2 FLOATING-POINT ZAPIS DECIMALNIH BROJEVA

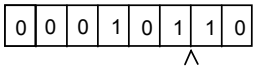
Decimalni brojevi se mogu zapisivati na dva načina: *sa nepomičnim zarezom* i *sa pomičnim zarezom*. U slučaju zapisa sa nepomičnim zarezom, pozicija decimalnog zareza se podrazumijeva na fiksnom (tačno odredjenom) mjestu. Brojevi se tumače u skladu sa usvojenom pozicijom decimalnog zareza. Na primjer, ako se podrazumijeva da 8-mobitni zapis u digitalnom sistemu ima dva decimalna mjesta, onda se struktura zapisa označenog decimalnog broja grafički može prikazati na sljedeći način:

Broj bitova predvidjen za zapisivanje cijelog i za zapisivanje decimalnog dijela broja je ograničen. To znači da je ograničen i dozvoljeni opseg brojeva koji se mogu zapisati u ovom zapisu. Ako je za potrebu zapisivanja cjelobrojnog dijela nekog decimalnog broja neophodno više bitova nego što ih za tu svrhu ima u usvojenom formatu zapisa, dolazi do prekoračenja dozvoljenog opsega memorijske riječi.

U cilju ublažavanja ovog problema, potrebno je proširiti dozvoljeni opseg brojeva, povećavanjem broja bitova predvidjenih za zapisivanje cjelobrojnog dijela decimalnog broja. Medjutim, ovo je (u slučaju unaprijed utvrdjene fiksne dužine memorijske riječi) moguće postići samo na račun smanjenja broja bitova predvidjenih za predstavljanje decimalnog dijela broja. Na taj način se, sa druge strane, smanjuje tačnost zapisivanih decimalnih brojeva, povećavajući potrebu za odsijecanjem (ili zaokruživanjem) onih decimalnih bitova koje je nemoguće zapisati na mjestima predvidjenim za ovu namjenu. Drugim riječima, nemoguće je, u okviru unaprijed zadate dužine memorijske riječi, istovremeno povećati opseg i tačnost zapisivanih decimalnih brojeva. Odnosno, proširivanje opsega je moguće postići samo na račun umanjenja tačnosti i obrnuto (tačnost zapisivanih decimalnih brojeva se povećava na račun sužavanja dozvoljenog opsega zapisivanih decimalnih brojeva).

Primjer 2-1: Predstaviti binarni broj x=+101.1011 u zapisu sa nepomičnim zarezom, ako je za cjelobrojni dio rezervisano 5 bitova, a za decimalni dio 2 bita.

Rješenje: Binarni broj x=+101.1011 se može, u opisanom formatu, zapisati na sljedeći način:

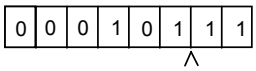


Zapisani broj, prilagodjen opisanom formatu zapisivanja, se obično u literaturi označava sa $Q[x]$ (tzv. „kvantizirano x“). Primijetimo da se on razlikuje od originalnog broja x, zato što su, opisanim formatom, za zapisivanje decimalnog dijela broja predvidjena dva mjesta, a broj x ima četiri decimale. Greška napravljena prilikom zapisivanja (tzv. *greška kvantizacije*) iznosi:

$$\varepsilon = x - Q[x]. \tag{2-1}$$

U našem primjeru, greška kvantizacije iznosi $\varepsilon = 0.0011_{(2)}$.

Primijetimo da smo u cilju zapisivanja broja *x* u prethodnom primjeru izvršili *odsijecanje* onih decimala koje nije bilo moguće smjestiti u usvojenom formatu zapisivanja. Prilikom zapisivanja binarnog broja može se primijeniti i metod *zaokruživanja* na mjestu najmanje značajnog bita. U tom slučaju, *LSB* uzima vrijednost jedinice ukoliko je prva cifra odsječenog dijela broja jedinica, odnosno vrijednost nule ukoliko je prva cifra odsječenog dijela broja 0. Primjenjujući metod zaokruživanja u prethodnom primjeru, posmatrani broj bi bio zapisan na sljedeći način:



U posmatranom primjeru, greška kvantizacije (nastala zaokruživanjem) iznosi $\varepsilon = -0.0001_{(2)}$, što znači da je ona po apsolutnoj vrijednosti manja od greške kvantizacije nastale odsijecanjem.

U opštem slučaju, ako imamo na raspolaganju *m* bita za predstavljanje decimalnog dijela broja, greška kvantizacije (ε) koju pravimo *odsijecanjem* nižih decimalnih mjesta od mjesta težine 2^{-m} nalazi se u intervalu:

$$0 \le \varepsilon < 2^{-m}. \tag{2-2}$$

Primjenjujući *zaokruživanje*, greška kvantizacije se nalazi u intervalu:

$$-\frac{1}{2}2^{-m} \le \varepsilon < \frac{1}{2}2^{-m}. \tag{2-3}$$

Primjer 2-2: Posmatrajmo zapis označenih decimalnih binarnih brojeva sa nepomičnim zarezom. Pretpostavimo da za smještanje decimalnog dijela broja imamo na raspolaganju 4 bita. U kojim granicama se kreće greška kvantizacije ako:

a) primijenimo metod odsijecanja nižih decimalnih mjesta;

b) primijenimo metod zaokruživanja nižih decimalnih mjesta.

Rješenje: Ukoliko se primijene izrazi iz jednačina (2-2) i (2-3) jednostavno pronalazimo granice u kojima se nalaze greške kvantizacije:

a) Za slučaj odsijecanja:

$$0 \le \varepsilon < 2^{-4}$$

$$0 \le \varepsilon < 0.0625.$$

b) Za slučaj zaokruživanja:

$$-\frac{1}{2}2^{-4} \le \varepsilon < \frac{1}{2}2^{-4}$$

$$-0.03125 < \varepsilon < 0.03215.$$

Primjer 2-3: Posmatrajmo binarne brojeve *x* = +101.1011 i *y* = +10.011 u osmobitnom zapisu označenih decimalnih brojeva sa nepomičnim zarezom i sa dva decimalna mjesta. Izračunati grešku kvantizacije koja se pravi izvršavanjem aritmetičkih operacija:

a) sabiranja, $r = x + y$,

b) množenja, $r = x \cdot y$.

Rješenje: Opisanim formatom zapisivanja decimalnih brojeva, kvantizirane vrijednosti zadatih brojeva *x* i *y*, u slučaju odsijecanja nižih decimalnih mjesta, su:

$$Q[x] = 000101.10_{(2)}$$

$$Q[y] = 000010.01_{(2)}$$

i njima odgovarajuće greške kvantizacije:

$$\varepsilon_x = 0.0011_{(2)}$$
$$\varepsilon_y = 0.001_{(2)}.$$

a) Kvantizirana vrijednost rezultata operacije sabiranja je:

$$Q[r] = Q[x] + Q[y] = 000111.11_{(2)}$$

dok je tačna (nekvantizirana) vrijednost istog rezultata:

$$r = x + y = 001000.0001_{(2)}.$$

Greška, napravljena kvantizacijom rezultata operacije sabiranja (ε_a), je:

$$\varepsilon_a = r - Q[r] = 0.0101_{(2)}.$$

Primijetimo da je $\varepsilon_a = \varepsilon_x + \varepsilon_y = 0.0011_{(2)} + 0.001_{(2)} = 0.0101_{(2)}$. Dobijeni rezultat je očekivan. Naime, u slučaju zapisa decimalnih brojeva sa nepomičnim zarezom, greška kvantizacije definisana izrazom (2-1) je aditivna ($x = Q[x] + \varepsilon$, tj. originalna vrijednost *x* se dobija sabiranjem (dodavanjem) greške kvantizacije na kvantiziranu vrijednost $Q[x]$). Takodje, pošto izvršavamo operaciju sabiranja, zadržavamo aditivnost grešaka napravljenih kvantizacijom:

$$\begin{aligned} r = x + y &= Q[x] + \varepsilon_x + Q[y] + \varepsilon_y \\ &= Q[x] + Q[y] + \varepsilon_x + \varepsilon_y \\ &= Q[r] + \varepsilon_a. \end{aligned}$$

U opštem slučaju, sabiranjem *N* decimalnih brojeva predstavljenih zapisom sa nepomičnim zarezom, rezultantna greška kvantizacije jednaka je zbiru svih pojedinačnih grešaka, nastalih kvantizacijom odgovarajućih sabiraka. Maksimalna moguća greška kvantizacije, koja može nastati tom prilikom, jednaka je zbiru maksimalnih mogućih grešaka, nastalih kvantizcijom pojedinih sabiraka. O ovoj pojavi posebno treba voditi računa u slučaju velikog broja sabiraka *N*, pošto tada maksimalna moguća greška kvantizacije može uzeti značajnu vrijednost i time veoma uticati na tačnost rezultata predstavljenih u memorijskim riječima ograničene dužine.

b) U memorijskim riječima konačne dužine po pravilu se smještaju kvantizirane vrijednosti operanada *x* i *y*, odnosno *Q*[*x*] i *Q*[*y*]. Tačne vrijednosti operanada se mogu smjestiti samo u slučaju dovoljne dužine memorijske riječi, što u našem primjeru nije zadovoljeno. Takodje, množenjem kvantiziranih vrijednosti *Q*[*x*] i *Q*[*y*] po pravilu se dobija brojna vrijednost koja se ne može smjestiti u memorijsku riječ zadatog formata zapisivanja. To znači da je proizvod $Q[x] \cdot Q[y] = 0001100.0110_{(2)}$ neophodno kvantizirati kako bi se mogao smjestiti u memorijsku riječ zadatog formata zapisivanja.

