1 Aritmetika s pomičnom točkom

Zapis broja -27.77 u tzv. **znanstvenoj notaciji** glasi

$$-2.777 \times 10^1,$$

pri čemu je:

- $-$ **predznak** broja,
- 2.777 **mantisa** ili razlomljeni dio broja,
- 10 **baza**,
- 1 **eksponent**.

Općenito, svaki se realni broj može na jedinstven način zapisati u obliku

$$\pm m \times 10^e, \quad 1 \leq m < 10.$$

Računala koriste sličan zapis, ali ne u bazi 10 već u bazi 2.

Npr. broj $3.625_{10} = 11.1011_2$ (zapisan u bazi 2) u znanstvenoj notaciji ima zapis

$$\begin{aligned} 11.1011_2 &= 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} \\ &= \left(1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5}\right) \cdot 2^1 \\ &= 1.11011_2 \cdot 2^1 \end{aligned}$$

1.1 Pretvaranje decimalne u binarnu reprezentaciju

Neka broj x ima konačan binarni zapis. Tada x možemo zapisati kao sumu njegovog cijelog i razlomljenog dijela na način:

$$\begin{aligned} x &= (a_k a_{k-1} \cdots a_1 a_0 . b_1 b_2 \cdots b_{l-1} b_l)_2 = x_c + x_r \\ x_c &= (a_k a_{k-1} \cdots a_1 a_0)_2 \\ &= a_k 2^k + a_{k-1} 2^{k-1} + \cdots + a_1 2^1 + a_0 2^0 \\ x_r &= (. b_1 b_2 \cdots b_{l-1} b_l)_2 \\ &= b_1 2^{-1} + b_2 2^{-2} + \cdots + b_{l-1} 2^{-l+1} b_l 2^{-l}. \end{aligned}$$

Pitanje je sljedeće: ako je cijeli broj x zadan u decimalnom obliku, kako odrediti njegove binarne znamenke $a_k a_{k-1} \cdots a_1 a_0$ koristeći nama razumljivu decimalnu aritmetiku?

Očito je broj x potrebno dijeliti s 2 uzastopce (u našem primjeru k puta) sve dok ne dobijemo ostatak $a_0 \in \{0, 1\}$. Kako je x bio cijeli broj, on će i u binarnom zapisu imati konačan cjelobrojni prikaz. Pri tom će binarni prikaz imati točno

$$[\log_2 x_c] + 1$$

binarnih znamenaka, odnosno bitova. Oznaka $[q]$ označava najveći cijeli broj koji stane u q.

Ako x_c ima p decimalnih znamenaka, onda vrijedi

$$10^{p-1} \leq x_c \leq 10^p.$$

Elementarnim računom dobijemo

$$\log_2 10^{p-1} \leq \log_2 x_c \leq \log_2 10^p,$$

odnosno

$$(p-1)\log_2 10 \leq \log_2 x_c \leq p \log_2 10.$$

Kako je $\log_2 10 \approx 3.3219$, to binarna reprezentacija zahtijeva oko $3.3p$ znamenaka, dakle oko 3.3 puta više znamenaka nego decimalna.

Promotrimo sada x_r. Uzastopnim množenjem broja x_r i uzimanjem cijelog dijela dobijemo traženi prikaz. Postupak, naravno, može biti i beskonačan.

1.2 Reprezentacija brojeva u računalu

U programskim jezicima postoji nekoliko vrsta aritmetika koje korite posve određene tipove podataka.

- **Cjelobrojna aritmetika** koristi cjelobrojni tip podataka koji čine konačan interval u skupu cijelih brojeva $\mathbb{Z}$.
- **Realna aritmetika** koristi realni tip podataka koji čine interval s konačnim granicama u skupu racionalnih brojeva $\mathbb{Q}$.
- **Aritmetika s dvostrukom preciznošću** koristi zapis realnih brojeva kod koga je mantisa dvostruko dulja od mantise u standardnom zapisu brojeva realnog tipa.
- **Kompleksna aritmetika** koristi kompleksni tip podataka koji čine konačan podskup skupa kompleksnih brojeva $\mathbb{C}$.

Općenito, današnja računala imaju ćelije od 32 bita, pa je tomu prilagođena i aritmetika. Iznimno se koriste i računala s 64 bitnim ćelijama.

