Aritmetika pomičnog zareza

De Martini, Antonio

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Antonio De Martini

ARITMETIKA POMIČNOG ZAREZA

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio De Martini

Matični broj: 46055/17–R

Studij: Poslovni sustavi

ARITMETIKA POMIČNOG ZAREZA

ZAVRŠNI RAD

Mentor :

Doc. dr. sc. Bojan Žugec

Varaždin, rujan 2021.

Antonio De Martini

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema rada je aritmetika pomičnog zareza. Za potpuno savladavanje teme aritmetike pomičnog zareza prvo je potrebno razjasniti što su to realni brojevi te kako računalo vidi te brojeve u binarnom brojevnom sustavu te kako računalo te brojeve sprema u memoriju. Zatim slijede pojašnjenja svake aritmetičke operacije na način kako ljudi izvršavaju aritmetičke operacije kako bi se lakše objasnila računalna aritmetika. Za kraj ostaje pojašnjenje grešaka zaokruživanja. Greške zaokruživanja se trebaju zasebno objasniti iz razloga što se neki realni brojevi ne mogu zapisati u ograničeni memorijski prostor. Svaka cjelina će biti pojašnjenja kroz primjere i kroz programsko rješenje istih primjera.

Ključne riječi: Realni brojevi; Aritmetika; IEEE754; Binarni sustav; Greške zaokruživanja;

Sadržaj

1. Uvod

S realnim brojevima se susrećemo još u osnovnoj školi te cijelo vrijeme dok se školujemo učimo više o realnim brojevima. Učenje realnih brojeva se ostvaruje u malim koracima. Na početku školovanja se uči skup prirodnih brojeva i rad s takvim brojevima. Kasnije se znanje proširuje na skup cijelih brojeva i računanje takvih brojeva. Kad se utvrde prethodna znanje ona ostane za obraditi skup racionalnih brojeva prije nego se krene na savladavanje realnih brojeva. Kada se jednom savladaju osnove, znanje o realnim brojevima se samo nadograđuje i proširuje, kako kroz ostatak školovanja tako i u primjeni u stvarnom životu.

Obrađivanje realnih brojeva dosta često može biti dugotrajan proces u kojem se može vrlo lako pogriješiti. Razvijanje prvih računalnih pomagala, elektroničkih pomagala pa sve do današnjih modernih pomagala i sve naprednijih programskih rješenja za razne računske operacije olakšavaju ljudima rješavanje sve težih i kompliciranijih operacija vrlo velikih brojeva po pitanju količini znamenaka u znatno kraćem vremenskom periodu uz vrlo malu šansu pojave nekakve greške pri računanju.

Iako osobama koje koriste recimo računalo za obradu realnih brojeva nije bitno na koji način računala rješavaju zadani problem već je tim osobama bitna samo brzina i točnost. Dok s druge strane, razni inženjeri, računalni stručnjaci i programeri koji razvijaju opremu za rješavanje navedenih problema moraju jako dobro znati kako računalo interpretira realne brojeve, kako računalo sprema takve brojeve u memoriju, koji je problemi mogu javiti prilikom spremanja podataka, kako računalo obrađuje aritmetičke operacije, koje su naredbe potrebne te mnoga druga pitanja.

U ovom radu će se na početku pobliže pojasniti binarni brojevni sustav, to jest, pretvaranje iz dekadskog brojevnog sustava koje čovjek čita i razumije u binarni brojevni sustav, brojevni sustav koje računalo interpretira podatke vanjskog svijeta u oblik koji računalo može čitati, spremiti i obraditi. Za pojašnjenje spremanja realnih brojeva u računalo će se objasniti preko IEEE754 standarda, 32-bitnog standarda za pomični zarez. Zatim slijedi aritmetika računala gdje će se pojasniti svaka aritmetička operacija te kako je računalo izvršava. Za kraj ostaje obraditi greške prilikom zaokruživanja koje se javljaju kada se jako veliki ili jako mali broj pokuša spremiti u ograničeni memorijski prostor koji je nedovoljno velik.

2. Metode i tehnike rada

Rad je potrebno napraviti u LaTeX-u. Na računalu je Windows 10 operacijski sustav te iz tog razloga rad će biti napisan u TeXworks-u. TeXworks je besplatan softver dostupan za Windows operacijske sustave te je usmjeren na izravno generiranje PDF dokumenata. TeXworks ima grafičko korisničko sučelje koje pojednostavljuje izradu završnog rada u TeX obliku. Kasnije tokom izrade završnog rada sam na preporuke od kolega studenata prešao na Overlaaf, online web LaTeX uređivač sa istim mogućnostima, ali većom jednostavnošću te je završni rad u njemu i dovršen.

Za izradu programskog rješenja potrebno je koristiti Python programski jezik. Python je interpreterski, interaktivni, objektno orijentirani programski jezik visoke razine. Python je fleksibilan programski jezik koji služi većinom za pisanje jednostavnih programa, ali se može koristiti i za složenije programe te se lagano uči zbog toga što ima jednostavnu sintaksu što ga čini jednostavnim i za čitanje. Pošto je Python interpreterski jezik, programi napisani u njemu vrše se sporije pogotovo ako je u pitanju složeni program, ali za potrebe završnog rada program napravljen u Python-u radi savršeno.

Za obradu teme literaturu sam prikupljao kako bi pobliže shvatio greške zaokruživanja (i ostalo za greške). Za teorijsku obradu binarnog brojevnog sustava, pretvaranje brojeva u IEEE754 zapis i aritmetičke operacije realnih brojeva sam koristio dosta manje izvora literature pošto sam se cijelo srednjoškolsko obrazovanje i za vrijeme studiranja na FOI-u susretao s navedenim pojmovima. Za navedene pojmove sam literaturu koristio samo da se podsjetim kako ne bi radio greške dok sam primjere samostalno izrađivao. Za programsko riješenje sam koristio W3Schools, besplatnu obrazovnu web stranicu za učenje programskih jezika. Nisam ima većih problema za savladavanje programskog jezika Python pošto se moje srednjoškolsko obrazovanje bilo bazirano na programiranju te sam u drugom razredu srednje škole imao predmet posvećen programiranju u Python-u,a zatim na fakultetu iz predmeta Informatika 1 sam imao rad za bodove u Python-u.

3. Realni brojevi u računalu

Davno prije, prvim razvojem ljudskog društva se stvorila potreba za računanjem i mjerenjem. Razvijanjem prvih civilizacija se razvijala i svijest o potrebama matematike. Tako su se razvojem matematike od jednostavnih brojevnih sustava stvarali složeniji brojevni sustavi, a od složenijih brojevnih sustava su se otvarale mogućnosti za razvoj računskih operacija, razvoj računskih operacija dovodi do potrebe za drugim i većim skupom brojeva te se tako matematika razvijalo do danas. Matematika koju danas poznajemo ima nekoliko skupova brojeve i mnoštvo operacijskih operacija.