Kvantizirana vrijednost proizvoda $Q[x] \cdot Q[y]$ iznosi:

$$Q[r] = Q[Q[x] \cdot Q[y]] = 001100.01_{(2)} = 12.25_{(10)}.$$

Tačna (nekvantizirana) vrijednost istog rezultata je:

$$r = x \cdot y = 1101.1000001_{(2)} = 13.5078125_{(10)}.$$

Greška, napravljena kvantizacijom rezultata operacije množenja (ε_m), je:

$$\varepsilon_m = r - Q[r] = 1.0100001_{(2)} = 1.2578125_{(10)}.$$

Drugim riječima, greška kvantizacije, napravljena izvršavanjem operacije množenja

1. Iznosi čak oko 10% vrijednosti dobijenog rezultata množenja,
2. Značajno je veća (po apsolutnoj vrijednosti) od greške kvantizacije, napravljene izvršavanjem operacije sabiranja.

Objasnimo dobijeni rezultat. Izvršavanjem operacije množenja dobija se:

$$\begin{aligned} r = x \cdot y &= (Q[x] + \varepsilon_x) \cdot (Q[y] + \varepsilon_y) \\ &= Q[x] \cdot Q[y] + Q[x] \cdot \varepsilon_y + Q[y] \cdot \varepsilon_x + \varepsilon_x \cdot \varepsilon_y \\ &= Q[r] + \varepsilon_m. \end{aligned}$$

Greška napravljena kvantizacijom iznosi:

$$\varepsilon_m = Q[x] \cdot \varepsilon_y + Q[y] \cdot \varepsilon_x + \varepsilon_x \cdot \varepsilon_y. \tag{2-4}$$

Ona ne potiče samo od proizvoda grešaka uzrokovanih kvantizacijom pojedinih multiplikanata ($\varepsilon_x \cdot \varepsilon_y$) već na njenu veličinu presudno utiču sabirci $Q[x] \cdot \varepsilon_y$ i $Q[y] \cdot \varepsilon_x$. Njihove vrijednosti su proporcionalne kvantiziranim (približno stvarnim) vijednostima multipilikanata $Q[x]$ i $Q[y]$. U slučaju numeričkih vrijednosti multiplikanata većih od 1, sabirci $Q[x] \cdot \varepsilon_y$ i $Q[y] \cdot \varepsilon_x$ mogu postati značajni, uzrokujući time značajnu grešku kvantizacije ε_m. U tom slučaju se može zanemariti veličina sabirka $\varepsilon_x \cdot \varepsilon_y$, tako da greška kvantizacije (2-4) postaje:

$$\varepsilon_m = Q[x] \cdot \varepsilon_y + Q[y] \cdot \varepsilon_x.$$

Relativna greška kvantizacije, napravljena izvršavanjem računske operacije množenja i uzrokovana uticajem ograničene dužine memorijske riječi u slučaju formata zapisivanja decimalnih brojeva sa nepomičnim zarezom, iznosi:

$$\frac{Q[x] \cdot \varepsilon_y + Q[y] \cdot \varepsilon_x}{Q[x]Q[y]} = \frac{\varepsilon_y}{Q[y]} + \frac{\varepsilon_x}{Q[x]} \cong \frac{\varepsilon_y}{y} + \frac{\varepsilon_x}{x}.$$

Prema tome, u slučaju implementacije računske operacije množenja, relativne greške, koje su napravljene kvantizacijom, se sabiraju. Može se pokazati da ovaj zaključak važi i u slučaju množenja *N* činilaca, uvećavajući sa svakim novim činiocem ukupnu relativnu grešku uzrokovanu ograničenom dužinom riječi.

Slično kao u slučaju zapisivanja cijelih brojeva, ukoliko se prilikom izvršavanja aritmetičkih operacija dobije brojna vrijednost koja se ne može smjestiti u okviru raspoloživog formata zapisivanja, računar detektuje da je došlo do prekoračenja (*overflow*). U tom slučaju se, uglavnom, obustavlja dalje procesuiranje rezultata. U našem primjeru sa osmobitnim formatom zapisa, od čega se pet bitova koristi za predstavljanje cjelobrojnog dijela broja, prekoračenje se pojavljuje ako pokušamo zapisati broj veći od 011111.11 ili manji od 100000.00 (negativni brojevi se zapisuju u svom dvojnom komplementu). Primijetimo da su binarne kombinacije za maksimalnu i minimalnu vrijednost, koje je moguće predstaviti u ovom formatu zapisivanja, iste kao u slučaju zapisa označenih cijelih brojeva, samo je različit način tumačenja zapisanih brojeva.

Na kraju, iz prethodnih primjera možemo primijetiti da opisanim formatom zapisivanjaja raspoloživi memorijski prostor nije optimalno iskorišćen. Na primjer, za smještanje 3 cjelobrojne binarne cifre u primjeru 2-1 raspolagali smo sa 5 mjesta, dok je broj raspoloživih decimalnih mjesta bio nedovoljan za smještanje decimalnih cifara binarnog broja. Ovaj nedostatak se može bitno ublažiti primjenom zapisa sa pomičnim zarezom.

2.1 ZAPIS DECIMALNIH BROJEVA SA POMIČNIM ZAREZOM (FLOATING-POINT IEEE 754 ZAPIS)

Nedostatak uočen prilikom zapisivanja decimalnih brojeva sa nepomičnim zarezom značajno se može ublažiti izbjegavanjem zapisivanja cifara koje ne doprinose vrijednosti broja (na primjer, vodeće dvije nule u primjeru 2-1). Umjesto njih u posmatranoj riječi mogu biti zapisane decimale koje doprinose tačnosti zapisivanog broja (na primjer, decimale na težinskim mjestima -3 i -4 iz primjera 2-1). Na ovim principima se zasniva zapis decimalnih brojeva sa pomičnim zarezom. Njime se značajno

povećava opseg zapisivanih brojeva u poredjenju sa zapisom sa nepomičnim zarezom, ali i tačnost zapisivanih brojeva.

Da bi objasnili ovaj način zapisivanja brojeva, prisjetimo se zapisivanja velikih dekadnih brojeva pomoću eksponencijalnog zapisa (tzv. *naučna notacija*). Na primjer, dekadni broj 120000000000 se može zapisati kao 1.2×10^{11}, dok dekadni broj 0.0000000378 zapisujemo kao 3.78×10^{-8}. Prilikom predstavljanja prvog navedenog broja u eksponencijalnom zapisu, umjesto 12 cifara, zahtijeva se zapisivanje dvije najznačajnije cifre različite od nule (1.2) i vrijednost eksponenta (11), odnosno samo 4 značajne cifre. Isti broj cifara je dovoljan za zapisivanje broja i u drugom primjeru. Tada se zapisuju cifre 3.78 (najznačajnije cifre različite od 0) i eksponent 8.

Vodeći (decimalni) dio broja (u prvom slučaju 1.2, a u drugom 3.78), predstavljen u formatu zapisivanja sa nepomičnim zarezom, naziva se *mantisa*. Eksponent desetke, predstavljen označenim cijelim brojem koji odgovara stvarnom položaju decimalnog zareza, naziva se jednostavno *eksponent*.

Slična logika se primjenjuje i prilikom zapisa binarnih brojeva.

