Egész számok ábrázolása (jegyzet)

Bérci Norbert

2015. szeptember 10-i óra anyaga

Tartalomjegyzék

1. Számrendszerek

A számrendszer [numeral system - nem numeric system!] a szám (mint matematikai fogalom) írott formában történő megjelenítésére alkalmas módszer. 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.

[0] $Revision : 64$ ($Date : 2014 - 09 - 2013 : 31 : 30 + 0200(Sat, 20Sep2014)$)

1.1. A számrendszer alapja és a számjegyek

A helyiértéken alapuló számrendszerek két legfontosabb paramétere *a számrendszer alapja* [base, radix] és *az egyes pozíciókba írható számjegyek* [digit]. Ezek nem függetlenek: a számrendszer alapja meghatározza az egyes pozíciókba írható számjegyek maximumát: ha a számrendszer A alapú, akkor a legkisebb felhasználható számjegy a 0, a legnagyobb az $A - 1$.

1.1.1. példa. A tízes számrendszerben a $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ számjegyek szerepelhetnek, a nyolcas számrendszerben a $0, 1, 2, 3, 4, 5, 6, 7$ számjegyek közül választhatunk, míg a kettesben a $0, 1$ a két lehetséges számjegy.

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.

1.1.2. 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 $80_{[16]}$.

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*. Az előzőekben említett, indexben történő számrendszer megadás mellett bináris számrendszer jelölésére használatos a b postfix, oktális esetben egy kezdő 0 szerepeltetése, hexadecimális számok esetén a 0x, 0X prefixek vagy a h postfix. Az informatikában ezeket a jelöléseket használjuk a leginkább. Például: 100b (bináris), 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 leírtakat.

1.2. Alaki- és helyiérték

Egy adott számrendszerben leírt szám esetében egy *számjegy értéke* egyenlő a számjegy *alaki értéké*nek és *helyiérték*ének 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 ezek: a=10, b=11, c=12, d=13 stb.

1.2.1. 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 ebben a példában a 3 számjegy értéke: $3 \cdot 10^1 = 3 \cdot 10 = 30$.

1.2.2. 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 ebben a példában a 2 számjegy értéke: $2 \cdot 10^0 = 2 \cdot 1 = 2$.

1.3. Egész számok leírása

Egész számokat általános esetben az $a_n a_{n-1} \dots a_1 a_0$ alakban írhatunk fel, és az így felírt szám értéke (A alapú számrendszert feltételezve):

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

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

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

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

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

1.3.4. példa. $0\text{xA3} = 10 \cdot 16^1 + 3 \cdot 16^0 = 10 \cdot 16 + 3 \cdot 1 = 163$

A negatív egész számokat úgy írjuk le, hogy abszolút értéküket az előző módon felírjuk valamely számrendszerben, majd elé $-$ jelet teszünk (bár ezt a jelölést a tízes számrendszeren kívül a gyakorlatban nem alkalmazzuk).

1.4. Nem egész számok leírása

Az egész számoknál 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. Általános esetben tehát ennek alakja: $a_n a_{n-1} \dots a_1 a_0 a_{-1} \dots a_{-k}$, és az így felírt szám értéke (A alapú számrendszert feltételezve):

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

Annak érdekében, hogy a mindkét végén (egész- illetve tört rész) tetszőlegesen bővíthető felírás egyértelmű legyen, ennek a két résznek a határát jelöljük tizedesvesszővel. Mi a magyar helyesírással ellentétben, a nem egész számok felsorolásának könnyebb olvashatósága érdekében a továbbiakban a tizedespontos[1] jelölést fogjuk alkalmazni. (Pl. 1,6, 2,4, 5,9 helyett $1.6, 2.4, 5.9$)

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

1.4.2. példa. $405.23_{[8]} = 4 \cdot 8^2 + 0 \cdot 8^1 + 5 \cdot 8^0 + 2 \cdot 8^{-1} + 3 \cdot 8^{-2} = 4 \cdot 64 + 5 \cdot 1 + 2 \cdot \frac{1}{8} + 3 \cdot \frac{1}{8^2} = 256 + 5 + \frac{2}{8} + \frac{3}{64} = 261\frac{19}{64} = 261.296875$

[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).

