ábrázolható számok közül a legnagyobb a 0011100001 tárolt mantisszájú és a 11100 tárolt exponensű szám: $1.0011100001_{[2]} \cdot 2^{13} = 1.2197265625 \cdot 8192 = 9992$

Az előzőekből adódik, hogy tízezer körül a hiba 8 (ami a tízezer 0.08%-a).

1.4.2. példa. A mantissza 10 biten, az exponens 5 biten történő excess-15 ábrázolása esetén

- az egy tízezrednél kisebb, pontosan ábrázolható számok közül a legnagyobb a 1010001101 tárolt mantisszájú és 00001 tárolt exponensű szám: $1.1010001101 \cdot 2^{-14} = 1.6376953125 \cdot 0.00006103515625 = 0.00009995698929$

- a tízezrednél nagyobb, pontosan ábrázolható számok közül a legkisebb a 0 előjelű, a 1010001110 tárolt mantisszájú és a 00001 tárolt exponensű szám: $1.1010001110 \cdot 2^{-14} = 1.638671875 \cdot 0.00006103515625 = 0.000100016593933$

Az előzőekből adódik, hogy egy tízezred körül a hiba kevesebb, mint egy tízmilliomod (ami az egy tízezred 0.1%-a).

1.4.3. feladat. A mantissza 10 biten, az exponens 5 biten történő excess-15 ábrázolása esetén adjuk meg (az előző két példához hasonló módon) az ezernél, a száznál, a tíznél, az egy tizednél, az egy századnál és az egy ezrednél nagyobb számok közül a legkisebb ábrázolhatót illetve a kisebb számok közül a legnagyobb ábrázolhatót, és számítsuk ki az abszolút, relatív hibát. Hogyan változik a relatív hiba az eltárolt szám függvényében?

A nem pontos tárolás akkor is látható, ha meggondoljuk, hogy a bináris normalizált alak felírásakor nem minden számjegy tárolható el, így azt kénytelenek vagyunk a tárolási méretre csökkenteni.

1.4.4. példa. A mantissza 10 biten, az exponens 5 biten történő excess-15 ábrázolása esetén a 10000.125 ábrázolása (lásd az 1.3.1. feladatot a 10000 ábrázolásához): $10000.125 = 10011100010000.001_{[2]} = 1.0011100010000001_{[2]} \cdot 2^{13}$, azaz tárolni kellene a 0 előjelet, a 00111000 10000001 mantisszát és a 13 exponenst, utóbbi az excess-15 ábrázolás miatt 11100 formában. Jól látszik azonban, hogy a mantissza 10 biten történő tárolása miatt a 0011100010000001 első tíz karaktere tárolható csak el, így kényszerű kerekítés történik: a 10000.125 helyett a 10000 kerül tárolásra. (Ezen nem csodálkozhatunk, hiszen az 1.4.1 példában láthattuk, hogy 10000 és 10008 között nem tárolható el más szám.)

1.4.5. feladat. Mi történik a lebegőpontos szám ábrázolási határaival ill. pontosságával, ha a

- mantissza méretét 1 bittel növelem?
- exponens méretét 1 bittel növelem?

1.5. Alulcsordulás

A túlcsorduláshoz (ami pozitív vagy negatív irányban túl nagy szám ábrázolásának kísérletét jelenti, azaz a számábrázolási intervallumból lépünk ki) hasonló az alulcsordulás [underflow]: olyan kis abszolút értékű számot akarunk ábrázolni, ami már nem ábrázolható.

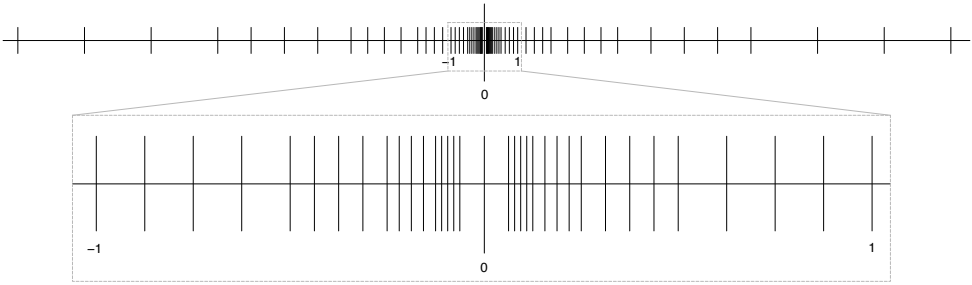
A kettes számrendszerben történő normalizált ábrázolásnak köszönhető megtakarítás (azaz hogy a mantissza első számjegye fixen 1, így a mantisszának csak az 1-től jobbra lévő részét tároljuk el, ezzel 1 bitet megtakarítva) hátrányos hatása itt jelentkezik: a legkisebb tárolható mantissza $1.0\ldots0$, a legkisebb karakterisztika $A^{K_{min}}$ (ahol A jelenti a számrendszer alapját, K_{min} pedig a legkisebb ábrázolható karakterisztikát), a legkisebb tárolható pozitív szám ezek szorzata: $1.0\ldots0 \cdot A^{K_{min}} = A^{K_{min}}$.