Primjer 2-4: Zapisati u eksponencijalnom obliku binarne brojeve:

a) 11010000000;

b) 0.000000001001.

Rješenje: Eksponencijalni zapisi binarnih brojeva su sljedeći:

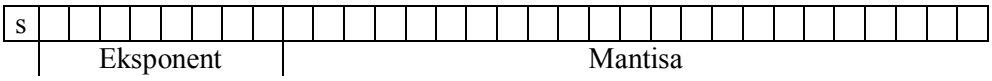
a) $11010000000 = 1.101\times2^{10}$;

b) $0.000000001001 = 1.001\times2^{-9}$.

Prethodno brojevi su zapisani ne samo u naučnoj notaciji, već i u normalizovanoj naučnoj notaciji, koja podrazumijeva položaj decimalnog zareza u mantisi neposredno iza prve nenulte cifre (u slučaju binarnih brojeva ta cifra mora biti 1). Preciznije, brojna vrijednost 11010000000 može biti zapisana i na sljedeće načine: $11010000000 = 0.1101\times2^{11}$, kao i $11010000000 = 11.01\times2^{9}$. Na oba načina broj je predstavljen u naučnoj, ali ne i u normalizovanoj notaciji (u prvom slučaju decimalni zarez se nalazi ispred prve nenulte cifre, dok se u drugom slučaju decimalni zarez nalazi iza druge nenulte cifre).

2.1.1 Floating-point zapis decimalnih brojeva uskladjena sa standardom IEEE 754

Jedna od mogućih struktura zapisa binarnog broja sa pomičnim zarezom grafički se može, u slučaju 32-bitne memorijske riječi, prikazati na sljedeći način:



gdje je s – bit znaka mantise (u literaturi poznat po nazivu *sign bit*).

Prikazani raspored polja i broj bitova upotrijebljen za reprezentaciju pojedinih polja odgovara zapisu sa pomičnim zarezom, uskladjenim sa *IEEE 754 standardom*. Ovaj standard za zapis mantise upotrebljava 23 bita, eksponent se zapisuje sa 8 bita, a jedan bit se koristi za zapisivanje znaka broja, koji odgovara znaku mantise. Naravno, podrazumijeva se da je osnova eksponenta jednaka 2. Ovim načinom raspodjele bitova memorijske riječi (koju pretpostavljamo da je fiksne dužine i iznosi 32 bita) nastoji se postići kompromis izmedju dužine exponenta i mantise. Naime, povećavanjem jednog od njih (polja exponenta ili polja mantise) smanjuje dužinu drugog polja. Sa druge strane, eksponentom je odredjen opseg zapisivanih brojeva, dok se mantisom odredjuje tačnost njihovog zapisivanja. Stoga se želi što veće broj bitova za oba posmatrana polja (polje eksponenta i polje mantise), a to je nemoguće postići kod memorijske riječi fiksne dužine. Zbog toga se pribjegava pravljenju kompromisa.

Bit znaka, označen sa *s*, odredjuje znak floating-point broja. Ukoliko je bit znaka jednak nuli (s=0) u pitanju je pozitivan broj, a ukoliko je bit znaka jednak jedinici (s=1) radi se o negativnom broju.

Kao što je već rečeno, mantisa se zapisuje u tzv. **normalizovanom** obliku, što znači da se decimalni broj transformiše u oblik koji ima samo jednu cjelobrojnu nenultu cifru. Kod binarnih brojeva, ta cifra je uvijek jedinica. Zato se ta cifra ne mora zapisivati u formatu broja (implicitno se zapisuje – podrazumijeva se da postoji) i time se dobija još jedno mjesto za zapisivanje cifara mantise (povećava se preciznost zapisanog broja). Drugim riječima, predstavljenim floating-point zapisom generalno se zapisuju decimalni brojevi čija vrijednost se može predstaviti na sljedeći način:

$$(-1)^s \times (1+F) \times 2^E \tag{2-5}$$

gdje je sa E označen eksponent, označeni cijeli 8-mobitni broj zajedno sa znakom broja, sa F brojna vrijednost 23-bitnog decimalnog dijela mantise, dok je sa s označen bit znaka.

Postavlja se pitanje: *zašto normalizovani floating-point zapis?* Razlozi za to su sljedeći:

1. Olakšava zamjenu (prevodjenje) broja u floating-pont zapis,
2. Pojednostavljuje algoritme upotrebljavane u floating-point aritmetici,
3. Povećana tačnost brojeva zapisanih u floating-point zapisu (ne zapisuju se vodeće nule koje ne doprinose vrijednosti broja, ali se stoga zapisuju realni digiti sa desne strane binarnog zareza, koji doprinose i vrijednosti zapisanog broja i njegovoj tačnosti).

Eksponent može uzeti pozitivnu ili negativnu vrijednost, pa se zato u polju predvidjenom za njegov zapis mora obezbijediti mogućnost zapisivanja i pozitivnih i negativnih cijelih brojeva. Ovo bi se, naravno, moglo obezbijediti zapisom označenih cijelih brojeva u osmobitnoj riječi, koji pretpostavlja zapis negativnih brojnih vrijednosti u svom dvojnom komplementu. U ovom slučaju možemo odmah uočiti da bi 8-mobitni eksponent uzimao dekadne brojne vrijednosti iz opsega cijelih brojeva od –128 do 127. Drugim riječima, predstavljenim floating-point zapisom bi se omogućila reprezentacija decimalnih brojeva približno iz opsega brojeva od -2×2^{127} do 2×2^{127}, kao i veoma malih brojeva približno iz opsega brojeva od -2×2^{-128} do 2×2^{-128}. Prilikom izvodjenja opsega bojeva uzeta je maksimalna vrijednost mantise veličine 2, iako precizno rečeno, njena maksimalna vrijednost iznosi $2_{(10)}-(2^{-24})_{(10)}\cong2_{(10)}$.

Primijetimo da je opseg brojeva koji se omogućava prezentiranim floating-point zapisom veoma širok. Medjutim, ipak ne dovoljno širok da bi se prevazišla mogućnost pojave efekata over/underflow-a prilikom implementacije operacija binarnog sabiranja i oduzimanja (*Overflow* nastaje kada je exponent pozitivan i veći od maksimalnog pozitivnog cijelog broja koji može da se zapiše u polju predvidjenom za zapisivanje eksponenta (eksponent veći od 127); *Underflow* nastaje kada je ekponent negativan i manji od minimalnog negativnog polja koji može da se zapiše u polju predvidjenom za zapisivanje eksponenta (eksponent manji od –128)).

U cilju potiskivanja (gotovo prevazilaženja) efekta over/underflow-a uvodi se floating-point zapis decimalnih brojeva sa *dvostrukom preciznošću*, koji upotrebljava dvije 32-bitne memorijske riječi (64 bita), te za zapisivanje eksponenta koristi 11 bitova, a za zapisivanje mantise 52 bita. Ovim zapisom se istovremeno povećava opseg, ali i preciznosti zapisivanih decimalnih brojeva. Sada se prezentirani 32-bitni floating point zapis naziva zapisom sa *jednostrukom preciznošću*.

Medjutim, zapisom negativnih cjelobrojnih vrijednosti eksponenta u dvojnom komplementu, bilo u jednostukoj ili u dvostrukoj preciznosti, javlja se problem prilikom njihovog sortiranja, odnosno uporedjivanja dvije različite vrijednosti. Naime, u takvom zapisu broj koji ima jedinicu na višem težinskom mjestu u odnosu na svog konkurenta ne bi bio nužno i veći broj. Konačno, svi negativni brojevi imaju jedinicu na najvišem bitu (bitu znaka), pa bi prilikom uporedjivanja na osnovu težina bitova (vizuelno) izgledali većim od pozitivnih brojeva.

Sa druge strane, IEEE 754 standard predvidja zapisivanje vrijednosti eksponenta ispred zapisane apsolutne vrijednosti mantise (vidi ilustraciju zapisa sa pomičnim zarezom), upravo s razlogom sticanja ispravne vizuelne predstave o veličini zapisanog broja. Naime, veličinu broja zapisanog u eksponencijalnoj notaciji dominantno odredjuje veličina njegovog eksponenta. To znači

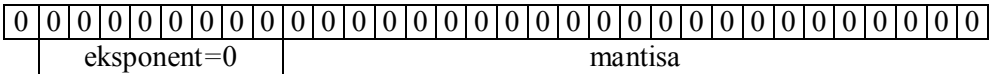
da bi zapisivanjem eksponenta u zapisu sa dvojnim komplementom bila obesmišljena osnovna namjera IEEE 754 standarda: *jednostavno uporedjivanje (odnosno sortiranje) zapisanih brojnih vrijednosti*.