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

Negatív nem egész számok leírása a negatív egész számok leírásához hasonlóan a $-$ jel szám elé írásával történik (amit szintén csak a tízes számrendszer esetében használunk).

1.5. Átváltás számrendszerek között

Az adott számrendszerből tízes számrendszerbe váltást az 1.3 és az 1.4 részek példáiban hallgatólagosan már bemutattuk. A fordított átváltásra nem térünk ki (a módszer könnyen kitalálható, lásd 1.6.5. feladat).

Az átváltás nagymértékben egyszerűsödik, ha 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:

1.5.1. példa. $1010111001_{[2]} = 001\ 010\ 111\ 001_{[2]} = 1271_{[8]}$

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:

1.5.2. példa. $2b9_{[16]} = 0010\ 1011\ 1001_{[2]} = 1010111001_{[2]}$

Oktálisból hexadecimálisba vagy decimálisból hexadecimálisba illetve fordítva a bináris számrendszert közbeiktatva is átválthatunk ezzel a módszerrel:

1.5.3. példa. $2b9_{[16]} = 0010\ 1011\ 1001_{[2]} = 1010111001_{[2]} = 001\ 010\ 111\ 001_{[2]} = 1271_{[8]}$

1.6. Feladatok

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

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

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

1.6.4. feladat. $9a2d_{[16]} = ?_{[2]} = ?_{[8]} = ?_{[16]}$

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

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

A http://www.exploringbinary.com/binary-converter/ oldalon kipróbálhatók, ellenőrizhetők az átváltások.

1.7. Számrendszerek pontossága

Fontos kiemelni, hogy nem egész számok felírása esetén nem biztos, hogy a szám pontosan leírható véges számjeggyel! Sőt, egy konkrét nem egész szám ábrázolásának pontossága függ a számrendszer alapjától: például az $\frac{1}{3}$ tízes számrendszerben nem írható fel véges számjeggyel, ugyanakkor hármas számrendszerben pontosan felírható: $\frac{1}{3} = 0.1_{[3]} = 0.33333\ldots_{[10]}$

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

1.7.2. feladat. a) Adjunk meg néhány példát olyan számra, ami egyetlen számrendszerben sem írható fel véges számjeggyel! b) Felírhatók ezek a számok tört alakban? c) Milyen számhalmazt alkotnak ezek a számok?

1.7.3. feladat. Kiválasztható olyan alapú számrendszer, amiben minden racionális szám pontosan ábrázolható véges hosszú karaktersorozattal? Indokoljuk meg!

2. Mértékegységek

Az informatikában használatos legkisebb egység a bit [bit] (sok esetben b-vel rövidítik, de a legfrissebb szabvány[2] a rövidítés nélküli formát ajánlja, ami kézenfekvő a számrendszerek részben tárgyaltak miatt, hiszen a b postfix a bináris számrendszert jelöli). Értéke 0 vagy 1 lehet. Használhatjuk tárolókapacitás vagy információmennyiség 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 a 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 alapja valamely hatványának hatványai. Az

[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

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).

SI		bináris	
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. Egész számok gépi ábrázolása

A gépi számábrázolás a számok (számító)gépek memóriájában vagy egyéb egységében történő tárolását vagy valamely adathálózaton történő továbbítás formátumát adja meg.

3.1. Nem negatív egész számok ábrázolása

Egy nem negatív (előjel nélküli) egész szám [unsigned integer] ábrázolása megegyezik a bináris számrendszernél megismert leírással, 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 binárisból hexadecimálisba váltás nem más, mint négy bitesével csoportosítás, ahogy azt az előzőekben láthattuk.)

A kapott értékeket általában valamilyen fix hosszon tároljuk (a nem használt helyiértékekre nullát írunk), ami a gyakorlatban kizárólag egész byte méretű ábrázolást jelent. Így az előjel nélküli egészek is legtöbbször 1, 2, 4, 8, ... byte (8, 16, 32, 64, ... bit) hosszúak lehetnek. Így is hívjuk ezeket: 8 bites előjel nélküli egész, 16 bites előjel nélküli egész stb.

3.1.1. példa. A $46_{[10]}$ számot a memóriában a következőképpen tároljuk 1 bájton: 00101110 (=0x2E).

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

3.1.3. feladat. Hogyan dönthető el két előjel nélküli egész számról, hogy melyik a nagyobb? Adjunk rá algoritmust!