Za potpuno razumijevanje skupa realnih brojeva potrebno je razjasniti sve skupove brojeva koje realni brojevi obuhvaćaju. Najosnovniji skup brojeva je skup prirodnih brojeva. Označava se slovom $\mathbb{N}$ i predstavlja sve pozitivne cijele brojeve ($\mathbb{N} = 1, 2, 3, 4, ...n$). Skup cijelih brojeva je skup svih cijelih brojeva i označava se slovom $\mathbb{Z}$. U odnosu s prirodnim skupom, u skup cijelih brojeva ulaze i negativni ($\mathbb{Z} = -n, ..., -3, -2, -1, 0, 1, 2, 3, ..., n$). Slijedeći prošireni skup brojeva je skup racionalnih brojeva $\mathbb{Q}$, a u taj skup brojeva pripadaju $\mathbb{Q} = \frac{a}{b}; a, b \in \mathbb{Z}, b \neq 0$. Postoje brojevi koji se ne mogu zapisati u skupu racionalnih brojeva kao recimo $\sqrt{2}$. Takvi brojevi se zovu iracionalni brojevi te se označavaju sa $\mathbb{I}$. Skup realnih brojeva je skup koji sadrži sve racionalne i sve iracionalne brojeve ($\mathbb{R} = \mathbb{Q} \cup \mathbb{I}$) što znači da se svi mogući brojevi nalaze unutar skupa realnih brojeva.

Kako se razvijalo društvo i matematika unutar društva te razvojem matematike, računanje je postajalo učestalije i kompliciranije što je dovodilo do razvoja prvih pomagala za računanje. Od prvog takvog pomagala kao što je Abak (niz štapića na kojima je niz kuglica) koji je služio za osnovne aritmetičke operacije pa do Računala prve generacije su se računala razvijala i rješavala sve kompliciranije i kompleksnije operacije. Računala prve generacije označavaju početak izrade modernih računala koja rješavaju sve kompleksnije operacije, postaju brža, sve većih kapaciteta, mogućnošću obrade veće količine podataka pa sve do računala kakva danas poznajemo. Današnje računalo se definira kao složeni uređaj koji služi za izvršavanje matematičkih operacija koje se mogu izraziti u numeričkom ili logičkom obliku. Računala su sastavljena od dijelova koji obavljaju jednostavne i jasno određene funkcije. Složenim među djelovanjem tih dijelova rezultira sposobnošću računala da obrađuje informacije te da prema unaprijed zadanim uputama ili instrukcijama osnovne operacije unosa podataka, obrade podataka, prikazivanje podataka i pohranjivanje podataka.

U današnje vrijeme je nemoguće živjeti bez računala te ljudi svakodnevno koriste računalima. Danas je većina ljudi informatički pismena, to jest, zna se koristiti računalom u bilo kakvu osobnu svrhu, ali većina tih ljudi ne zna kako računalo vidi recimo pretraživanje na računalu ili uređivanje teksta u Word-u. Računalo kod obrade slike ne obrađuje sliku već niz električnih signala koji zajedno tvore sliku. Električne signale dijelimo na „nema struje“ i „ima struje“ te ih označavamo s 0 i 1 što nas dovodi do pojma binarnog sustava.

3.1. Binarni sustav

Kao što je već rečeno, računalo sve podatke koje obrađuje u binarnom sustavu te pomoću programa ih prikazuje na način koje ih čovjek može razumjeti. Kako bi se obradila tema Aritmetika pomičnog zareza potrebno je prvo obraditi binarni brojevni sustav. Postoji više brojevnih sustava, ali za potrebe završnog rada potreban je samo dekadski i binarni brojevni sustav. Dekadski brojevni sustav je sustav s bazom 10 te može prikazati 10 znakova: 0, 1, 2, 3, 4, 5, 6, 7, 8 i 9. Binarni brojevni sustav je brojevni sustav s bazom 2 što znači da taj brojevni sustav može prikazati dva znaka: 0 i 1. Svaki dekadski broj se može prikazati u binarnom te se svaki binarni broj može prikazati u dekadskom. Proces pretvaranja iz jednog u drugi binarni sustav je najlakše objasniti kroz primjere. Kako bi pretvorili dekadski broj u binarni brojevni sustav, broj se treba dijeliti sa 2 dok se ne dođe do 0, a ostatak se zapisuje sa strane. Za primjer uzmimo broj $123_{(10)}$:

$$123 : 2 = 61(\text{ ostatak je } \mathbf{1})$$

$$61 : 2 = 30(\mathbf{1})$$

$$30 : 2 = 15(\mathbf{0})$$

$$15 : 2 = 7(\mathbf{1})$$

$$7 : 2 = 7(\mathbf{1})$$

$$3 : 2 = 1(\mathbf{1})$$

$$1 : 2 = 0(\mathbf{1})$$

Za kraj kako bi se prikazao broj 123 u binarnom brojevnom sustavu, ostatke treba zapisati na način da se krene zapisivati od dolje pa na gore te se dobije $1111011_{(2)}$. Za računanje broja iz binarnog u dekadski brojevni sustav potrebno je znati da je vrijednost binarne prve binarne znamenke desne strane 2^0 te svako mjesto lijevo od tog broj se povećava eksponent za 1:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1

Zatim se svaki broj u binarnom broju množi sa vrijednošću svog mjesta:

$$1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$$

$$1 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 123$$

Tako se računaju pozitivni cijeli brojevi. Sličan je postupak za pretvaranje decimalnog broja binarni sustav. Za pretvaranje decimalnog broja u binarni potrebno je znati da se cijeli dio broja računa isto kao u prethodnom primjeru. Za računanje decimalnog djela potrebno je

broj iza decimalne točke množiti sa 2 te se dobivena cijela vrijednost zapisuje, a ostatak iza decimalnog broja ponovo množi s 2 i tako sve dok broj iza decimalne točke nije 0, a rezultat se zapisuje s gore na dolje. Uzmimo broj $5.78125_{(10)}$:

$$\begin{aligned} 5 : 2 = 2(\mathbf{1}) \qquad & 0.78125 \cdot 2 = \mathbf{1}.5625 \\ 2 : 2 = 1(\mathbf{0}) \qquad & 0.5625 \cdot 2 = \mathbf{1}.125 \\ 1 : 2 = 0(\mathbf{1}) \qquad & 0.125 \cdot 2 = \mathbf{0}.25 \\ & 0.25 \cdot 2 = \mathbf{0}.5 \\ & 0.5 \cdot 2 = \mathbf{1} \end{aligned}$$

Rezultat pretvaranja decimalnog broja $5.78125_{(10)}$ u binarni sustav je $101.11001_{(2)}$. Kod pretvaranja binarnog broja u dekadski brojevni sustav potrebno je znati da prva vrijednost iza decimalne točke poprima vrijednost 2^1 , a svako iduće mjesto na desno poprima vrijednost eksponenta za jedan manje. Kao u prošlom primjeru, svaki broj se množi s vrijednošću svoga mjesta:

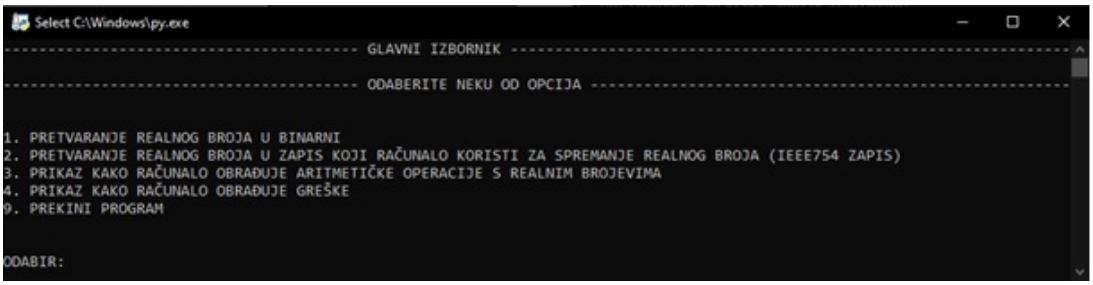
$$1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} =$$

$$1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 + 1 \cdot 0.5 + 1 \cdot 0.25 + 0 \cdot 0.125 + 0 \cdot 0.0625 \cdot 1 \cdot 0.03125 = 5.78125$$