1.5.1. példa. 10 bites mantissza és 5 bites excess-15 exponens tárolás esetén a legkisebb ábrázolható pozitív szám: $1.0\ldots0\cdot2^{-15}=\frac{1}{2^{15}}=0.000030517578125$.

1.5.2. példa. 10 bites mantissza és 5 bites excess-15 exponens tárolás esetén a legnagyobb ábrázolható negatív szám: $-1.0\ldots0\cdot2^{-15}=-\frac{1}{2^{15}}=-0.000030517578125$.

Az előző két példa eredményéből következik, hogy a nullát sem tudjuk ábrázolni!

Sajnos a legkisebb pozitív és a legnagyobb negatív érték között a szomszédos távolságokhoz képest nagy ábrázolhatatlan tartomány húzódik, amit *alulcsordulási rés*nek [underflow gap] nevezünk (lásd a 4. ábrán). A kis számok ábrázolhatatlansága mellett sokkal nagyobb probléma az, hogy ha az eddigiek alapján járnánk el, akkor bármely két ábrázolható lebegőpontos szám különbsége nem biztos, hogy ábrázolható maradna, azaz nullával helyettesítődne. Ez óriási probléma a kis abszolút értékű számokkal dolgozó algoritmusok esetében: ha egy kivonás után nem garantálható, hogy az eredmény ábrázolható (azaz nem nulla), akkor hogyan lehetnénk abban biztosak, hogy a következő számítás nem fog hibához vezetni? Példák: Ha egy számot elosztunk másik két, nem nulla szám különbségével, lehet, hogy nullával fogunk osztani? Ha egy nem nulla számból kivonunk egy másik nem nulla számot, majd később ugyanezt hozzáadjuk, akkor nem fogjuk visszakapni az eredeti számunkat?



4. ábra. Lebegőpontos számábrázolás esetén
a nulla körüli alulcsordulási rés
(2 bites mantissza, 3 bites exponens excess-4 módon tárolva).

1.5.3. példa. Számítsuk ki 10 bites mantissza és 5 bites exponens excess-15 tárolása esetében a második legkisebb pozitív számot: $1.0000000001\cdot2^{-15}=2^{-15}+2^{-25}$. Ha ebből kivonjuk a legkisebb ábrázolható számot (lásd Az 1.5.1. példát) az eredmény 2^{-25} lesz, ami nyilván nem ábrázolható 10 bites mantisszán és 5 bites exponensen excess-15 formában (az eddig ismertetettek szerint).

2. Az IEEE 754 lebegőpontos számábrázolás

A lebegőpontos számok ábrázolásának a gyakorlatban is alkalmazott nemzetközi szabványa a az IEEE 754 = IEC 599 = ISO/IEC 60559. Az ebben definiált konkrét bináris lebegőpontos ábrázolások közül néhány látható az 5. ábrán. A szabvány a tárgyaltakon kívül még sok más tulajdonságot, funkciót is definiál, például a kerekítés szabályait, műveleteket, kivételkezelést, sőt tízes alapú lebegőpontos számábrázolást is, de ezekkel jelen tárgy keretében nem foglalkozunk.

elnevezés	mantissza mérete [bit]	karakterisztika mérete [bit]	karakterisztika eltolás
binary16	10	5	15
binary32	23	8	127
binary64	52	11	1023
binary128	112	15	16383

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

2.1. Tárolási sorrend

A lebegőpontos szám tárolási sorrendje a következő: előjel, exponens, mantissza.

2.1.1. példa. Az 1.3.1. példa esetében (binary16 formátum) tárolandó: 0 11100 0011100010.

2.1.2. példa. Az 1.3.2. példa esetében (binary16 formátum) tárolandó: 1 01011 0100000000.

2.1.3. példa. A binary16 formátumban tárolt 0000 0100 0000 0000 bitminta értelmezése: előjel: 0, exponens: 00001, exponens értéke: $1 - 15 = -14$, mantissza: 0000000000, kiegészítve a nem tárolt bittel: 1.0000000000, azaz a tárolt szám: $1.0000000000 \cdot 2^{-14} = 2^{-14}$.

2.1.4. példa. A binary16 formátumban tárolt 0000 0100 0000 0001 bitminta értelmezése: előjel: 0, exponens: 00001, exponens értéke: $1 - 15 = -14$, mantissza: 0000000001, kiegészítve a nem tárolt bittel: 1.0000000001 azaz a tárolt szám: $1.0000000001 \cdot 2^{-14} = 2^{-14} + 2^{-24}$.