3.2. Negatív egész számok ábrázolása

Ebben a részben a negatív egészek ábrázolásának változatait tekintjük át.

3.2.1. Előjelbites ábrázolás

A legegyszerűbb módszer az előjeles egészek ábrázolására, ha az előjel nélküli egészek ábrázolásához egy előjelet jelentő bitet adunk (ami 0, ha pozitív az előjel és 1, ha negatív az előjel) és az ábrázolásból fennmaradó többi biten tároljuk a szám abszolút értékét az előzőekben tárgyaltak szerint.

3.2.1. példa. A -32 előjelbites ábrázolása 8 biten (1 bit előjel + 7 bit érték): 10100000

3.2.2. példa. A 18 előjelbites ábrázolása 8 biten (1 bit előjel + 7 bit érték): 00010010

Ez a megoldás sok szempontból nem megfelelő: a legkézenfekvőbb probléma, hogy ezzel a módszerrel lehetséges a $+0$ és a -0 ábrázolása is (8 biten ezek a következők: $+0 = 00000000$, $-0 = 10000000$), 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.

3.2.3. feladat. A 3.1.2. feladatban kitalált összeadás művelet elvégezhető-e *módosítás nélkül* az előjelbites számábrázolási módszer használatával? Adjunk meg egy példát!

3.2.4. feladat. Módosítsuk a 3.1.3. feladatban kitalált algoritmust, hogy az két előjeles szám közül is ki tudja választani a nagyobbikat!

3.2.2. Kettes komplemens ábrázolás

Sokkal jobb eredményre vezet a *kettes komplemens* ábrázolás: ahelyett, hogy egy előjelbittel jelölnénk az előjelet, a következő módon járunk el: a negatív számhoz egyet hozzáadunk, az eredmény abszolút értékét binárisan ábrázoljuk a megadott számú biten (az előzőekben tárgyaltak szerint, mivel ez nem negatív), végül az így kapott számjegyeket invertáljuk. Ebből a számítási módból következik az ábrázolás neve: kettes komplemens.

A kettes komplemens számábrázolási módszert *előjeles egész* [signed integer] számábrázolásnak nevezzük.

3.2.5. példa. A -2 kettes komplemens ábrázolása 8 biten: $-2 + 1 = -1$ ennek abszolút értéke: 1, ábrázolva: 00000001, invertálva: 11111110.

3.2.6. példa. A -19 kettes komplemens ábrázolása 8 biten: $-19 + 1 = -18$ ennek abszolút értéke: 18, ábrázolva: 00010010, invertálva: 11101101.

Fontos tudnivalók:

- Kettes komplemens ábrázolásban is lehetséges nem negatív számok ábrázolása, aminek módja megegyezik az előjel nélküli egészek tárolási módjával. (Azaz ebben az esetben nem kell az előzőekben ismertetett műveleteket elvégezni.)
- A kettes komplemens ábrázolásban már csak egyetlen ábrázolási módja van a nullának.
- Az esetek túlnyomó többségében a gépi számábrázolás során az előjeles egészek ábrázolására a kettes komplemens ábrázolást használjuk.

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

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

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

3.2.10. feladat. A 3.1.2. feladatban kitalált összeadás művelet elvégezhető-e *módosítás nélkül* a kettes komplemens számábrázolási módszer használatával? Adjuk össze az előző két feladatban kiszámolt, 8 bites -1 és 1 értéket, és ellenőrizzük, hogy nullát kaptunk-e!

3.2.11. feladat. Két, kettes komplemens módon ábrázolt számról hogyan dönthető el, hogy melyik a nagyobb? Alkalmazható *módosítás nélkül* ugyanaz az algoritmus, mint a 3.1.3 feladatban?

3.2.3. Eltolt ábrázolás

Soroljuk fel egy listában az n biten történő előjel nélküli számábrázolással felírható értékeket növekvő sorrendben. Az *eltolt* [excess] számábrázolási módszer ezeket az eltolás mértékében lefelé tolja úgy, hogy az újonnan belépő elemek az érték szerint csökkenő negatív számok legyenek (lásd 2. ábra).

3.2.12. feladat. Létezik olyan excess ábrázolás, ami a negatív számok esetében megegyezik a kettes komplemens ábrázolással?