Za kraj kako bi se pokrio cijeli skup realnih brojeva još treba znati kada se želi zapisati predznak u binarnom brojevnom sustavu ispred broja se zapisuje bit predznaka (0 za pozitivan, a 1 za negativan broj). Za to bi se trebalo predodrediti koliko bitova se želi odvojiti za memorijski prostor broja. Ako je riječ o 8-bitnom zapisu broja, to bi značilo da je prvi bit zapravo bit predznaka a ostalih 7 za zapis broja. Recimo broj -100 bi u 8-bitnom zapisu s predznakom bio 11100100

3.1.1. Informacije o Glavnom izborniku programa u Pythonu te realizacija dijela za pretvorbu iz realnog broja u binarni

Praktični dio rada je napraviti program u Pythonu koji će potkrijepiti sve primjere koji se pojavljuju u završnom radu. Kada se pokrene program, korisniku se ispisuje glavni izbornik gdje odabire koju željenu funkciju korisnik želi upisom jednog od ponuđenih brojeva na mjesto gdje piše ODABIR: kako to izgleda na sljedećoj slici:



Slika 1: Glavni izbornik

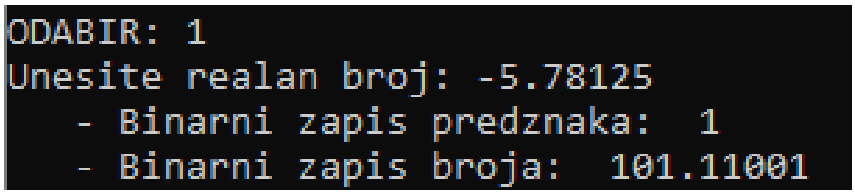
Izbornik se realizira ispisom što se vidi na ekranu pomoću print():

```
while True:
    print("----------------GLAVNI IZBORNIK----------------")
    print("---------------- ODABERITE NEKU OD OPCIJA ----------------\n")
    print("1. PRETVARANJE REALNOG BROJA U BINARNI")
    print("2. PRETVARANJE REALNOG BROJA U ZAPIS KOJI RACUNALO KORISTI ZA SPREMANJE
    REALNOG BROJA (IEEE754 ZAPIS)")
    print("3. PRIKAZ KAKO RACUNALO OBRADUJE ARITMETICKE OPERACIJE S REALNIM
    BROJEVIMA")
    print("4. PRIKAZ KAKO RACUNALO OBRADUJE GRESKE")
    print("9. PREKINI PROGRAM")
```

Nakon ispisa tog djela, program traži korisnika da unese željenu opciju. To se izvršava na način da se pored ODABIR: upiše broj uz željenu opciju. Ovisno o odabranoj opciji program ulazi u jedan od if opcija:

```
if ODABIR in ('1', '2','3','4','9'):
        if ODABIR == '1':
            unos=float(input("Unesite realan broj: "))
            ODABIR=1
            IEEE754(unos, ODABIR)

        elif ODABIR == '2':
            unos=float(input("Unesite realan broj: "))

            IEEE754(unos, ODABIR)

        elif ODABIR == '3':
            aritmetika()

        elif ODABIR == '4':
            greske()

        elif ODABIR=='9':
            print("Program je zavrsio s radom!!!!")
            break

    else:
        print("Ne postoji ta opcija!!!")
```

Kako bi se pokrenula funkcija računanja realnog broja u binarni potrebno je u ODABIR: upisati broj 1. Zatim program ulazi u funkciju IEEE754(unos, ODABIR) nakon što korisnik unese realni broj. Program nakon unosa računa binarni zapis predznaka i binarni zapis broja. Za primjer uzmimo $-5.78125_{(10)}$ te je rezultat sljedeći:



Slika 2: Rezultat funkcije 1

Nakon završetka, program se vrača na glavni izbornik. Funkcija u kojoj počinje realizacija pretvaranja je IEEE754(). Kako se ne bi gomilao kod te se ponovo sve pisalo, IEEE754() funkcija se koristi samo za računanje predznaka broja te nakon toga program ulazi u funkciju binarni(). Nakon završetka te funkcije se provjerava dali se u glavnom izborniku odabrala funkcija 1 te ako je tu se onda prekida IEEE754() dok kod ostalih opcija funkcija nastavlja dalje, ali više o toj funkciji u nastavku završnog rada.

```
def binarni(uneseni, mjestaMantise, predZ):

    cijeli, decimalni = str(uneseni).split(".")
    cijeli = int(cijeli)

    decBroj = (str(bin(cijeli))+".").replace('0b','')
    dec=0
    d=0
    for x in range(0, mjestaMantise):
        decimalni = str('0.')+str(decimalni)
        dec = (float(decimalni)\cdot2)
        cijeli, decimalni = str(dec).split(".")
        decBroj += cijeli
        if float(decimalni)==0 and d==0:
                d=1;
                print("   - Binarni zapis predznaka: ", predZ)
                print("   - Binarni zapis broja: ", decBroj)
    return decBroj
```

Na početku funkcije binarni(), str(uneseni).split(".") dijeli uneseni broj podjeljuje na cijeli i decimalni na mjestu gdje se točka nalazi, zatim se cijeli string pretvara u int kako bi se dobio cijeli broj. Funkcija bin() je python funkcija za pretvaranje dekadskog broja u decimalni, ali ta funkcija može pretvarati samo pozitivni cijeli broj tako da ta funkcija nije povoljna za pretvaranje realnih brojeva u binarni sustav. U retku (str(bin(cijeli))+".").replace('0b','') se računa binarni broj cijelog dijela, dodaje točka i briše „0b“. Funkcija bin() prije binarnog broja ispisuje „0b“ kako bi inducirao kako se radi o brojevnom sustavu, npr za bin(9) bi ispis bio „0b1001“. Za decimalni dio se ulazi u for petlju i u njoj se u varijablu decimalni sprema string 0. i onda ostali decimalni brojevi.

Kako je sljedeći dio koda malo kompliciran, bit će objašnjena kroz primjer. npr za broj

5.78125 trenutno je varijabla cijeli=5, a varijabla decimalni=0.78125, u varijablu dec se decimalni=0.78125 množi s 2 što bi dalo broj 1.5625. Zatim se taj broj podjeljuje na cijeli i decimalni dio (cijeli, decimalni = str(dec).split(".")) te bi nova vrijednost varijable cijeli bila 1, a decimalni 5625, zatim bi decBroj koji je trenutno 101. bi mu se dodala vrijednost cijeli te nova vrijednost bi bila 101.1, zatim bi se pogledalo dali je vrijednost decimalni jednaka 0 ako nije postupak se ponavlja. Kad se krene iz početka decimalni dobiva novu vrijednost 0.5625 te se tako radnja ponavlja sve dok decimalni nije 0 te bi se tu ispisao rezultat prvog zadatka.

Petlja for će se izvršavati 23 puta iz razloga što se u funkciji IEEE754 definira Broj brojeva u mantisi = 23. Razlog je taj što se u toj petlji se računa mantisa koja će biti objašnjena kod prikaza realnog broja na računala u IEEE754 standardu. Za pretvaranje realnog broja u binarni sustav je samo bitno da će se rezultat ispisati samo jednom kada se izračuna zbog toga što kad jednom program uđe u if varijabla d koja je uvjet za ulazak u if će se promijeniti na 1. Program će ispisati rezultat, a mantisa će se izračunati do kraja.