1.2.1 Reprezentacija cijelih brojeva u računalu

Pozitivni cijeli brojevi reprezentiraju se u 32 bitnoj ćeliji kao desno pozicionirani binarni brojevi. Npr., broj $(111)_{10} = (1101111)_2$ bit će smješten kao

$$0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.1.0.1.1.1.1.$$

Na taj način možemo prikazati sve brojeve od nule (32 nule) do $2^{32} - 1$ (32 jedinice). Međutim, ako moramo spremati i negativne brojeve, onda se situacija mijenja.

Prva je ideja potrošiti jedan bit za predznak što bi nam omogućilo prikaz brojeva od $-2^{31} + 1$ do $2^{31} - 1$. Ipak, gotovo sva računala koriste pametniji zapis: on se zove **drugi komplement**.

U tom sustavu se nenegativni cijeli broj $0 \leq x \leq 2^{32}-1$ smiješta kao binarna reprezentacija tog broja, dok se $-x,\ 1 \leq x \leq 2^{32}-1$ smiješta kao binarna reprezentacija broja $2^{32}-x$. Prvi komplement dobiva se jednostavnim komplementiranjem znamenaka, pa se za broj iz našeg primjera $(111)_{10}$ dobije

$$1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.0.0.1.0.0.0.0.$$

a dodamo li 1 toj reprezentaciji dobijemo drugi komplement

$$1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.0.0.1.0.0.0.1.$$

tj. dobijemo reprezentaciju broja $-111.$

Uočimo da binarnim zbrajanjem reprezentacija brojeva 111 i -111 dobijemo 1 i iza njega 32 nule, pa se 1 kao prekobrojni bit odbacuje.

Promotrimo operaciju $x+(-y)$, gdje su $0 \leq x, y \leq 2^{32}-1$. Kako $-y$ ima reprezentaciju $2^{32}-y$, to se zbroj može pisati kao

$$x+(-y) = 2^{32}+x-y = 2^{32}-(y-x).$$

Ako je $x \geq y$, krajnji lijevi bit u reprezentaciji broja $2^{32}+x-y$ bit će jedinica na poziciji 2^{32} i ona će se odbaciti, pa će ostati točan rezultat $x-y$. Ako je $x < y$, onda broj $2^{32}-(y-x)$ ostaje unutar 32 bitne reprezentacije i reprezentira broj $-(y-x)$.

Dakle, oduzimanje kao posebna operacija je suvišno!

1.2.2 Reprezentacija realnih brojeva u računalu

U računalu se realni brojevi reprezentiraju u **znanstvenoj notaciji**

$$x = \pm m \times 2^e, \quad 1 \leq m < 2.$$

Dakle,

$$m = (b_0.b_1 b_2 \cdots b_{p-1})_2 , \quad b_0 = 1.$$

Ovakav prikaz nazivamo **normaliziranom reprezentacijom** broja x.
Na primjer,

$$(111.5)_{10} = (1101111.1)_2 = (1.1011111)_2 \times 2^6.$$

Da bismo ovako reprezentiran broj spremili u računalo 32 bitnu čeliju razdijelimo na sljedeći način:

1. 23 bita za mantisu (za razlomljeni dio, b_0 je tzv. skriveni bit),
2. 8 bitova za eksponent (od -128 do 127),
3. 1 bit za predznak (0 za pozitivan, a 1 za negativan predznak).

Broj x nazivamo **egzaktno reprezentabilnim** u računalu ako se na opisani način može bez greške smjestiti u računalo. Ako broj nije egzaktno reprezentabilan u računalu, onda se mora prije smještanja u računalu **zaokružiti**.

PRIMJER 1. Vidjeli smo da je

$$x = (111.1)_{10} = \left(1101111.0\dot{0}01\dot{1}\right)_2 .$$

Nakon normalizacije dobijemo

$$x = \left(1.10111100\dot{0}1\dot{1}\right)_2 \times 2^6.$$

Očito je da sve znamenke ne mogu stati u 23 bita mantise, pa se spremi:

1. predznak 0,
2. eksponent $(6)_{10} = (00000110)_2$,
3. mantisa 10111100011001100110011.

PRIMJER 2. Pogledajmo broj

$$y = \left(-2^{22}\right)_{10} = (-4194304)_{10}\,.$$

On ima egzaktan zapis jer mu je:

1. predznak 1,
2. eksponent $(22)_{10} = (10110)_2\,$,
3. mantisa 00000000000000000000000.