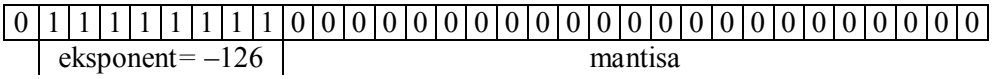
Primjer 2-4: Pretpostavljajući zapis eksponenta floating-point broja u formi označenog cijelog broja, čije su negativne vrijednosti predstavljene u dvojnom komplementu, predstaviti brojne veličine $1_{(10)}$, $0.5_{(10)}$ i $2_{(10)}$ u floating-point IEEE 754 zapisu.

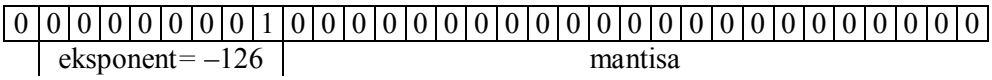
Rješenje: U normalizovanoj formi i jednostrukoj preciznosti zadati brojevi bi slijedili sljedeća pravila zapisivanja:

$$1_{(10)}=1.0\times2^{0}=(-1)^{0}\times(1+0.00\ldots0)\times2^{0},$$
$$0.5_{(10)}=1.0\times2^{-1}=(-1)^{0}\times(1+0.00\ldots0)\times2^{-1},$$
$$2_{(10)}=1.0\times2^{+1}=(-1)^{0}\times(1+0.00\ldots0)\times2^{+1}.$$

odnosno bili zapisani redom na sljedeći način:



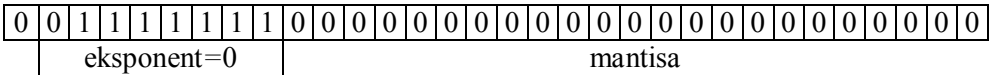


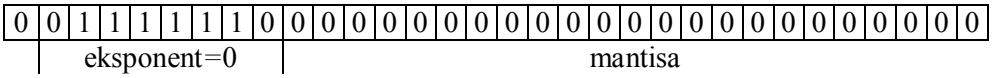


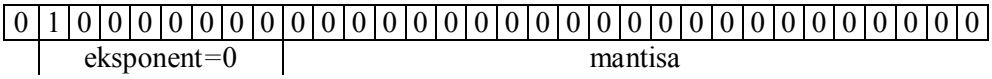
Primijetimo da je $1_{(10)}$ u ovom slučaju zapisana sa nulama u svim bitovima predvidjenim za zapis floating-point broja. Sa druge strane, u svim brojnim sistemima i svim reprezentacijama, uobičajeno je da se $0_{(10)}$ zapisuje sa nulama u svim bitovima reprezentacije, naravno bez mogućnosti implicintnog pamćenja vodeće jedinice u slučaju floating-point reprezentacije broja. Drugim riječima, predstavljeni zapis brojnih veličina $0_{(10)}$ i $1_{(10)}$ ne bi bio jednoznačan, što je neodrživo.

Takodje, posmatrajući samo binarne floating-point reprezentacije brojeva $0.5_{(10)}$ i $2_{(10)}$ zaključilo bi se da je $0.5_{(10)} > 2_{(10)}$ (jedinica u drugom po redu zapisu se nalazi na bitu težine 30, dok se u trećem po redu zapisu nalazi tek na bitu težine 23), što jednostavno ne odgovara istini.

Prevazilaženje nedostatka uočenih u prethodnom primjeru se, u slučaju IEEE 754 standarda, postiže dodavanjem konstantne vrijednosti na stvarnu vrijednost eksponenta (tzv. ponderisanje eksponenta). Dakle, u polje predvidjeno za zapis eksponenta se upisuje zbir vrijednosti eksponenta i konstantne vrijednosti (tzv. *bias*-a). Kod zapisa sa tzv. jednostrukom preciznošću ova konstanta iznosi 127, dok vrijednost *bias*-a u slučaju dvostruke preciznosti iznosi 1023. Dakle, ako je vrijednost eksponenta nula, u slučaju jednostruke preciznosti, u polje predvidjeno za zapis eksponenta se upisuje 127 (0+127=127). Ako je vrijednost eksponenta, na primjer, 12, u polje predvidjeno za zapis eksponenta se upisuje 139 (12+127=139). Slično tome, ako je vrijednost eksponenta –55, u polje predvidjeno za zapis eksponenta se upisuje 72 (–55+127=72). Sa druge strane, ako je u polju predvidjenom za zapis eksponenta upisana, na primjer, vrijednost 210, to znači da je stvarna vrijednost eksponenta zapravo 83 (210–127). Slijedeći ponderisanu (biased) notaciju zapisivanja eksponenta, brojevi $1_{(10)}$, $0.5_{(10)}$ i $2_{(10)}$, posmatrani u primjeru 2-4 se zapisuje na sljedeći način:







pošto su u biased notaciji predstavljeni sa:

$$1_{(10)}=1.0\times2^{0+127}=(-1)^0\times(1+0.00\ldots0)\times2^{127},$$
$$0.5_{(10)}=1.0\times2^{-1+127}=(-1)^0\times(1+0.00\ldots0)\times2^{126},$$
$$2_{(10)}=1.0\times2^{+1+127}=(-1)^0\times(1+0.00\ldots0)\times2^{128},$$

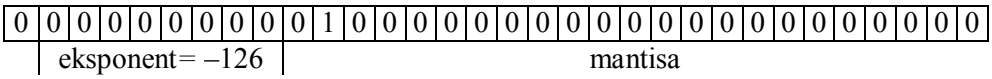
čime se otklanjaju ranije uočeni nedostaci.

Ipak, ponderisani (biased) zapis brojeva se suočava i sa nekim problemima. Jedan od njih je zapisivanje nule. Pošto se podrazumijeva da je cjelobrojni dio mantise binarnog broja koji se zapisuje u IEEE 754 formatu uvijek jedinica, nulu nije moguće na taj način zapisati. Zato se nula tretira kao specifična vrijednost i zapisuje se tako što se za nju rezerviše posebna binarna kombinacija: *na svim binarnim mjestima predvidjenim za zapis eksponenta i na svim binarnim mjestima predvidjenim za zapis mantise se upisuju nule*. Na ovaj način se mogu zapisati dvije nule: pozitivna i negativna – zavisno od vrijednosti bita znaka, kao što je prikazano u Tabeli 2-1.

Kao i kod drugih normalizovanih zapisa, i u ovom slučaju postoji relativno veliki raspon izmedju nule i najbližeg broja koji je moguće prikazati. Brojevi iz ovog raspona se ipak mogu prikazati zapisom sa pomičnim zarezom, ali samo koristeći tzv. *denormalizovani* zapis. Ako su na svim binarnim mjestima predvidjenim za zapis eksponenta nule, a na mjestima predvidjenim za zapis mantise se nalazi nenulta vrijednost, onda je broj zapisan na ovaj način predstavljen u *denormalizovanom* zapisu (broj nema podrazumijevanu jedinicu ispred decimalnog zareza). U tom slučaju demormalizovanog zapisa broja podrazumijeva se da je vrijednost eksponenta –126 (odnosno –1022 kod zapisa u dvostrukoj tačnosti). Dakle, ovim zapisom je predstavljen broj vrijednosti $(-1)^s\times0.f\times2^{-126}$, gdje je s bit znaka, a f je mantisa[1], kao što je prikazano u Tabeli 2-1. Uz sve ovo, važno je primjetiti i zabilježiti razliku izmedju *denormalizovanog* i *nenormalizovanog* zapisa. Svaki denormalizovani zapis broja ima samo jednu reprezentaciju (u kojoj je eksponent –126), dok je beskonačno mnogo nenormalizovanih reprezentacija broja (sa eksponentima proizvoljne veličine).

Primjer 2-5: Predstaviti broj 2^{-128} u binarnom obliku i zapisu sa pomičnim zarezom.