3.2. Prikaz realnog broja na računalu

3.2.1. Prikaz brojeva jednostruke preciznosti

Kod pohranjivanja jako velikih ili jako malih brojeva bi se zauzelo jako puno memorije te bi računalo sporije obrađivalo te brojeve. Za prikaz i pohranu takvih brojeva koristi se IEEE754 standard. IEEE754 je zapis koji se sastoji polja za pohranu predznaka, polja od 8 bita za pohranu eksponenta i 23 bita za pohranu mantise.

Pod prikaza realnog broja u binarnom brojevnom sustavu prvo što je potrebno napraviti je pomaknuti decimalnu točku na način da broju na početku piše 1., a zatim se prepiše ostatak broja od te jedinice. Ne smije se zaboraviti pomnožiti taj broj sa $2^{eksponent}$. Eksponent se računa na način da se broje mjesta za koja se decimalna točka pomakla. Ako se decimalna točka premješta na desno onda je eksponent pozitivan, a ako se pomiče na lijevo onda je negativan.

Iz tog zapisa se radi IEEE754 zapis. Predznak govori dali je broj pozitivan ili negativan te se označava s 0 ako je pozitivan i 1 ako je negativan te se prvi zapisuje. Eksponent se računa na način da se od 127 zbroji ili oduzme eksponent broja 2 ovisno o tome dali je pozitivan ili negativan te se takav zapisuje u idućih 8 bitova. Mantisa je zapis koji se dobije kada se prepiše broj od 1. pa na danje pa se onda ostala mjesta popune s nulom. Uzmimo za primjer:

$$40.5625_{(10)} = 101000.1001_{(2)} = 1.010001001 \cdot 2^5$$

Predznak: 0

Eksponent: $127 + 5 = 132_{(10)} = 10000100_{(2)}$

Mantisa: $0100\ 0100\ 1000\ 0000\ 0000\ 000$

IEEE754 zapis broja $40.5625_{(10)}$ je: $0\ 1000\ 0100\ 0100\ 0100\ 1000\ 0000\ 0000\ 000$

Pomoću IEEE754 zapis broja mogu se zapisati jako veliki i jako mali brojevi unutar 32 bita što govori činjenica da je najveći pozitivni broj koji se može prikazati ovim putem je :

$$3.402823466 \cdot 10^{38} = (0\ 1111\ 1110\ 1111\ 1111\ 1111\ 1111\ 1111\ 111)$$

, a najmanji je:

$$1.175494351 \cdot 10^{-38} = (0\ 0000\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000\ 000)$$

Broj 0 u IEEE754 sustavu ima dva prikaza: pozitivna i negativna nula. Mantisa i eksponent su nula dok je predznak 0 za pozitivnu, a 1 za negativnu nulu

$$+0 \ = \ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$

$$-0 \ = \ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$

Isto tako je moguće prikazati +/- beskonačno na način da je eksponent 255, a mantisa nula:

$$0\ 1111\ 1111\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 \ = \text{ beskonačno}$$

$$1\ 1111\ 1111\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 \ = \text{ -beskonačno}$$

3.2.2. Prikaz brojeva jednostruke preciznosti

Prikaz dvostruke točnosti se razlikuje od prikaza jednostruke točnosti po tome što može prikazati veći broj. Dvostruka preciznost koristi 11 bitova za eksponent i 52 za mantisu te tako povećava maksimalni prikazivi pozitivni broj na $1.7976931348623158 \cdot 10^{308}$, a najmanji prikazivi pozitivni broj $2.2250738585072014 \cdot 10^{-308}$

3.2.3. Pretvaranje realnog broja u IEEE754 u Pythonu

Python program ima funkciju pretvaranja realnog broja u IEEE754 standard sa jednostrukom preciznošću. Na glavnom izborniku pod brojem 2 je ta funkcija te nakon što se u ODABIR: unese broj 2 i pritisne, od korisnika se traži da unese realni broj koji želi da se pretvori u binarni zapis s pomičnom točkom. Nakon što se unese broj, program ulazi u funkciju IEEE754().

```
1 def IEEE754(n, OD0) :
2
3     predznak = 0
4     if n < 0 :
5         predznak = 1
6         n = n \cdot (-1)
7     Broj_brojeva_u_mantisi = 23
8
```

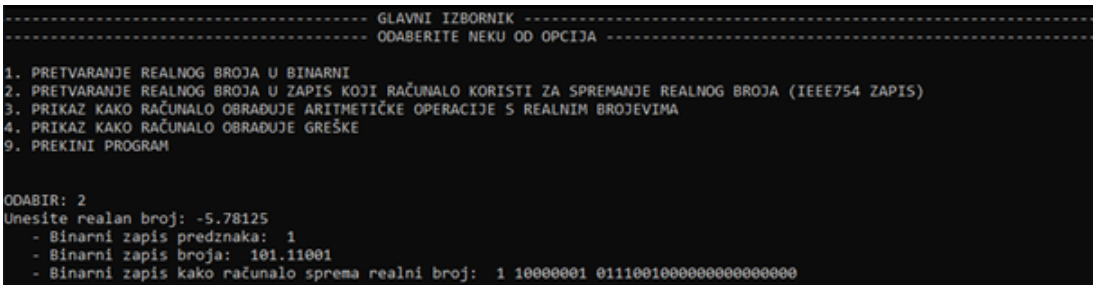
```
    lista = binarni(n, Broj_brojeva_u_mantisi, predznak)

    if OD0==1:
        return

    gdjeTocka = lista.find('.')
    prvaJedinica = lista.find('1')

    if prvaJedinica > gdjeTocka:
        lista = lista.replace(".","")
        gdjeTocka -= 1
        prvaJedinica -= 1
    elif prvaJedinica < gdjeTocka:
        lista = lista.replace(".","")
        gdjeTocka -= 1

    mantisa = lista[prvaJedinica+1:24]

    eksponent = gdjeTocka - prvaJedinica
    eksponentB = eksponent + 127

funkcija remove() ili delete())
    eksponentB = bin(eksponentB).replace("0b",'')

    print("   - Binarni zapis kako racunalo sprema realni broj: ", str(predznak),
    eksponentB.zfill(8), mantisa)
```

Na početku funkcije se provjerava dali je uneseni broj pozitivan ili negativan. Ako je pozitivan predznak ostaje 0, a ako je broj negativan predznak se mijenja u 1 te se broj množi sa -1 kako bi dobili pozitivan broj koji se dalje obrađuje. Nakon što se dobije cijeli broj, ulazi se u funkciju binarni() koja je prethodno objašnjena. Funkcija binarni() vraća realni broj u binarnom brojevnom sustavu koji se sprema u varijablu lista. Iza decimalne toče se prvo zapisuje rezultat decimalnog dijela pretvorbe broja te se nakon toga zapisuju nule sve do dvadeset i trećeg znaka. Jedini razlog popunjavanja mjesta s nulama je taj što broj iza decimalne točke predstavlja mantisu.

Dalje u kodu se traži indeks mjesta gdje se nalazi točka i indeks mjesta gdje se nalazi prva točka. Recimo za broj $5.5625_{(10)} = 101.1001_{(2)}$ prva jedinica je na prvom mjestu, odnosno, ima indeks 0 dok točka se nalazi na četvrtom mjestu te ima indeks 3. U if-u se provjerava koji je veći te ako jedinica ima manji indeks, iz liste se briše decimalna točka te se indeks točke smanjuje za jedan, a ako je indeks mjesta točke manji od indeksa prve jedinice, onda se briše točka te smanjuju oba indeksa za jedan. Jedini razlog smanjivanja indeksa je da se izračunaju točan eksponent i mantisa.