Ako su brojevi koje želimo spremiti preveliki, onda dolazi do **preljeva** ili **prekoračenja**. Što će računalo napraviti ovisi o compileru za jezik u kojem radimo. Najčešće se prekida računanje uz adekvatnu poruku. Ovo nije slučaj kod cjelobrojnog računa jer se tamo račun provodi po $\mathrm{mod}\, 2^{32}$.

1.2.3 Strojni ε, ulp i preciznost

Preciznost definiramo kao broj bitova u mantisi pri čemu se računa i skriveni bit. U opisanom sustavu je $p = 24$. Općenito, u sustavu s preciznošću p, normalizirani broj s pomičnom točkom ima oblik

$$\pm \left(b_0.b_1b_2 \cdots b_{p-1}b_{p-1}\right)_2 \times 2^e. \tag{1}$$

Najmanji takav broj već od 1 je broj

$$\left(1.00 \cdots 01\right)_2 \times 2^0 = 1 + 2^{-p+1}.$$

Razmak između njega i jedinice naziva se **strojni** ε i piše se

$$\varepsilon_M = 2^{-p+1}.$$

Za x reprezentiran kao u (1) definiramo **ulp** (*unit in the last place*) kao

$$\text{ulp}\,(x) = \left(0.00 \cdots 01\right)_2 \times 2^e = 2^{-p+1} \times 2^e = \varepsilon_M \times 2^e.$$

Ako je $x > 0$ $(x < 0)$, onda je ulp(x) razmak između x i sljedećeg većeg (manjeg) reprezentabilnog broja.

1.2.4 BSP ili brojevni sustav za prikazivanje

Da bismo shvatili koje točke na realnom pravcu odgovaraju reprezentabilnim brojevima uvodimo **brojevni sustav za prikazivanj**e ili kraće BSP. On se sastoji od brojeva oblika

$$\pm (b_0.b_1b_2)_2 \times 2^e, \quad e \in \{1-, 0, 1\}\,.$$

Očito, preciznost je $p = 3$, najveći prikazivi broj je

$$(1.11)_2 \times 2^1 = (3.5)_{10}\,,$$

a najmanji

$$(1.00)_2 \times 2^{-1} = (0.5)_{10}\,.$$

Kako je desni susjed broja 1 u BSP-u broj 1.25, to je $\varepsilon_M = 0.25$. Brojevi za koje je $e = 0$ su 1, 1.25, 1.5 i 1.75. Između svih je isti razmak ulp$(x) = \epsilon_M$. Brojevi za koje je $e = 1$ su 2, 2.5, 3 i 3.5, a između svih je isti razmak ulp$(x) = 2\epsilon_M$. Brojevi za koje je $e = -1$ su 0.5, 0.625, 0.75 i 0.825, a između svih je isti razmak ulp$(x) = \epsilon_M/2$.

Dakle, općenito je razmak između nekog broja i njegovog desnog susjeda u BSP-u jednak

$$ulp\,(x) = \epsilon_M \times 2^e.$$

Uočimo da razmaci nisu ravnomjerno raspoređeni!

2 IEEE aritmetika

Ovo je standard smještanja brojeva u računalo i računalne aritmetike koji je 1985. ustanovljen između vodećih proizvođača procesora (Intel i Motorola) te znanstvenika iz Berkeley-a.

Bitni zahtjevi IEEE standarda su:

- konzistentna reprezentacija brojeva s pomičnom točkom na svim računalima koja prihvaćaju standard;
- korektno zaokruživanje kod svih računalnih operacija u svim načinima rada;
- konzistentno tretiranje izvanrednih situacija (npr. kao što je dijeljenje s nulom).

U ovom standardu je vodeća jedinica skrivena, pa je potreban poseban načini prikaza nule. Također se posebno prikazuju $+\infty$ i $-\infty$ (to je u ovom standardu isti broj!) i neki posebni izmišljeni brojevi (kao $0/0$).
Ovaj standard razlikuje dva osnovna formata: **jednostruki** i **dvostruki**.