2.1.5. feladat. Mondjuk példát olyan műveletre vagy relációra, amelyet könnyebb elvégezni, ha az eltolt számábrázolást használjuk az exponens tárolására és a tárolási sorrend: exponens, mantissza (azaz nem a normalizált alak sorrendje: mantissza, exponens)

2.2. Subnormált ábrázolás

Az alulcsordulási hiba megszüntetéséhez – az IEEE 754 szabványnak megfelelve – az előzőekben tárgyaltakkal ellentétben a nullaként tárolt exponens értéket speciálisan kell kezelni: ebben az esetben a mantissza legnagyobb helyiértékű bitjét az előzőekben megismert fix 1 helyett nullának kell értelmezni, és az exponens eltolását is eggyel csökkenteni kell (azaz például excess-15-ről excess-14-re):

2.2.1. példa. Az IEEE binary16 formátumban tárolt 0000 0000 0000 0001 bitminta értelmezése: előjel: 0, exponens: 00000, exponens értéke: mivel az exponens tárolt értéke 0, az eredeti (excess-15) értelmezés ($0 - 15 = -15$) helyett a módosított számítási mód (excess-14) szerint: $0 - 14 = -14$, mantissza: 0000000001, kiegészítve a nem tárolt bittel: 0.0000000001, azaz a tárolt szám: $0.0000000001 \cdot 2^{-14} = 2^{-24}$. Ez az IEEE binary16 formátumban tárolható legkisebb érték.

2.2.2. példa. Az IEEE binary16 formátumban tárolt 0000 0000 0000 0010 bitminta értelmezése: előjel: 0, exponens: 00000, exponens értéke: mivel az exponens tárolt értéke 0, az eredeti (excess-15) értelmezés ($0 - 15 = -15$) helyett a módosított számítási mód (excess-14) szerint: $0 - 14 = -14$, mantissza: 0000000010, kiegészítve a nem tárolt bittel: 0.0000000010, azaz a tárolt szám: $0.0000000010 \cdot 2^{-14} = 2^{-23} = 2 \cdot 2^{-24}$.

2.2.3. példa. Az IEEE binary16 formátumban tárolt 0000 0000 0000 0011 bitminta értelmezése: előjel: 0, exponens: 00000, exponens értéke: mivel az exponens tárolt értéke 0, az eredeti (excess-15) értelmezés ($0 - 15 = -15$) helyett a módosított számítási mód (excess-14) szerint: $0 - 14 = -14$, mantissza: 0000000011, kiegészítve a nem tárolt bittel: 0.0000000011, azaz a tárolt szám: $0.0000000011 \cdot 2^{-14} = 3 \cdot 2^{-24}$.

Ha az IEEE binary16 formátumban tárolható második legkisebb pozitív számból (lásd a 2.2.2. feladatot) kivonjuk az ábrázolható legkisebb pozitív számot (lásd a 2.2.1. feladatot), akkor eredményül 2^{-24}-et kapunk, ami szintén ábrázolható. Így az IEEE 754 számábrázolási módszerrel bármely két nem nulla ábrázolható szám különbsége kizárólag akkor nulla, ha a két szám megegyezik. Ez egy nagyon fontos numerikus tulajdonság!

2.2.4. példa. Az IEEE binary16 formátumban tárolt 0000 0000 0000 0000 bitminta értelmezése: előjel: 0, exponens: 00000, exponens értéke: mivel az exponens tárolt értéke 0, az eredeti (excess-15) értelmezés ($0 - 15 = -15$) helyett a módosított számítási mód (excess-14) szerint: $0 - 14 = -14$, mantissza: 0000000000, kiegészítve a nem tárolt bittel: 0.0000000000, azaz a tárolt szám: $0.0000000000 \cdot 2^{-14} = 0$.

2.2.5. példa. Az IEEE binary16 formátumban tárolt 1000 0000 0000 0000 bitminta értelmezése: előjel: 1, exponens: 00000, exponens értéke: mivel az exponens tárolt értéke 0, az eredeti (excess-15) értelmezés ($0 - 15 = -15$) helyett a módosított számítási mód (excess-14) szerint: $0 - 14 = -14$, mantissza: 0000000000, kiegészítve a nem tárolt bittel: 0.0000000000, azaz a tárolt szám: $-0.0000000000 \cdot 2^{-14} = -0$.

A nulla kétfajta tárolási módjának az a jelentősége, hogy jelezhető, hogy az alulcsordult szám milyen irányból közelítette meg a nullát. (Hasonlóan jelezzük például a határérték számításnál, hogy a nullát milyen irányból közelítjük meg: $\lim_{x\to 0-} f(x)$ illetve $\lim_{x\to 0+} f(x)$)

A http://babbage.cs.qc.cuny.edu/IEEE-754/index.xhtml oldalon kipróbálhatók, ellenőrizhetők az átváltások.

2.3. Végtelenek és a NaN

Az IEEE 754 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.

Ha az exponens minden bitje 1 és a mantissza nulla, akkor az (előjeltől függően) $\pm\infty$ az ábrázolt érték, ha a mantissza nem nulla, akkor NaN az ábrázolt érték.

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

2.4. Numerikus matematika

A numerikus matematika foglalkozik már meglévő számítási algoritmusok vizsgálatával illetve olyan új algoritmusok tervezésével, amelyek figyelembe veszik, hogy a számítógépen végrehajtott számítás során az előzőekben ismertetett hibák az egymás után végzett műveletek során ne nőjenek olyan nagyra, hogy magának az eredménynek a használhatóságát veszélyeztetnék.