Mantisa se dobije na način da se, od binarnog broja koji je sad zapisan bez decimalne točke u varijablu lista, prepiše cijela lista, ali od indeksa broj 1. Npr, za broj $5.5625_{(10)} = 101.1001_{(2)}$ bez decimalne točke izgleda $1011001_{(2)}$ što znači da se u mantisu upisuje 011001 Razlog je taj što se u mantisu upisuje broj od decimalne točke na dalje, u ovom slučaju je broj s indeksom 1 pa sve do indeksa 24.

Eksponent se računa na način da se prvo oduzmu indeks prve jedinice od indeksa točke, zatim se rezultat oduzimanja zbraja s brojem 127, a zadnji korak kako bi se dobio željeni oblik eksponenta je da se prvo funkcijom bin(eksponentB) pretvori dekadski rezultat zbrajanja pretvori u binarni brojevni sustav. Funkcija bin() prije binarnog broja ispisuje „0b“ kako bi indicirao kako se radi o brojevnom sustavu, npr za bin(9) bi ispis bio „0b1001“, zato se uz funkciju bin() piše .replace(„0b“, ’ ’) kako bi se 0b obrisao. Zadnja linija koda funkcije IEE754 je ispis finalnog rezultata sljedećim redoslijedom: prvi bit koji predstavlja predznak, binarni zapis eksponenta (funkcija zfill(8) definira da taj broj treba biti zapis od 8 znamenki što znači da ako je nekim slučajem u eksponentB zapisan eksponent 101, zfill(8) funkcija će zapisati nule kako bi se dobio broj od 8 znamenki što bi rezultiralo brojem 00000101) i mantisa. Realizacija i izgled izračuna zapisa realnog broja u IEE754 oblik broja je prikazano na sljedećoj slici:



Slika 3: Rezultat funkcije 2

4. Aritmetičke operacije

Aritmetika je znanost o brojevima te kao najosnovanija grana matematika se bavi računanjem brojeva korištenjem aritmetičkih operacija: zbrajanje, oduzimanje, množenje i dijeljenje. Podrijetlo riječi aritmetika dolazi od grčke riječi "Arifmos" što znači "broj". Otkrivanje aritmetike te njen razvitak su započeli Grci, Arapi i Indijci, a kasnije se aritmetika razvijala u Europi sve do aritmetike koju danas poznajemo. Od samih početaka pa sve do danas su se razvile mnoge aritmetičke grane, a jedna od njih je i binarna aritmetika.

Binarna aritmetika nije ništa drugo nego aritmetika za računala. Računala pomoću niza naredbi obrađuje niz nula i jedinica koje vanjskom svijetu predstavljaju podatke. Aritmetičke operacije se u računalu izvršavaju u procesoru, točnije u aritmetičko-logičkoj jedinici. U procesor se podatci spremaju u internu memoriju te ovisno o naredbama iz upravljačke jedinice se ti podatci obrađuju. Iako aritmetika obrađuje brojeve aritmetičkim operacijama zbrajanja, oduzimanja, množenja i dijeljenja, aritmetičke operacije su nešto drugačije u računalu. Naime, računalo može samo zbrajati. Iz tog razloga se ostale aritmetičke operacije u računalu izvršavaju na način da se podatci mijenjaju kako bi se zbrajanjem dobio željeni rezultat. U nastavku će svaka aritmetička operacija u računalu biti detaljno objašnjena.

4.1. Programska realizacija dijela za aritmetičke operacije u Pythonu

Do programskih rješenja aritmetičkih operacija se dolazi na način da se u glavnom izborniku odabere opcija 3. PRIKAZ KAKO RAČUNALO OBRAĐUJE ARITMETIČKE OPERACIJE S REALNIM BROJEVIMA. Odabirom treće opcije se otvara pod izbornik u kojem se odabire željena aritmetička operacija upisivanjem u ODABERITE OPERACIJU: brojeve 1, 2, 3 ili 4. Uneseni odabir se sprema u varijablu ODABIR2 te pomoću te varijable i if funkcije se odabire željena aritmetička operacija. Svaki od aritmetičkih operacija se sastoji od istog linijskog koda samo se razlikuju po malim sitnicama kod ispisa.Recimo ako se odabere opcija broj 1, to jest, zbrajanje. Odabirom te funkcije se od korisnika traži da unese brojeve. Nakon unosa brojeva program pretvara te brojeve u binarni zapis i IEEE754 zapis te nakon toga uneseni brojevi i uneseni ODABIR2 ulaze u funkciju pretvoriUB().

```
1  if ODABIR2 in ('1', '2','3','4','9'):
2              if ODABIR2 == '1':
3                  print("\n------------------")
4                  print("--------------ODABERALI STE ZBRAJANJE-------------")
5                  unos1=float(input("Unesite prvi realan broj: "))
6                  unos2=float(input("Unesite drugi realan broj: "))
7                  print("\nPretvaranje realnog broja ",unos1," u racunalni IEEE754
       zapis:\n")
8                  IEEE754(unos1,0)
9                  print("\nPretvaranje realnog broja ",unos2," u racunalni IEEE754
       zapis:\n")
10                 IEEE754(unos2,0)
```

```
                print("\n-KOD ARITMETICKIH OPERACIJA, koriste se brojevi prikazani s
    eksponentom broja 2 \n")
                ODABIR2=1
                pretvoriUB(unos1,unos2, ODABIR2)
```