2.1 Jednostruki format

U jednostrukom formatu ćelija za spremanje broja organizirana je na sljedeći način:

$$\boxed{\pm}\boxed{a_1 a_2 \ldots a_8}\boxed{b_1 b_2 \cdots b_{23}}.$$

Pri tom na prvom mjestu stoji nula ako je broj pozitivan, a jedan ako je broj negativan. Nula je reprezentirana s

$$\boxed{\pm}\boxed{00\ldots0}\boxed{000\cdots00}$$

i za nju se ne smatra da joj je skriveni bit 1!

Eksponent nije prikazan ni kao drugi komplement niti kao pozitivan broj s predznakom, već kao $127 + e$. Ovaj broj zvat ćemo karakteristika. Npr. broj 1 se reprezentira kao

$$\boxed{0}\boxed{01111111}\boxed{000\cdots00}.$$

Uočimo da za eksponent vrijedi

$$\begin{aligned}(01111110)_2 &= (126)_{10} \longrightarrow 2^{-1},\\ (01111111)_2 &= (127)_{10} \longrightarrow 2^{0},\\ (10000000)_2 &= (128)_{10} \longrightarrow 2^{1},\\ &\dots\end{aligned}$$

Tako se broj

$$(0.1)_{10} = \left(1.1\dot{0}01\dot{1}\right)_2 \times 2^{-4}$$

uz odbacivanje viška znamenaka reprezentira kao

0	01111011	10011001100110011001100

jer se uzima u obzir da je

$$127 - 4 = 123 = (01111011)_2\,.$$

Evo nekih važnih podataka:

- **najmanji eksponent** je -126
- **najveći eksponent** je 127
- **najmanji normalizirani broj** je

$$N_{\min} = (1.00\ldots0)_2 \times 2^{-126} \approx 1.1755 \times 10^{-38}$$

  a reprezentira se kao

0	00000001	$000\cdots00$

- **najveći normalizirani broj** je

$$N_{\max} = (1.11\ldots1)_2 \times 2^{127} \approx 3.4028 \times 10^{38}$$

  a reprezentira se kao

0	11111110	$111\cdots11$

- niz bitova samih jedinica u eksponencijalnom dijelu reprezentacije vodi na $\pm\infty$ ako su u mantisi same nule, a inače dobivamo oznaku NaN (not a number)
- $\varepsilon_M = 2^{-23} \approx 1.2 \times 10^{-7}$

Postoje još i tzv. **subnormalni** ili **denormalizirani** brojevi kojima je, kao i nuli, skriveni bit jednak 0. Najmanji takav pozitivan broj je

$$(0.00\ldots1)_2 \times 2^{-126} = 2^{-149} < N_{\min},$$

dok je najveći

$$(0.11\ldots1)_2 \times 2^{-126} = N_{\min} - 2^{149}.$$

Subnormalni brojevi su manje točni od normaliziranih.

2.2 Dvostruki format

Kod zahtjevnijih računanja jednostruki format nije dovoljno dobar, kako zbog neizbježnih zaokruživanja, tako i zbog premalog raspona brojeva. Zato IEEE standard specificira i tzv. dvostruki format koji koristi 64 bitnu riječ

$\pm$	$a_1 a_2 \dots a_{11}$	$b_1 b_2 \cdots b_{52}$

.

Koncept je isti kao kod jednostrukog formata. Neki važni podaci su:

- **najmanji eksponent** je -1022
- **najveći eksponent** je 1023
- **najmanji normalizirani broj** je

$$N_{\min} = 2^{-1022} \approx 2.225 \times 10^{-308}$$

- **najveći normalizirani broj** je

$$N_{\max} = \left(2 - 2^{-52}\right) \times 2^{1023} \approx 1.797693 \times 10^{308}$$

- $\varepsilon_M = 2^{-52} \approx 2.2 \times 10^{-16}$

2.3 BSPT (Brojevni Sustav s Pomičnom Točkom)

Za realni broj x kažemo da leži u **intervalu normaliziranih brojeva** sustava s pomičnom točkom ako vrijedi

$$N_{\min} \leq x \leq N_{\max}.$$

Iz ovoga odmah slijedi da brojevi ± 0 i $\pm\infty$ nisu u tom intervalu, iako pripadaju spomenutom sustavu.

Pretpostavimo sada da je x neki realni broj koji nije reprezentabilan u sustavu brojeva s pomičnom točkom. Tada je točna barem jedna od sljedećih tvrdnji:

- x leži izvan intervala normaliziranih brojeva;
- binarna reprezentacija broja x zahtjeva više od p bitova za egzaktnu reprezentaciju.