Rješenje: U normalizovanoj formi (u jednostrukoj tačnosti) ovaj broj nije moguće zapisati, jer je najmanja moguća vrijednost eksponenta –126. Medjutim, isti broj se može zapisati u denormalizovanom obliku kao: 0.01×2^{-126}. To znači da zapis u IEEE 754 formatu izgleda ovako:



Kod IEEE 754 standarda predvidjena je i mogućnost zapisivanja beskonačne vrijednosti (∞). To je pogodno da bi se omogućio nastavak računanja u slučajevima kada dodje do prekoračenja (*overflow*), a kada je ipak moguće nastaviti sa računanjem u cilju dobijanja adekvatnog rezultata. Beskonačna vrijednost se označava sa svim jedinicama u polju predvidjenom za zapisivanje eksponenta i svim nulama u polju predvidjenom za zapisivanje mantise. Bitom znaka se definiše pozitivna (s=0) odnosno negativna (s=1) beskonačnost, kao što je prikazano u Tabeli 2-1.

Konačno, u ovom formatu se mogu zapisati i tzv. *NaN* (*Not a Number*) vrijednosti. To su izrazi koji predstavljaju neodredjene vrijednosti (npr. 0/0, ∞–∞, ∞/∞, ±∞×0, ...). Oni se označavaju sa svim jedinicama u polju predvidjenom za zapisivanje eksponenta i nenultom vrijednosti u polju predvidjenom za zapisivanje mantise.

---

[1] Kod zapisa u dvostrukoj tačnosti denormalizovani brojevi su oblika $(-1)^s\times0.f\times2^{-1022}$.

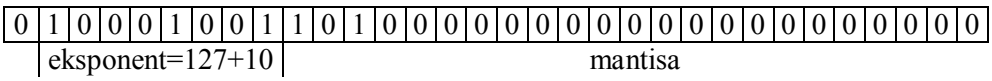
1-struka preciznost		2-struka preciznost		Brojna vrijednost
Biased Exp	F	Biased Exp	F	
0	0	0	0	0
0	≠0	0	≠0	± Denormalizovani broj $(-1)^S \times 0.f \times 2^{-126}$
1-254	bilo što	1-2046	bilo što	± Floating-point broj
255	0	2047	0	±∞
255	≠0	2047	≠0	NaN (0/0,0×∞, ∞–∞, ...)

Tabela 2-1. Brojne vrijednosti reprezentovane u floating-point IEEE 754 zapisu, u jednostrukoj i dvostrukoj preciznosti.

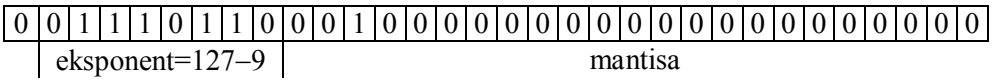
Primjer 2-6: Zapisati brojeve iz primjera 2-4 u strukturi zapisa usaglašenim sa IEEE 754 standardom.

Rješenje: Zapis brojeva izgleda ovako:

a) $11010000000 = 1.101 \times 2^{10}$;



b) $0.000000001001 = 1.001 \times 2^{-9}$.



Ukoliko vrijednost eksponenta ne može da se zapiše sa predvidjenim brojem bitova, dolazi do prekoračenja dozvoljenog opsega memorijske riječi usaglašene sa eksponencijalnim formatom zapisivanja. Ako je to slučaj sa mantisom, vrši se zaokruživanje ili odsijecanje najmanje značajnih bitova mantise.

Greška kvantizacije koja se pravi pri zaokruživanju ili odsijecanju mantise zavisi ne samo od pozicije bita na kojem se vrši zaokruživanje ili odsijecanje, već i od vrijednosti eksponenta. Pošto je vrijednost broja koji se zapisuje reda veličine eksponenta, to je greška kvantizacije koja se pravi odsijecanjem:

$$0 \le \varepsilon < 2^{-m} \cdot x,$$

gdje je m broj bita predvidjenih za predstavljanje mantise, dok x predstavlja vrijednost zapisivanog binarnog broja. Sa druge strane, greška kvantizacije koja se pravi zaokruživanjem istog broja je:

$$-\frac{2^{-m}}{2} \cdot x \le \varepsilon < \frac{2^{-m}}{2} \cdot x.$$

Greška koja zavisi i od vrijednosti broja čija se kvantizacija izvršava naziva se *multiplikativna greška*. Analiza uticaja multiplikativne greške na rezultat, u slučaju osnovnih aritmetičkih operacija, je slična kao i u ranije analiziranom slučaju aditivne greške kvantizacije (poglavlje 2.1.3).

Iz dosadašnjeg izlaganja je jasno da se u zapisu sa pomičnim zarezom mogu predstaviti brojevi iz veoma širokog opsega, značajno šireg od opsega obezbijedjenog zapisom sa nepomičnim zarezom ili zapisom cijelih brojeva, koji koriste isti broj bitova. Medjutim, dobitak na širini opsega je postignut na račun tačnosti prikazanih brojeva. Ovu činjenicu je najlakše objasniti na primjeru. Uzmimo da je potrebno zapisati binarni broj

1110000 11001100 10101010 00001111

(razmaci izmedju skupina od po 8 cifara su ostavljeni radi lakšeg praćenja izlaganja). U zapisu sa cjelobrojnim formatom, u slučaju riječi sa 32 bita, zapis bi izgledao ovako:

01110000 11001100 10101010 00001111 (2-6)

Pošto se svi značajni bitovi broja mogu smjestiti u ovom zapisu, broj je potpuno tačno zapisan.

Ako bi ovaj isti broj željeli da reprezentujemo u zapisu sa pomičnim zarezom, najprije bismo ga morali transformisati u eksponencijalni zapis. Broj bi tada bio zapisan na sljedeći način:

$$1.110000\ 11001100\ 10101010\ 00001111 \times 2^{30}$$

Pošto je kod zapisa sa pomičnim zarezom za mantisu predvidjeno 23 bita, to znači da će moći da se zapišu samo prve 23 cifre iza decimalnog zareza. Drugim riječima, ovaj broj bi se zapisao kao:

$$1.110000\ 11001100\ 10101010\ 0 \times 2^{30}$$

dok bi ostale cifre bile odsječene. Kada ovaj broj vratimo u početni oblik, dobijamo:

$$1110000\ 11001100\ 10101010\ 00000000.$$

Ukoliko uporedimo ovu vrijednost sa originalnim brojem (2-5), vidimo da se ti brojevi razlikuju. Dakle, kao posljedica zapisa broja u formatu sa pomičnim zarezom javila se nepreciznost. Iz ovog primjera se vidi da postoje brojevi koje je moguće tačno zapisati u cjelobrojnom zapisu, a nije moguće tačno zapisati u zapisu sa pomičnim zarezom, iako se za zapisivanje koristi isti broj bitova. I pored uočenog nedostatka zapisa sa pomičnim zarezom, on se ipak široko upotrebljava zbog velikog opsega brojeva koje je njime moguće zapisati.

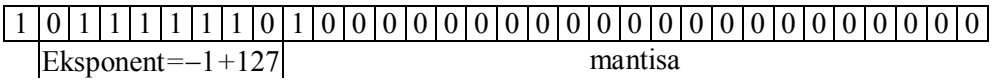
Primjer 2-7: Predstaviti u IEEE 754 standardu, u jednostrukoj i dvostrukoj preciznosti, brojnu vrijednost $-0.75_{(10)}$.

Rješenje: Slijedeći pravila ponderisanog (biased) floating-point zapisa decimalnih brojeva imamo:

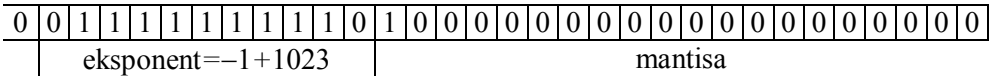
$$-0.75_{(10)}=-0.11_{(2)}=-0.11\times2^{0}=-1.1\times2^{-1}\rightarrow(-1)^{0}\times(1+0.10\ldots0)\times2^{-1+\mathrm{Bias}},$$

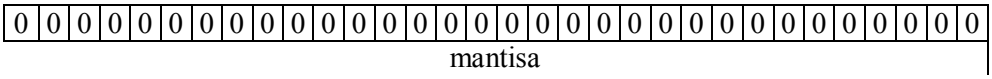
gdje je Bias=127 u slučaju jednostruke preciznosti i Bias=1023 u slučaju dvostruke preciznosti. Sada reprezentacija posmatranog broja izgleda:

U slučaju jednostruke preciznosti:

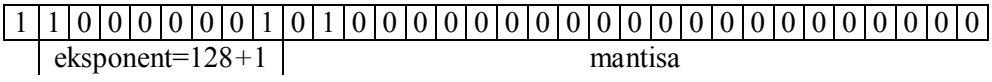


U slučaju dvostruke preciznosti:





Primjer 2-8: Koji dekadni decimalni broj je predstavljen sa dolje prikazanom 32-bitnom riječi, ukoliko se pretpostavi da je u njoj zapisan floating-point broj saglasno standard IEEE 754?