Ulaskom u funkciju pretvoriUB() se uneseni brojevi pripremaju za aritmetičke operacije: određuju se predznaci, binarni broj nakon pomicanja decimalne točke i eksponent broja 2 nakon pomicanja decimalne točke.

```
def pretvoriUB(u1,u2,ODABIR3):

    print("\nPrvi broj: ",u1)
    predznak1 = 0
    if u1 < 0 :
        predznak1 = 1
        u1 = u1 \cdot (-1)

    cijeli1, decimalni1 = str(u1).split(".")
    a1=len(decimalni1)
    cijeli1 = int(cijeli1)
    li1 = (str(bin(cijeli1))+".").replace('0b','')
    dec1=0
    d1=0
    for x1 in range(a1):
        decimalni1 = str('0.')+str(decimalni1)
        dec1 = (float(decimalni1)\cdot2)
        cijeli1, decimalni1 = str(dec1).split(".")
        li1 += cijeli1
    print("   - Binarni zapis predznaka:",predznak1)
    print("   - Binarni zapis realnog broja: ",li1)


    gdjeTocka1 = li1.find('.')
    prvaJedinica1 = li1.find('1')


    if prvaJedinica1 > gdjeTocka1:
        li1 = li1.replace(".","")
        varijablaA=str("1")+str(".")+li1[prvaJedinica1+1:]+str("0")
        eksponentA=-(prvaJedinica1-1)
        print("   - Uneseni broj nakon pomicanja zareza sa eksponentom broja 2: ",
    varijablaA, " * 2^(",eksponentA,")")
    elif prvaJedinica1 < gdjeTocka1:
        li1 = li1.replace(".","")
        varijablaA=str("1")+str(".")+li1[prvaJedinica1+1:]
        eksponentA= gdjeTocka1-1
        print("   - Uneseni broj nakon pomicanja zareza sa eksponentom broja 2: ",
    varijablaA, "* 2^(",eksponentA,")")
```

```
    print("\nDrugi broj: ",u2)
    predznak2 = 0
    if u2 < 0 :
        predznak2 = 1
        u2 = u2 \cdot (-1)

    cijeli2, decimalni2 = str(u2).split(".")
    a2=len(decimalni2)
    cijeli2 = int(cijeli2)
    li2 = (str(bin(cijeli2))+".").replace('0b','')
    dec2=0
    d2=0
    for x2 in range(a2):
        decimalni2 = str('0.')+str(decimalni2)
        dec2 = (float(decimalni2)\cdot2)
        cijeli2, decimalni2 = str(dec2).split(".")
        li2 += cijeli2
    print("   - Binarni zapis predznaka:",predznak2)
    print("   - Binarni zapis realnog broja: ",li2)


    gdjeTocka2 = li2.find('.')
    prvaJedinica2 = li2.find('1')


    if prvaJedinica2 > gdjeTocka2:
        li2 = li2.replace(".","")
        varijablaB=str("1")+str(".")+li2[prvaJedinica2+1:]+str("0")
        eksponentB=-(prvaJedinica2-1)
        print("   - Uneseni broj nakon pomicanja zareza sa eksponentom broja 2: ",
    varijablaB, " • 2^(",eksponentB,")")
    elif prvaJedinica2 < gdjeTocka2:
        li2 = li2.replace(".","")
        varijablaB=str("1")+str(".")+li2[prvaJedinica2+1:]
        eksponentB= gdjeTocka2-1
        print("   - Uneseni broj nakon pomicanja zareza sa eksponentom broja 2: ",
    varijablaB, "• 2^(",eksponentB,")")

    if(ODABIR3==1):
        zbrajanje(varijablaA, eksponentA, varijablaB, eksponentB)
        print("Sto bi u dakadskom brojevnom sustavu bilo: ", u1+u2,"\n\n")
    elif(ODABIR3==2):
        oduzimanje(varijablaA, eksponentA, varijablaB, eksponentB)
        print("Sto bi u dakadskom brojevnom sustavu bilo: ", u1-u2,"\n\n")
    elif(ODABIR3==3):
        mnozenje(varijablaA, eksponentA, varijablaB, eksponentB, predznak1,
    predznak2)
        print("Sto bi u dakadskom brojevnom sustavu bilo: ", u1\cdotu2,"\n\n")
    elif(ODABIR3==4):
        djeljenje(varijablaA, eksponentA, varijablaB, eksponentB, predznak1,
    predznak2)
        print("Sto bi u dakadskom brojevnom sustavu bilo: ", u1/u2,"\n\n")
```

Nakon što se odrede predznaci, binarni brojevi nakon pomicanja decimalne točke i eksponenti, ovisno o odabiru iz prošle funkcije program se nastavlja izvoditi u jednoj od navedenih funkcija: zbrajanje(), oduzimanje(), mnozenje() i djeljenje(). U svaku funkciju se prenose prethodno izračunate vrijednosti iz funkcije pretvoriUB(). Svaka od aritmetičkih funkcija će biti detaljno objašnjene u nastavku.

4.2. Zbrajanje

Zbrajanje prirodnih brojeva je jednostavan proces. Za početak uzmimo jednoznamenkaste binarne brojeve. Moguće su četiri kombinacije zbrajanja:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$1 + 1 = 10$ (pisati 0,a pamti 1)

Zbrajanje višeznamenkasti binarnih brojeva je u principu isto kao i kod dekadskih, brojevi se potpišu jedan ispod drugoga te se zbraja znamenku po znamenku, uzimajući u obzir prethodno napisane jednakosti. Uzmimo na primjer dva broja: $10_{(10)} + 14_{(10)}$:

$$10_{(10)} = 1010_{(2)}$$

$$14_{(10)} = 1110_{(2)}$$

$$\begin{array}{r} 1010 \\ +1110 \\ \hline 11000 \end{array}$$

$$11000_{(2)} = 24_{(10)}$$

Zbrajanje realnih brojeva je nešto malo drugačije iz razloga što je potrebno koristiti metodu pomičnog zareza. Kako bi se izračunali takvi brojevi potrebno je decimalnu točku pomaknuti do prve jedinice s lijeve strane na način da da bude „1.“ i ovisno za koliko mjesta se pomogne točka toliki će biti eksponent na broju 2. Isto tako je bitno u koju stranu se pomiče točka, ako se točka pomiče prema lijevo eksponent broja 2 je u plusu, a ako se pomiče u desno onda je eksponent negativan. Kada se odrede eksponenti pronalazi se koji je od njih veći. Kada se pronađe koji je veći, onaj manji se treba izjednačiti s većim. Npr:

$$0.03125_{(10)} + 7.5625_{(10)} = 0.00001_{(2)} + 111.1001_{(2)}$$

$$0.00001_{(2)} = 1.0 \cdot 2^{-5}$$

$$111.1001_{(2)} = 1.111001 \cdot 2^{2}$$

Prvi broj se treba prilagoditi drugom na način da se decimalna točka pomakne za 7 mjesta u lijevo te se nakon toga mogu zbrojiti:

$$0.0000001_{(2)} \cdot 2^{2}$$

$$\begin{array}{r} 0.0000001 \\ +1.111001 \\ \hline 1.1110011 \end{array}$$

Nakon zbrajanja, dobivenom broju treba vratiti decimalnu točku za onoliko mjesta koliko eksponent broja 2 to zahtjeva te se tako dolazi do krajnjeg rješenja:

$$1.1110011 \cdot 2^{2} = 111.10011_{(2)} = 7.59375_{(10)}$$

4.2.1. Programska realizacija dijela za aritmetičku operaciju zbrajanja u Pythonu

Do aritmetičke operacije zbrajanja se dolazi na način da se u glavnom izborniku u ODABIR: unese broj 3. Tom opcijom se otvara novi izbornik koji nudi 4 aritmetičke operacije na odabir, ali za zbrajanje nas zanima odabir pod brojem jedan: 1.ARITMETIČKA OPERACIJA ZBRAJANJA. Zatim se od korisnika traže da se unesu dva broja. Kako je planirano da program ispisuje sve informacije vezane uz unesene brojeve, uneseni brojevi se prvo pretvaraju u IEEE754, a zatim i ispisuju kako bi se mogla provjeravati točnost rješenja.

Ulaskom u funkciju pretvoriUB() se svaki broj zasebno pretvara u binarni kao u funkciji binarni(), jedina razlika je u tome što se uz ispis Binarni zapis predznaka: i Binarni zapis realnog broja: ispisuje i Uneseni broj nakon pomicanja zareza sa eksponentom broja 2.Kada se oba unesena broja pretvore u binarne brojeve s eksponentom broja 2 ulazi se u drugu funkciju zbrajanje().

```
1
2 print("\n\nSljedece sto trebamo napraviti je usporediti eksponente i pronaci koji je
      veci")
3 print("Zatim broj sa manjim eksponentom se treba prilagoditi vecem")
4 print("   - Prvi uneseni broj nakon pomicanja zareza sa eksponentzom broja 2: ",
    varijablaA, "•2^(",eksponentA,")")
5 print("   - Drugi uneseni broj nakon pomicanja zareza sa eksponentzom broja 2: ",
    varijablaB, "• 2^(",eksponentB,")")
6
7 if eksponentA > eksponentB:
8     print("\n\nVidimo da je eksponent prvog broja veci te se treba drugoga
    prilagoditi prvome:")
9 elif eksponentA < eksponentB:
```

```
    print("\n\nVidimo da je eksponent drugoga broja veci te se treba prvi
    prilagoditi drugome:")
elif eksponentA == eksponentB:
    print("\n\nVidimo da su eksponenti jednaki te se moze dalje racunati:")
```

U funkciji zbrajanje() prvo se provjerava čiji broj ima veći eksponenta na bazi 2.,a nakon toga slijedi ispis rezultat provjere.