A lista eltolása helyett az ábrázolandó értékeket úgy is megkaphatjuk, hogy az ábrázolandó számhoz hozzáadjuk az eltolás mértékét, és az eredményül kapott számot ábrázoljuk az előjel nélküli egész számábrázolási módszere szerint.

3.2.13. feladat. Mi biztosítja, hogy az előző módszer működik? (Mi garantálja, hogy nem negatív számot kapunk, ha az ábrázolandó számhoz hozzáadjuk az eltolás mértékét?) Ha mégsem működik, az mit jelent?

3.2.14. példa. A -2 excess-2 ábrázolása 3 biten: $-2 + 2 = 0$, tehát ábrázolandó a 0 a nem negatív egészek ábrázolása szerint: 000 (lásd 2. ábra első sora).

tárolt adat	adat értelmezése		
3 biten	előjel nélküli egész	excess-2	excess-4
000	0	-2	-4
001	1	-1	-3
010	2	0	-2
011	3	1	-1
100	4	2	0
101	5	3	1
110	6	4	2
111	7	5	3

2. ábra. A 3 biten tárolható értékek előjel nélküli egész és excess-2 illetve excess-4 szerinti értelmezése

3.2.15. példa. A -3 excess-4 ábrázolása 3 biten: $-3+4=1$, tehát ábrázolandó az 1 a nem negatív egészek ábrázolása szerint: 001 (lásd 2. ábra második sora).

3.2.16. példa. Az 5 excess-2 ábrázolása 3 biten: $5+2=7$, tehát ábrázolandó a 7 a nem negatív egészek ábrázolása szerint: 111 (lásd 2. ábra utolsó sora).

3.3. Egész számok adatábrázolásainak összehasonlítása

3.3.1. feladat. Hasonlítsuk össze az előzőekben ismertetett, negatív számok ábrázolására is alkalmas módszereket az alábbi szempontok alapján:

- Az összeadás művelet elvégezhető ugyanúgy, mint a nem negatív egészek ábrázolásánál?
- Két ábrázolt szám esetében a kisebb/nagyobb eldöntése (rendezés) elvégezhető ugyanúgy, mint a nem negatív egészeknél?
- Hányféleképpen ábrázolható a nulla?
- Hogyan végezhető el az invertálás (diszkrét matematikai nyelven az additív inverz számítása)?
- Hogyan végezhető el a kivonás művelet?

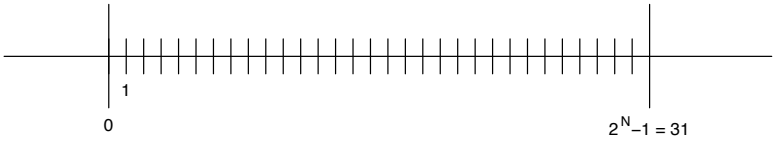
3.3.2. feladat. Hasonlítsuk össze a 8 bites számábrázolások esetén az előjel nélküli egész, az előjelbites egész, a kettes komplemens, a 127-tel eltolt, a 255-tel eltolt és a 256-tal eltolt számábrázolásokat! (Táblázatosan foglaljuk össze: egy sor legyen a tárolt 8 bit, az oszlopok legyenek a vizsgált ábrázolási módok, egy adott mezőbe írjuk be a mező sorának megfelelő bitsorozat értelmezését az oszlopnak megfelelő számábrázolás esetében, hasonlóan a 2. ábrához!)

3.4. Egész számok ábrázolási határai és pontossága

3.4.1. Előjel nélküli egész tárolás ábrázolási határai és pontossága

Az N biten történő, előjel nélküli egész számábrázolás esetén a tárolható legkisebb érték: 0, a tárolható legnagyobb érték: $2^N - 1$.

Előjel nélküli egész számábrázolás esetében a tárolás pontos, hiszen csak egész számokat kell tárolni, és a határokon belül minden egész szám pontosan tárolható. Ebből adódóan az ábrázolási intervallumot az ábrázolható számok egyenletesen töltik ki (lásd a 3. ábrán).



3. ábra. 5 bites előjel nélküli egész számábrázolás esetén
az ábrázolási intervallum és az ezen belül ábrázolható számok.

3.4.1. 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 255).