Rješenje: Slijedeći pravila ponderisanog (biased) floating-point zapisa decimalnih brojeva zapažamo da je: s=1, ponderisani (biased) eksponent (označimo ga sa Biased_Exp) Biased_Exp$=129_{(10)}$, te da je razlomljeni (decimalni) dio mantise $0.0100\ldots0_{(2)}=(2^{-2})_{(10)}$. Uvrštavajući ove podatke u izraz (2-5), uz zapažanje da je E=Biased_Eksponent–Bias=129–127=2, jednostavno dobijamo vrijednost reprezentovanog floating-point broja:

$$\mathrm{Broj}=(-1)^{1}\times(1+2^{-2})\times2^{2}=-1.25_{(10)}\times4=-5_{(10)}.$$

2.2 OSNOVNE RAČUNSKE OPERACIJE SA DECIMALNIM BROJEVIMA ZAPISANIM SA POMIČNOM ZAREZOM

2.2.1 Floating-point sabiranje

Floating-point sabiranje binarnih brojeva se izvršava na adekvatan način kao u dekadnom brojnom sistemu, odnosno nizom sljedećih koraka, algoritamski predstavljenih na slici 1-1:

I korak: *Izjednačavanje exponenata sabiraka* dovodjenjem eksponenta manjeg od sabiraka (inkrementiranjem) na nivo eksponenta većeg sabirka i shiftovanjem mantise ovog broja za onoliko mjesta u desnu stranu za koliko se vrši inkrementiranje njegovog eksponenta,

II korak: *Sabiranje mantisa sabiraka, prethodno dovedenih na istu vrijednost eksponenta*. Sabiranjem mantisa se istovremeno odredjuje znak sume,

III korak: *Normalizacija sume*, koja se postiže pomjeranjem (shiftovanjem) u lijevu/desnu stranu mantise sume izračunate u II koraku i dekrementiranjem/inkrementiranjem vrijednosti exponenta iste sume. Nakon toga, pošto je moguća promjena veličine eksponenta (njegovim inkrementiranjem/dekrementiranjem), potrebno je provjeriti da li je došlo do overflow-a (underflow-a). Ukoliko je došlo do over/underflow-a, prekida se sa izvršavanjem i poziva se procedura (*exception* procedura), kojom se nastoji prevazići problem i nastaviti sa izvršavanjem algoritma. Primijetimo da je prilikom moguće promjene vrijednosti eksponenta njegovim inkrementiranjem ili dekrementiranjem uvijek potrebno provjeriti da li je došlo do over/underflow-a. U prvom koraku takodje vršimo inkrementiranje eksponenta jednog od sabiraka, ali ipak ne provjeravamo da li je došlo do over/underflow-a zato što inkrementiranje vršimo do vrijednosti eksponenta većeg od sabiraka, čijom vrijednošću nije došlo do izuzetka u odnosu na over/underflow,

IV korak: *Zaokruživanje mantise sume na odgovarajući broj bitova*. Pošto se zaokruživanjem može dobiti nenormalizovani broj, potrebno je provjeriti se da li je zaokruženi broj normalizovan! Ukoliko zaokruženi broj nije normalizovan, vraćamo se na III korak (njegovu normalizaciju).

Primjer 2-9: Sabrati brojeve $0.5_{(10)}+(-0.4375)_{(10)}$ u binarnom obliku i floating-point zapisu, pretpostavljajući da su 4 bita raspoloživa za zapisivanje decimalnog dijela mantise, kao i 2 bita za zapisivanje eksponenta.

Rješenje: Predstavimo najprije zadate dekadne brojeve u binarnom obliku i floating-point normalizovanom zapisu:

$$0.5_{(10)} \rightarrow 0.1_{(2)} = 0.1_{(2)} \times 2^0 = 1.000 \times 2^{-1}$$

$$-0.4375_{(10)} \rightarrow -0.0111_{(2)} = -0.0111_{(2)} \times 2^0 = -1.110 \times 2^{-2}$$

I korak: *Izjednačavanje exponenata sabiraka*

$$= -1.110 \times 2^{-2} = -0.111 \times 2^{-1}$$

II korak: *Sabiranje mantisa sabiraka, prethodno dovedenih na istu vrijednost eksponenta*

$$1.000 - 0.111 = 0.001.$$

Sabiranjem mantisa zadatih sabiraka istovremeno se odredjuje i bit znak sume $\Rightarrow$ Suma$=0.001 \times 2^{-1}$.

III korak: *Normalizacija sume*, Suma$=1.000 \times 2^{-4}$.

Ispitujemo da li je došlo do over/underflow-a:

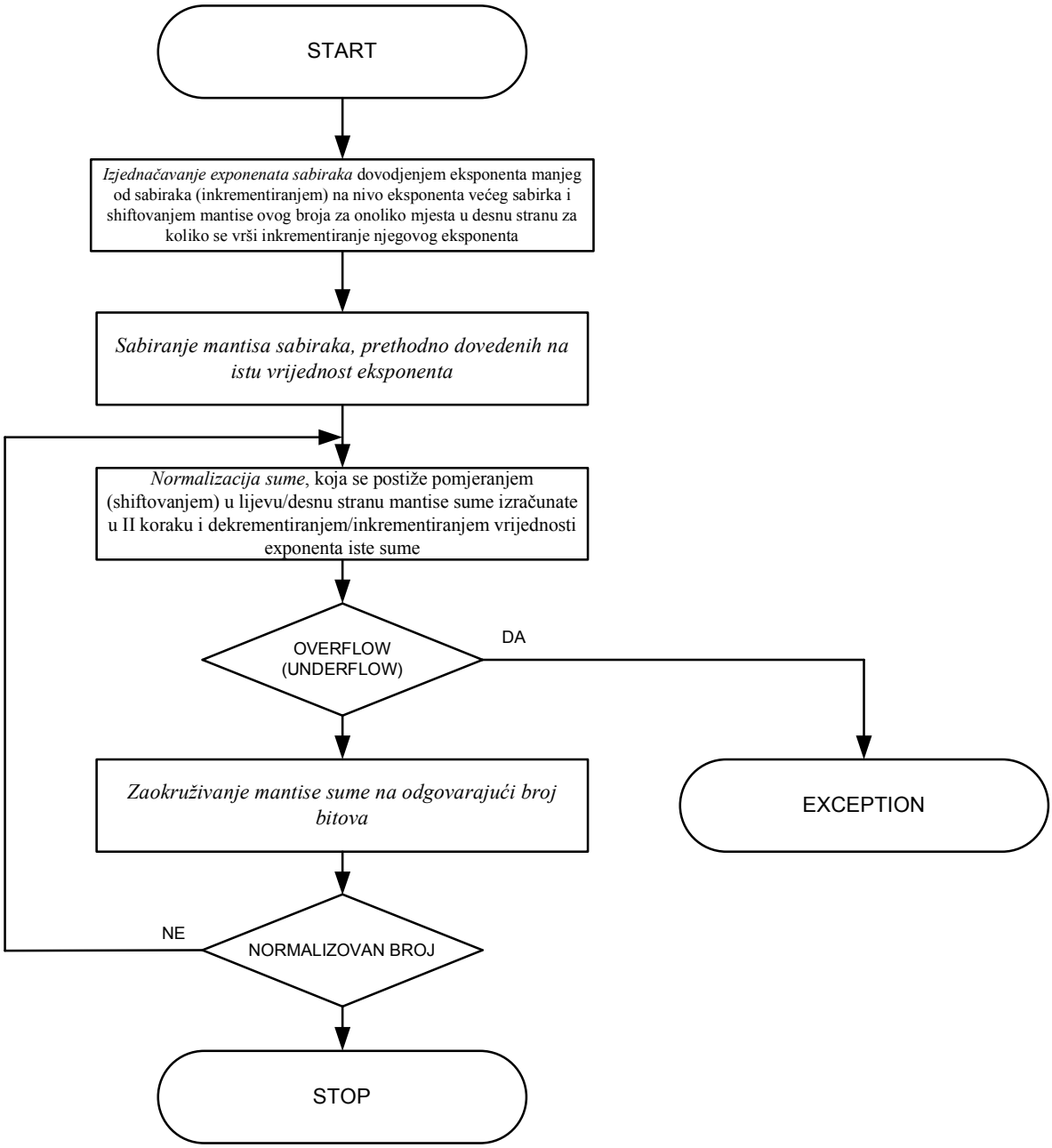
$-126 \leq -4 \leq 127$ (posmatrajući originalne veličine eksponenata),

$1 \leq -4 + 127 \leq 254$ (posmatrajući biased veličine eksponenata) $\Rightarrow$ **nije došlo** do over/under flow-a!