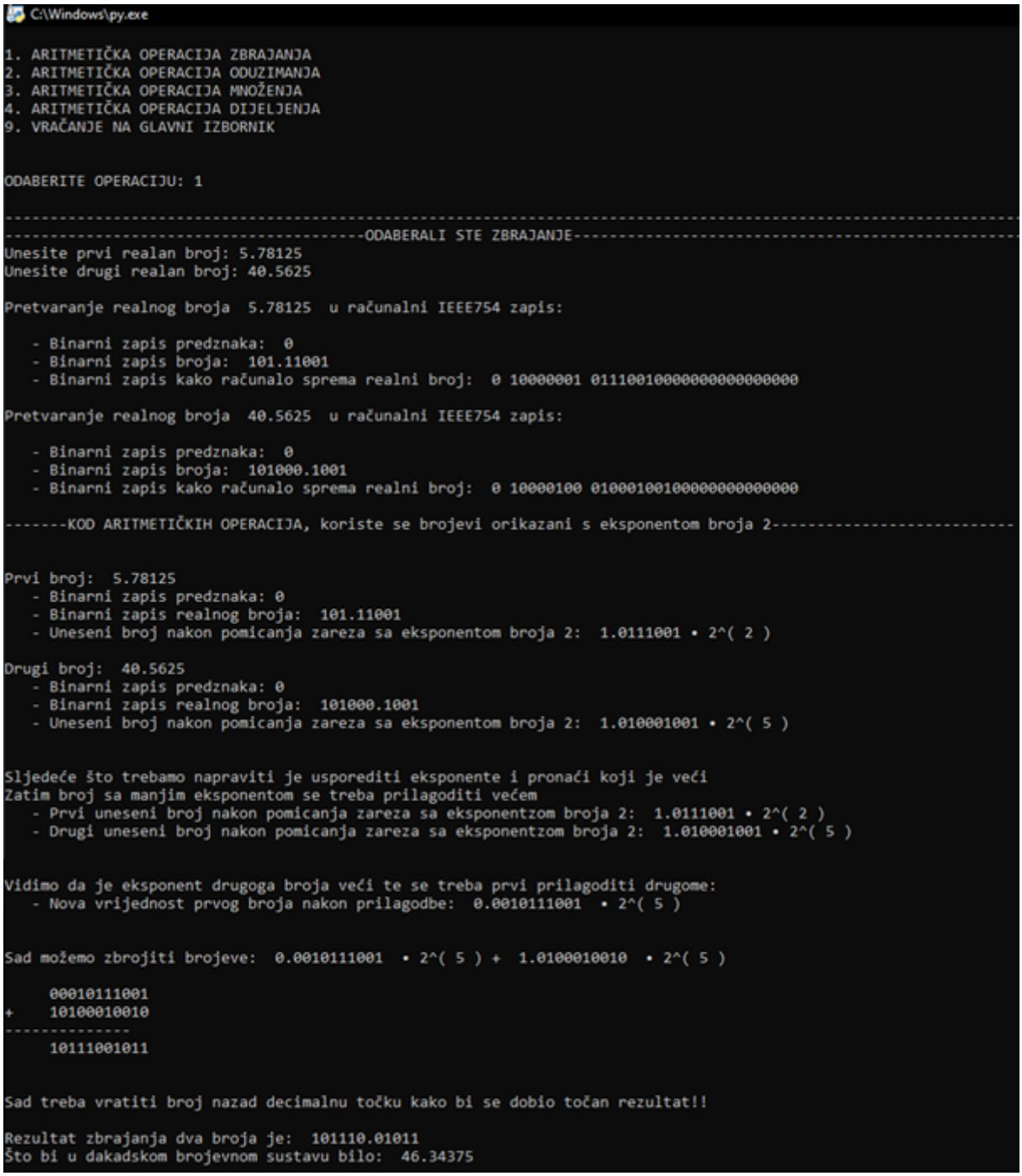
```
eA=eksponentA
eB=eksponentB
vA=varijablaA
vB=varijablaB

if eksponentA > eksponentB:
    razlikaEksponenata=eksponentA-eksponentB
    eksponentB=eksponentA
    for x in range(razlikaEksponenata):
        vB = varijablaB.replace(".","")
        varijablaB=str("0")+str(".")+vB
    print("   - Nova vrijednost drugog broja nakon prilagodbe: ", varijablaB, " •
    2^(",eksponentB,")")


    elif eksponentA < eksponentB:
        razlikaEksponenata=eksponentB-eksponentA
        eksponentA=eksponentB
        for x in range(razlikaEksponenata):
            vA = varijablaA.replace(".","")
            varijablaA=str("0")+str(".")+vA
        print("   - Nova vrijednost prvog broja nakon prilagodbe: ", varijablaA, "
    • 2^(",eksponentA,")")
```

Broj sa manjim eksponentom se mijenja na način da eksponent bude jednak onom većem što je objašnjeno u prethodnom poglavlju. Nova vrijednost se ispisuje te su brojevi spremni za zbrajanje nakon što im se još obrišu decimalne točke. Dio programa koji zbraja binarne brojeve prilično jasan sam po sebi te se rezultat zapisuje ispod podvučene crte, a kasnije i finalni rezultat kada se vrati decimalna točka.

```
if eksponentA==eksponentB:
    print("\n\nSad mozemo zbrojiti brojeve: ", varijablaA, " • 2^(",eksponentA,") +
    ", varijablaB, " • 2^(",eksponentB,")")


    for x in range(0,len(varijablaA)):
        vA = varijablaA.replace(".","")
        vB = varijablaB.replace(".","")

    print("\n    ", vA)
    print("+   ", vB)
    print("--------------")

    finalniRez=""
    jedanDalje=int(0)
```

```
15
16      for y in range(0,len(vA)):
17
18          x=y\cdot(-1)-1
19          rezultat=int(vA[x])+int(vB[x])+jedanDalje
20
21          jedanDalje=0
22          if rezultat==0:
23              finalniRez=str('0')+ finalniRez
24
25          elif rezultat==1:
26              finalniRez=str('1')+ finalniRez
27
28          elif rezultat==2:
29              finalniRez=str('0')+ finalniRez
30              jedanDalje=1
31
32          elif rezultat==3:
33              finalniRez=str('1')+ finalniRez
34              jedanDalje=1
35
36
37      if jedanDalje==1:
38          finalniRez=str('1')+ finalniRez
39          print("   ", finalniRez)
40          jedanDalje=0
41          vC=finalniRez[:eksponentA+2]+str(".")+finalniRez[eksponentA+2:]
42      else:
43          print("    ", finalniRez)
44          vC=finalniRez[:eksponentA+1]+str(".")+finalniRez[eksponentA+1:]
45
46      print("\n\nTreba vratiti broj nazad decimalnu tocku kako bi se dobio tocan
        rezultat!!")
47
48      print("\nRezultat zbrajanja dva broja je: ",vC)
```

Kada je program pokrenut i kada se računa aritmetička operacija zbrajanja, korisnik aplikacije će imati uvid u sve informacije vezane uz brojeve kao što su: predznak, binarni broj unesenog broja, IEEE754 izgled unesenog broja, ispis unesenog broja s pomičnim zarezom, postupak zbrajanja, rezultat zbrajanja i provjera rezultata. Čini se kao puno informacija, ali i cilj je bio da se aritmetika prikazuje korak po korak. Na sljedećoj slici je prikazan zbroj dva broja, npr.$5.78125_{(10)} + 40.5625_{(10)}$