U oba slučaja potrebno je x zamijeniti brojem iz sustava brojeva s pomičnom točkom.

- Razmotrimo ovu drugu mogućnost. Neka je

$$x_- \leq x \leq x_+,$$

pri čemu su brojevi s pomičnom točkom x_- i x_+ brojevi skupa BPT najbliži broju x. Ako pretpostavimo da je

$$x = (1.b_1 b_2 \cdots b_{p-1} b_p b_{p+1} \cdots)_2 \times 2^e,$$

onda su ti brojevi upravo

$$\begin{aligned} x_- &= (1.b_1 b_2 \cdots b_{p-1})_2 \times 2^e \\ x_+ &= \left[(1.b_1 b_2 \cdots b_{p-1})_2 + (0.00 \cdots 01)\right] \times 2^e, \end{aligned}$$

a razmak između x_- i x_+ je $\text{ulp}(x_-) = 2^{p-1} \times 2^e$.

- Ako je $x > N_{\max}$, onda je $x_- = N_{\max}$ i $x_+ = +\infty$.
- Ako je $0 < x < N_{\min}$, onda je $x_- = 0$ ili je x_- neki subnormalni broj, a x_+ je subnormalni broj ili $N_{\min}$.
- Ako je x negativan, onda je situacija zrcalna u odnosu na ishodište.

IEEE standard definira **korektno zaokruženu vrijednost** broja x, u oznaci **round**(x), na način koji slijedi. Ako je x broj s pomičnom točkom, onda je round$(x) = x$. Ako nije, onda se round(x) odredi na jedan od četiri načina ovisno o načinu (modu) zaokruživanja koji je aktivan. Ti načini su:

- round$(x) = x_-$ (**zaokruživanje prema dolje**),
- round$(x) = x_+$ (**zaokruživanje prema gore**),
- round$(x) = \begin{cases} x_- \,, & x > 0 \\ x_+ \,, & x < 0 \end{cases}$ (**zaokruživanje prema nuli**)
- round$(x) = \begin{cases} x_- \,, & |x - x_-| < |x - x_+| \\ x_+ \,, & |x - x_-| > |x - x_+| \end{cases}$ (**zaokruživanje prema najbližem**);

  Ako je $|x - x_-| = |x - x_+|$, onda se uzme x_- ili x_+, već prema tome je li u x_- ili u x_+ najmanje značajan bit b_{p-1} nula.

Ako je $x < 0$ i $|x| > N_{\max}$ uzima se round$(x) = -\infty$, a ako je $x > N_{\max}$ uzima se round$(x) = +\infty$.

U IEEE standardu se po defaultu uzima četvrti način.

Uočimo da kod zaokruživanja prema najbližem vrijedi

$$|\text{round}\,(x) - x| \leq 2^{-p} \times 2^{e},$$

to jest greška zaokruživanja je najviše pola ulp-a.

Općenito vrijedi

$$\text{round}\,(x) = x\,(1+\delta)\,, \quad |\delta| \leq u = 2^{-p}$$

ako se koristi zaokruživanje prema najbližem, a $u = 2^{-p+1}$ za ostale načine zaokruživanja.

2.4 Korektno zaokruživanje kod računskih operacija

Jedna od najvažnijih značajki IEEE standarda jest zahtjev da se prilikom izvođenja osnovnih računskih operacija rezultat dobiva kao da je izračunat točno i zatim zaokružen.

Označimo redom s $\oplus, \ominus, \otimes$ i $\oslash$ operacije $+, -, \times$ i $/$ kako su stvarno implementirane u računalu. Također označimo s $\circ$ proizvoljni element skupa $\{+, -, \times, /\}$, a s $\odot$ proizvoljni element skupa $\{\oplus, \ominus, \otimes, \oslash\}$. Za IEEE standard vrijedi

$$x \odot y = fl\,(x \circ y) = \text{round}\,(x \circ y) = (x \circ y)\,(1+\delta)\,, \quad |\delta| \leq u.$$

No naglasimo da osim za ove četiri osnovne operacije standard mora vrijediti i za unarnu operaciju drugog korijena. Aritmetika koja zadovoljava ovaj uvijet ponekad se naziva i **aritmetika s korektnim zaokruživanjem**.