IV korak: *Zaokruživanje mantise sume.* Izračunata suma je u okviru dozvoljenog broja bitova, tako da nema potrebe za njenim zaokruživanjem. Takodje, broj je prethodno normalizovan i nema potrebe za ponovnom normalizacijom, pošto u ovom koraku broj nije mijenjan zaokruživanjem.

$$\mathbf{Suma = 1.000 \times 2^{-4} \rightarrow (1/2^4)_{(10)} = 0.0625_{(10)}}$$

Provjerom u dekadnom brojnom sistemu da $0.0625_{(10)}$ predstavlja zbir zadatih dekadnih brojeva $0.5_{(10)}$ i $(–0.4375)_{(10)}$. Drugim riječima, dobijeni rezultat je odgovarajući.

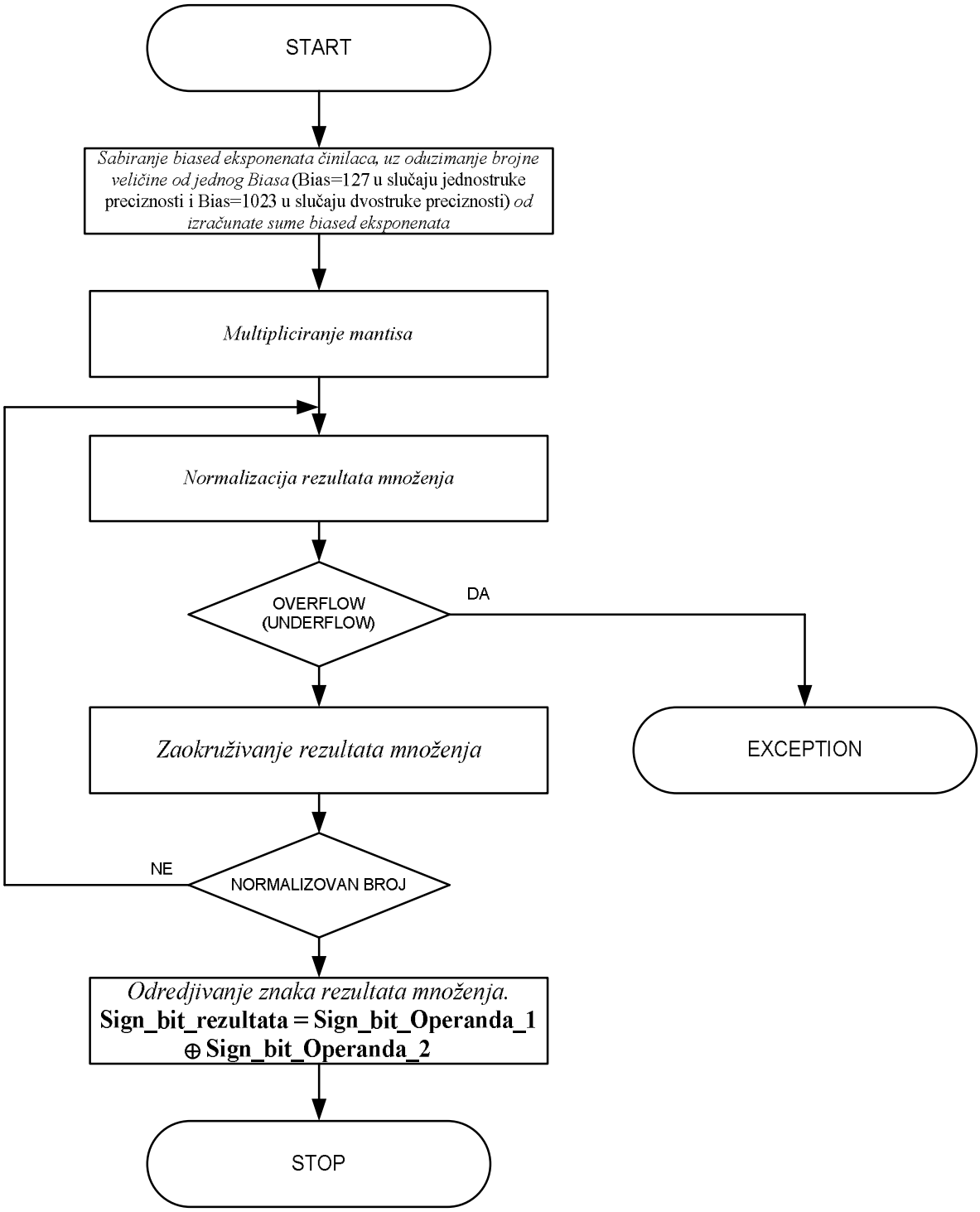


Slika 1-1. Algoritam za izvršavanje floating-point sabiranja.

2.2.2 Floating-point množenje

Floating-point množenje binarnih brojeva se izvršava na adekvatan način kao u dekadnom brojnom sistemu, odnosno nizom sljedećih koraka, algoritamski predstavljenih na slici 1-2:

I korak: *Sabiranje biased eksponenata činilaca, uz oduzimanje brojne veličine od jednog Biasa* (Bias=127 u slučaju jednostruke preciznosti i Bias=1023 u slučaju dvostruke preciznosti) *od izračunate sume biased eksponenata*. Naime, sumiranjem biased eksponenata činilaca dva puta se unosi vrijednost Bias u sumu biased eksponenata. Sa druge strane, biased eksponent traženog proizvoda se pronalazi dodavanjem jednog Biasa na vrijednost eksponenta proizvoda. Stoga, u cilju dobijanja korektnog rezultata, brojna vrijednost veličine jednog Biasa se mora oduzeti od sume biased eksponenata činilaca,



Slika 1-2. Algoritam za izvršavanje floating-point množenja.

II korak: *Multipliciranje mantisa*,

III korak: *Normalizacija rezultata množenja, zajedno sa testiranjem da li je došlo do over/underflow-a*,

IV korak: *Zaokruživanje rezultata množenja, zajedno sa testiranjem da li je dobijeni rezultat množenja i dalje normalizovan*,

V korak: *Odredjivanje znaka rezultata množenja*. Znak rezultata množenja je pozitivan (s=0) ukoliko su činioci (operandi) istog znaka (Sign_bit_Operanda_1=Sign_bit_Operanda_2), odnosno

negativan (s=1) ukoliko činioci (operandi) ***nijesu*** istog znaka (Sign_bit_Operanda_1 ≠Sign_bit_Operanda_2). Drugim riječima, znak rezultata množenja (Sign_bit_rezultata) se odredjuje rješavanjem jednostavnog logičkog izraza:

Sign_bit_rezultata = Sign_bit_Operanda_1 ⊕ Sign_bit_Operanda_2.

Primjer 2-10: Pomnožiti brojeve $0.5_{(10)}\times(-0.4375)_{(10)}$ u binarnom obliku i floating-point zapisu, pretpostavljajući da su 4 bita raspoloživa za zapisivanje decimalnog dijela mantise, kao i 2 bita za zapisivanje eksponenta.

Rješenje: Predstavljeno binarnim floating-point zapisom: $(1.000\times2^{-1})\times(-1.110\times2^{-2})$.

I korak: *Sabiranje biased eksponenata činilaca, uz oduzimanje jednog Biasa od izračunate sume.*

$$(-1+127)+(-2+127)-127=-3+127=124.$$

II korak: *Multipliciranje mantisa*, $1.000\times1.110=1.110000$.

$$\Rightarrow \text{Result}=1.110\times2^{-3}.$$

III korak: *Normalizacija* rezultata *množenja*: Rezultat je već normalizovan!

Provjera over/underflow-a: $-126 \le -3 \le 127$ (posmatrajući originalne veličine eksponenata),

$1 \le -3+127 = 124 \le 254$ (posmatrajući biased veličine eksponenata)

$\Rightarrow$ **nije došlo** do over/under flow-a!