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

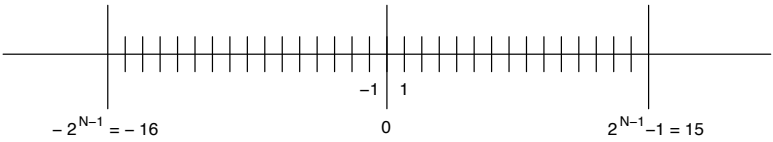
3.4.3. feladat. Összesen hány különböző érték tárolható 8, 16, 32, 64 biten, előjel nélküli egész számábrázolás esetében?

3.4.2. Kettes komplemens tárolás ábrázolási határai és pontossága

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: $2^{N-1} - 1$. Kettes komplemens számábrázolás esetében a tárolás pontos, hiszen csak egész számokat kell tárolni, és a határokon belül minden egész szám pontosan tárolható. Ebből adódóan az ábrázolási intervallumot az ábrázolható számok egyenletesen töltik ki (lásd a 4. ábrán).

3.4.4. 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).

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



4. ábra. 5 bites kettes komplemens számábrázolás esetén
az ábrázolási intervallum és az ezen belül ábrázolható számok.

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

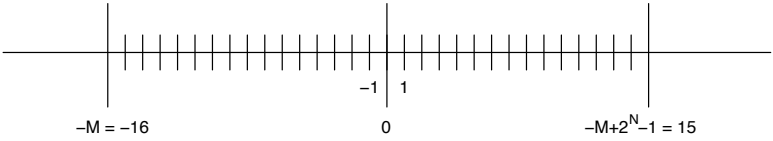
3.4.7. feladat. Eldönthető egyszerűen (ránézésre) egy kettes komplemens módon ábrázolt számról, hogy az negatív vagy pozitív?

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

3.4.9. feladat. Mi a kapcsolat a 3.4.3. feladat és a 3.4.8. feladatban kapott eredmények között?

3.4.3. Eltolt tárolás ábrázolási határai és pontossága

Az N biten történő eltolt-M ábrázolás esetén a legkisebb ábrázolható szám a $-M$, a legnagyobb ábrázolható szám a $-M+2^N-1$. Eltolt számábrázolás esetében a tárolás pontos, hiszen csak egész számokat kell tárolni, és a határokon belül minden egész szám pontosan tárolható. Ebből adódóan az ábrázolási intervallumot az ábrázolható számok egyenletesen töltik ki (lásd az 5. ábrán).



5. ábra. 6 bites excess-16 számábrázolás esetén
az ábrázolási intervallum és az ezen belül ábrázolható számok.

3.4.4. 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 [overflow] nevezzük. Túlcsordulás tehát lehetséges pozitív és negatív irányban is! Figyelem, az alulcsordulás (lásd a **??**. részt) *nem* a

negatív irányban történő túlcsordulást jelenti! Könnyebben megjegyezhető, ha úgy tekintünk a túlcsordulásra, hogy a szám abszolút értéke túl nagy és emiatt nem ábrázolható.

Túlcsordulás esetén – megvalósítástól függően – lehetséges

- *levágás*: a túlcsordult eredmény még ábrázolható részét tároljuk, a nem ábrázolható részt egyszerűen „elfelejtjük”. [4] A legtöbb architektúra így működik.
- *szaturáció*: a túlcsordult eredmény helyett a legnagyobb illetve legkisebb ábrázolható értéket tároljuk.

3.4.10. példa. Túlcsordulás pozitív irányban: 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).

3.4.11. példa. Túlcsordulás negatív irányban: ha 8 bites előjeles egészekkel dolgozunk, a -84+(-79)=-163 összeget már nem tudjuk 8 biten tárolni (mert a legkisebb tárolható érték a -127).

3.4.12. 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.

3.4.13. 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.

3.4.14. feladat. Mi (volt) az Y2K probléma? Mi a kapcsolat a túlcsordulás és az Y2K probléma között?

[4] Például mechanikus gázóránál vagy régebbi autók kilométer számlálójánál figyelhető meg ilyen jelenség, mert fix számú helyiértéken történik a mérés. A kilométer számlálók tekintetében ezt a tulajdonságot kihasználva tekerik körbe egyes nepperek az órát, hogy a kocsi kevesebbet futottnak tűnjön.