PRIMJER. Neka je $x = 1$, $y = 2^{-25}$ i $z = 1$. Ti brojevi su u BPT. Kako je $y = 1.0 \times 2^{-25}$, suma

$$x + y = 1.00000000000000000000000001$$

se ne može egzaktno implementirati u IEEE standardu s jednostrukim formatom. Zaokruživanje prema najmanjemu daje $x \oplus y = 1$, a vrijedi $(x \oplus y) \ominus z = 0$.
S druge strane, egzaktni rezultat je

$$(x + y) - z = 2^{-25}.$$

Pokazali smo da je

$$fl\left((x + y) - z\right) = 0$$

iako je

$$\text{round}\left((x + y) - z\right) = 2^{-25}.$$

Vidimo da za ovaj izraz ne vrijedi aksiom IEEE standarda! To posljedica činjenice da u računalnoj aritmetici ne vrijedi asocijativnost zbrajanja.

Ipak, vrijedi:

- $x \oplus y = y \oplus x$
- $x \otimes y = y \otimes x$
- $1 \otimes x = x$
- $x \oslash x = 1$
- $x \ominus y = 0 \Longrightarrow x = y$
- $x \ominus y = -\left(y \ominus x\right)$

2.5 Implementacija operacija u računalu

2.5.1 Zbrajanje

Napomenimo najprije da nećemo posebno razlikovati oduzimanje od zbrajanja jer je

$$x - y = x + (-y)\,.$$

Ako operandi nemaju isti eksponent, onda se po modulu manji operand napiše u obliku nenormaliziranog broja s eksponentom većeg. To znači da mu se mantisa pomakne u desno za onoliko binarnih mijesta kolika je bila razlika u eksponentima operanada. Nakon izvršenja operacije rezultat se svede na normalizirani oblik, pri čemu se prvo mantisa opet pomakne u lijevo ili u desno (po potrebi), zaokruži, i ako je potrebno, mantisa se opet pomakne lijevo ili desno.

PRIMJER. Promotrimo računanje razlike $x - y$ u jednostrukom formatu, pri čemu je

$$x = (1.0)_2 \times 2^0, \quad y = (1.111\cdots1)_2 \times 2^{-1}.$$

Izjednačavanjem eksponenata dobije se

$$(1.00000000000000000000000 \,|)_2 \times 2^0$$

$$-\,(0.11111111111111111111111 \mid 1) \times 2^0$$

$$\begin{aligned} &= (0.00000000000000000000000 \mid 1)_2 \times 2^0 \\ &= (1.00000000000000000000000 \mid 0)_2 \times 2^{-24}. \end{aligned}$$

Uočimo da smo za račun koristili jedan dodatni bit. Ovo je također i primjer kraćenja jer se skoro svi bitovi rezultata pokrate.
No katkad nije dovoljan jedan dodatni bit.

PRIMJER. Promotrimo računanje razlike $x - y$ u jednostrukom formatu, pri čemu je

$$x = (1.0)_2 \times 2^0, \quad y = (1.000 \cdots 01)_2 \times 2^{-24}.$$

Korištenjem čak 25 dodatnih bitova dobijemo

$$(1.00000000000000000000000 \,|)_2 \times 2^0$$

$$\begin{aligned} & - (0.00000000000000000000000 \mid 01000000000000000000000001)_2 \times 2^0 \\ = \; & (0.11111111111111111111111 \mid 10111111111111111111111)_2 \times 2^0 \\ = \; & (1.11111111111111111111111 \mid 01111111111111111111110)_2 \times 2^{-1} \\ = \; & (1.11111111111111111111111 \,|)_2 \times 2^{-1}. \end{aligned}$$

Uočimo da bismo pogrešan rezultat

$$(1.00000000000000000000000 \,|)_2 \times 2^0$$

dobili korištenjem bilo kojeg manjeg broja dodatnih bitova.