IV korak: *Zaokruživanje* rezultata *množenja*: Nema potrebe za zaokruživanjem rezultata množenja! Takodje, pošto rezultat nije mijenjan zaokruživanjem, *nema potrebe ni za normalizacijom*!

V korak: Odredjivanje znaka rezultata množenja.

Sign_bit_rezultata = Sign_bit_Operanda_1 ⊕ Sign_bit_Operanda_2 = 0 ⊕ 1 = 1

$$\Rightarrow \textbf{Result}=\mathbf{-1.110\times2^{-3}}.$$

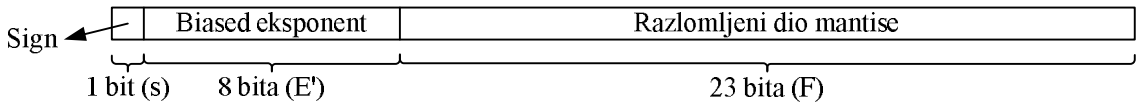
Provjerom u dekadnom brojnom sistemu: $-1.75_{(10)}\times2^{-3}=(-1.75/8)_{(10)}=-0.21875_{(10)}$, što je proizvod zadatih dekadnih brojeva $0.5_{(10)}$ i $(-0.4375)_{(10)}$.

2.3 RIJEŠENI ZADACI ZA VJEŽBANJE:

1. Predstaviti brojeve **7.6875** i **−17.5** u zapisu sa pokretnim zarezom i jednostrukom tačnošću (IEEE 754 standard).

Za ovu tačnost (4 bajta) opseg vrijednosti je $\pm\mathbf{2}\times\mathbf{2}^{\mathbf{-128}}$ do $\pm\mathbf{2}\times\mathbf{2}^{\mathbf{127}}$.

Po ovom standardu, realni brojevi se predstavljaju na sljedeći način:



Vrijednost V, na ovaj način zapisanog broja, u 32-bitnom registru je:

$$\mathbf{V} = (-1)^{s}\times(1+\mathbf{F})\times2^{\,E'-127}$$

Mantisa predstavlja decimalni dio binarnog broja u *normalizovanom zapisu*, dok je E' −127 prava vrijednost eksponenta broja u normalizovanom zapisu. Normalizovani zapis binarnog

broja je zapis čija je cjelobrojna vrijednost uvijek jednaka **1**, tj. za naše brojeve 7.6875 i −17.5 normalizovani zapis je:

$$7.6875_{10} = 111.1011_2 = 1.111011\times2^2$$
$$-17.5_{10} = -10001.1_2 = -1.00011\times2^4$$

Normalizovani zapis je polazna tačka za pretvaranje brojeva u zapis sa pokretnim zarezom i jednostrukom tačnošću (ili dvostrukom). Iz tog zapisa direktno čitamo vrijednost s, E' i F. Za naša dva broja imamo:

Za 7.6875: **s = 0**, **E'** = 2+127 = 129_{10} = **10000001_2**,

F = $11101100000000000000000_2$

odnosno:

0	**10000001**	**11101100000000000000000**

Za -17.5: **s = 1**, **E'**= 4+127 = 131_{10} = **10000011_2**,

F = $00011000000000000000000_2$

odnosno:

1	**10000011**	**00011000000000000000000**

Kod zapisa sa dvostrukom tačnošću stvar je potpuno ista, s tim što važe sljedeće razlike:

- dvostruka tačnost podrazumijeva zapis sa **64** bita,
- eksponent zauzima **11** bita, a mantisa **52** bita,
- opseg vrijednosti je $\pm2\times2^{-1024}$ do $\pm2\times2^{1023}$
- pomjerena mantisa se dobija sabiranjem prave mantise sa **1023**.

2. Šta predstavlja sekvenca bitova:

1100 0001 0101 0101 0001 0000 0000 0000

pretpostavljajući da je u pitanju:

a) cio broj u zapisu sa dvojnim komplementom?
b) broj u zapisu sa pokretnim zarezom i jednostrukom tačnošću?

a) 1100 0001 0101 0101 0001 0000 0000 0000
0011 1110 1010 1010 1110 1111 1111 1111

\+ 1

0011 1110 1010 1010 1111 0000 0000 0000

Dakle, kada sekvencu tumačimo kao cio broj njegova vrijednost je:

$$-(2^{12}+2^{13}+2^{14}+2^{15}+2^{17}+2^{19}+2^{21}+2^{23}+2^{25}+2^{26}+2^{27}+2^{28}+2^{29})$$

b) Kada je tumačimo kao broj u zapisu sa pokretnim zarezom i jednostrukom tačnošću onda koristimo sličnu proceduru kao u prvom zadatku, tj. vršimo podjelu sekvence na znak bit, eksponent i mantisu. U našem slučaju ćemo onda imati:

s = **1** → broj je negativan
E' = $\mathbf{10000010}_2 = \mathbf{130}_{10}$ → **E = E '** – 127 = **3**
F = $\mathbf{10101010001000000000000}_2$

Na osnovu ovih polja zaključujemo da je vrijednost broja predstavljenog datom sekvencom:

$$-(\mathbf{1} + \mathbf{2^{-1}} + \mathbf{2^{-3}} + \mathbf{2^{-5}} + \mathbf{2^{-7}} + \mathbf{2^{-11}}) \times \mathbf{2^3}$$

3. Sabrati brojeve $\mathbf{3.5}_{(10)}$ i $\mathbf{1.125}_{(10)}$ koristeći algoritam za sabiranje binarnih brojeva. Preciznost zapisa mantise je 8 bita.

Predstavimo najprije brojeve u normalizovanom zapisu:

$3.5_{(10)} = 11.1_{(2)} = 1.11\times2^1$, odnosno u biased formi: $1.11\times2^{1+127}=1.11\times2^{128}$
$1.125_{(10)} = 1.001_{(2)} = 1.001\times2^0$, odnosno u biased formi: $1.001\times2^{0+127}=1.001\times2^{127}$

KORAK 1. *Izjednačavanje eksponenata sabiraka*:

$1.001\times2^{127} = 0.1001\times2^{128}$

KORAK 2. *Saberu se mantise sabiraka*:

$1.11 + 0.1001 = 10.0101$

KORAK 3. *Normalizovanje sume i provjera overflow-a i underflow-a*:

$10.0101\times2^{128} = 1.00101\times2^{129}$

$1 \le E_a' = 129 \le 254 \rightarrow$ Nema overflow-a i underflow-a!

KORAK 4. *Zaokruživanje dobijene normalizovane sume*:

Nema potrebe za zaokruživanjem! Dobijeni broj je:

$1.00101\times2^{129} = 100.101\times2^{127}$,

odnosno kada se oslobodimo biased forme: $100.101_{(2)} = 4.625_{(10)}$

4. Pomnožiti brojeve $\mathbf{2.25}_{(10)}$ i $\mathbf{-1.5}_{(10)}$ koristeći algoritam za množenje binarnih brojeva. Preciznost zapisa mantise je 8 bitova.

Predstavimo prvo brojeve u normalizovanom zapisu:

$2.25_{(10)} = 10.01_{(2)} = 1.001\times2^1$, odnosno u biased formi: $1.001\times2^{1+127}=1.001\times2^{128}$
$-1.5_{(10)} = -1.1_{(2)} = -1.1\times2^0$, odnosno u biased formi: $-1.1\times2^{0+127}=-1.1\times2^{127}$

KORAK 1. *Sabiranje eksponenata brojeva*:

$E_m' = 128+127-127=128$, odnosno $E = E_m' - 127 = 1$

KORAK 2. *Pomnože se mantise brojeva*:

$1.001\times1.1 =$ 1001
+1001
1.1011 → Res= 1.10110000×2^1

KORAK 3. *Normalizovanje proizvoda i provjera overflow-a i underflow-a*:

1.1011×2^{128} → Nema potrebe za normalizovanjem!

$1 \leq E_m' = 128 \leq 254$ → Nema overflow-a i underflow-a!

KORAK 4. *Zaokruživanje dobijenog normalizovanog proizvoda*:

Nema potrebe za zaokruživanjem!

KORAK 5. *Znak proizvoda*:

$S_m = S_1 \oplus S_2$

$S_1 = 0$; $S_2 = 1$ → $S_m = 0 \oplus 1 = 1$

Zaključujemo da je rezultat negativan:

$-1.1011 \times 2^{128} = -11.011_{(2)} \times 2^{127}$

odnosno kada se oslobodimo biased forme: $-11.011_{(2)} = -3.375_{(10)}$