Ipak, u praksi je dovoljno koristiti tek tri dodatna bita, točnije dva **zaštitna** i jedan **zalijepljeni** bit. Zovemo ga zalijepljeni jer se on aktivira tek onda kada je potrebno pomaknuti mantisu za više od dva mjesta, a jednom kada se postavi više se ne mijenja. U našem primjeru bi to izgledalo ovako:

$$
\begin{aligned}
& (1.00000000000000000000000 \,|)_2 \times 2^0 \\
& \\
& -(0.00000000000000000000000 \mid 011)_2 \times 2^0 \\
= \; & (0.11111111111111111111111 \mid 101)_2 \times 2^0 \\
= \; & (1.11111111111111111111111 \mid 01)_2 \times 2^{-1} \\
= \; & (1.11111111111111111111111 \,|)_2 \times 2^{-1}.
\end{aligned}
$$

2.5.2 Množenje i dijeljenje

Ove dvije operacije u BSPT ne zahtijevaju poravnavanje eksponenata. Ako je

$$x = m_x \times 2^{e_x}, \qquad y = m_y \times 2^{e_y},$$

onda je

$$\begin{aligned} z &= x \cdot y = (m_x \cdot m_y) \times 2^{e_x + e_y}, \\ w &= x/y = (m_x/m_y) \times 2^{e_x - e_y}, \end{aligned}$$

nakon čega se primjenjuju normalizacija i pravilno zaokruživanje.
Današnji dizajneri čipova su postigli to da uz dovoljnu rezervu memorije množenje bude gotovo jednako brzo kao zbrajanje, no s dijeljenjem to nije slučaj.
Uočimo da zbog $1 \leq m_x < 2$ i $1 \leq m_y < 2$ vrijedi

$$1 \leq m_z < 4.$$

Dakle, lijevo od binarne točke u binarnoj reprezentaciji mantise m_z mogu biti samo kombinacije brojeva 11, 10 ili 01, stoga se mantisa može pomaknuti najviše jedno mjesto udesno uz istodobno povečanje eksponenta za 1.
Analogno za dijeljenje vrijedi

$$\frac{1}{2} < m_w < 2,$$

pa pomak može biti jedno mjesto ulijevo ili udesno.

2.6 Konverzija formata

IEEE standard zahtijeva podršku za prevođenje između raznih vrsta formata. Tu spada:

- Konverzija iz kraćega u dulji format koja mora biti egzaktna.
- Konverzija iz duljega u kraći format koja zahtijeva korektno zaokruživanje.
- Konverzija iz BSPT u cjelobrojni format koja zahtijeva zaokruživanje na najbliži cijeli broj. Ako je početni broj cijeli, onda rezultat mora biti taj isti broj (osim ako nema reprezentaciju u cjelobrojnom formatu!).
- Konverzija iz cjelobrojnog formata u BSPT koja može zahtijevati zaokruživanje.
- Konverzija iz decimalnog u binarni sustav i obratno.

2.7 Izuzeci (koji prelaze u NaN)

- $a/0$
- $0 \times \infty$
- ∞/∞
- $\infty - \infty$

2.8 Prekoračenje i potkoračenje

O **prekoračenju** se govori kada je egzaktan rezultat neke operacije konačan broj, ali po modulu veći od $N_{\max}$. O **potkoračenju** se govori kada je egzaktan rezultat neke operacije broj različit od nule, ali po modulu manji od $N_{\min}$. Postupamo na sljedeći način:

- ako je nastupilo prekoračenje zaokružimo dani broj na $\pm\infty$ ili $\pm N_{\max}$ ovisno o tipu zaokruživanja;
- ako je nastupilo potkoračenje dani broj pravilno zaokružimo (rezultat je subnormalni broj, ± 0 ili $\pm N_{\min}$).

Ovaj drugi postupak je najkontraverzniji dio IEEE standarda jer daje manju relativnu točnost rezultata (to manju što je potkoračenje veće), no upravo ono dovodi do toga da u BSPT vrijedi implikacija

$$x \ominus y = 0 \Longrightarrow x = y.$$

PRIMJER. Promotrimo računanje razlike $x - y$ u jednostrukom formatu, pri čemu je

$$x = 25 \times 2^{-131}, \quad y = 0.1 \times 2^{-123}.$$

Izjednačavanjem eksponenata dobije se

$$\begin{aligned} & (1.10011001100110011001101)_2 \times 2^{-127} \\ & - (1.10010000000000000000000) \times 2^{-127} \\ = \ & (0.00001001100110011001101)_2 \times 2^{-127} \\ = \ & (1.10011001100110011010000)_2 \times 2^{-132}. \end{aligned}$$

Vidimo da je rezultat subnormalni broj koji bi bez potkoračenja bio nula!