Slika 4: Aritmetička operacija zbrajanja

4.3. Oduzimanje

Oduzimanje binarnih brojeva se zapravo svodi na zbrajanje pomoću metode dvojnog komplementa. Metodom dvojnog komplementa se zapisuju negativni brojevi u računalu. Metoda dvojnog komplementa se vrši na umanjitelju na način da se broj prvo invertira, to jest, u binarnom broju se nule zamjene sa jedinicama, a jedinice sa nukama. Zatim se taj broj zbraja sa 1 te se tako dobija dvojni komplement broja koji je spreman za zbajanje sa umanjenikom. Za primjer: $14_{(10)} - 10_{(10)}$

$$14_{(10)} = 1110_{(2)}$$

$$10_{(10)} = 1010_{(2)}$$

Slijedeći korak je binarnom broju 1010 zamijeniti znamenke (jedinice sa nulama i nule sa jedinicama) i taj broj zbrojiti s 1:

$$1010_{(2)} = 0101_{(2)}$$

$$\begin{array}{r} 0101 \\ +\ 1 \\ \hline 0110 \end{array}$$

zbrajanjem broja 1110 i dobivenog dvojnog komplementa 0110 se dobiva razlika zadanih brojeva:

$$\begin{array}{r} 1110 \\ +0110 \\ \hline 10100 \end{array}$$

U ovom primjeru se može vidjeti kako je rezultat zbrajanja $10100_{(2)} = 20_{(10)}$ što nije točan rezultat oduzimanja. Zato kod oduzimanja se jedinacia koja se prenosi na kraju zanemaruje(1 | 0100) tako da je rezultat zapravo točan rezultat $100_{(2)} = 4_{(10)}$.

Kod oduzimanja realnih brojeva, kao i kod zbrajanja, prvo se decimalna točka pomiče do prve jedinice s lijeve strane na način da da bude „1.“ i ovisno za koliko mjesta se pomasgne točka toliki će biti eksponent na broju 2, zatim se brojevi izjednačavaju da eksponenti brojeva 2 budu jednaki te se računa dvojni komplement drugog broja. Npr, $7.5625_{(10)} - 3.625_{(10)}$:

$$7.5625_{(10)} - 3.625_{(10)} = 111.1001_{(2)} - 11.101$$

$$111.1001_{(2)} = 1.111001 \cdot 2^2$$

$$11.101_{(2)} = 1.1101 \cdot 2^1 = 0.11101 \cdot 2^2$$

Na redu je dvojni komplement drugog broja te nakon toga zbrajanje brojeva:

$$0.11101 = 1.00010 + 1 = 1.00011$$

$$\begin{array}{r} 1.111001 \\ +1.00011 \\ \hline 1 \quad 0.111111 \end{array}$$

Za kraj se decimalna točka pomiče za vrijednos eksponenta broja 2, u ovom slučaju za 2 mjesta u desno te je rezultat:

$$0.111111 \cdot 2^2 = 11.1111_{(2)} = 3.9375_{(10)}$$

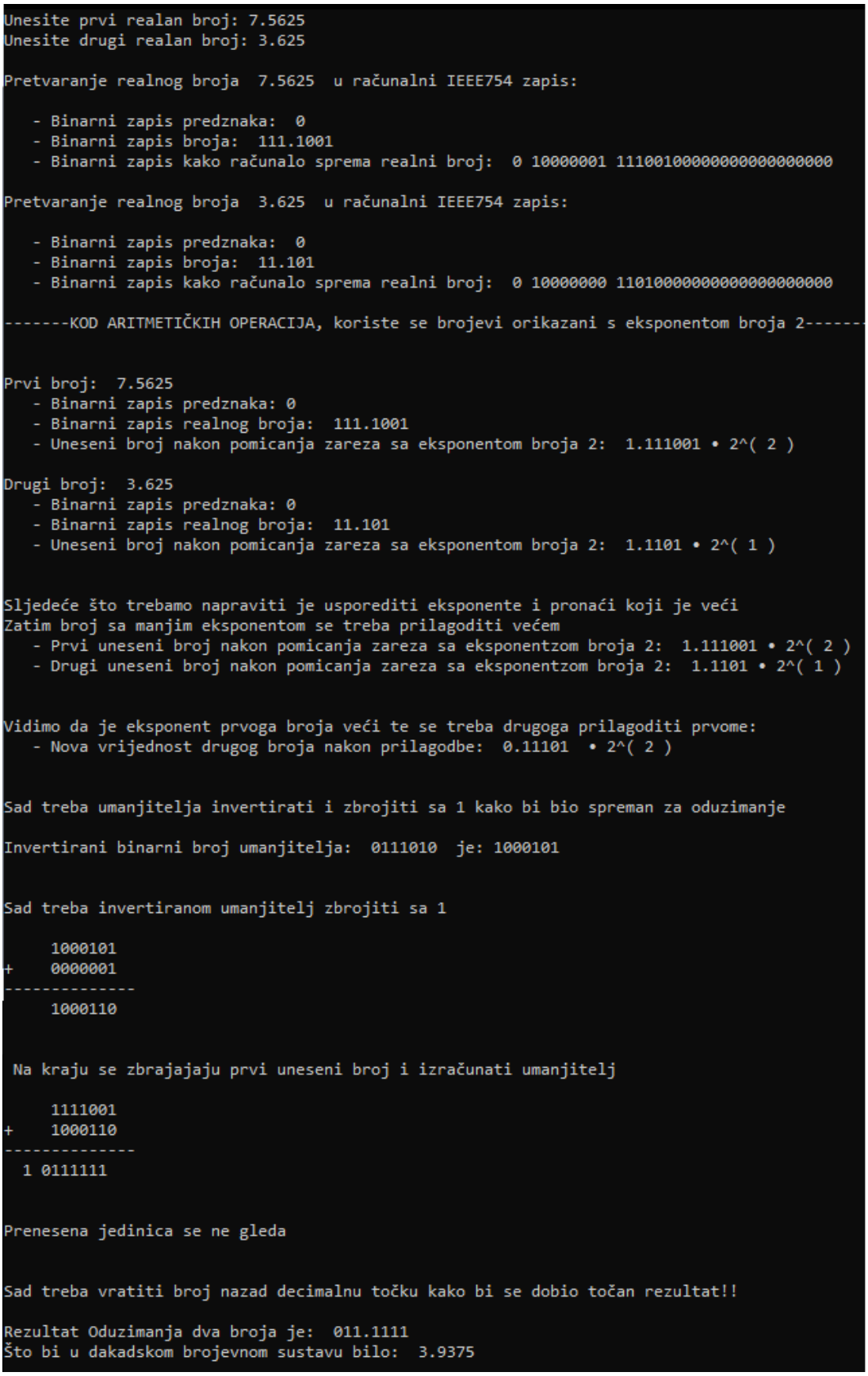
4.3.1. Programska realizacija dijela za aritmetičku operaciju oduzimanja u Pythonu

Do aritmetičkih operacija se dolazi na način da se u glavnom izborniku u ODABIR: unese broj 3 zatim kako bi se odabrala operacija oduzimanja se u novootvorenom izborniku u ODABERITE OPERACIJU se upisuje broj 2. Zatim se od korisnika zahtjeva da upiše dva broja koja želi oduzeti. Programska opcija za zbrajanje i programska opcija za oduzimanje su praktički isti. Nakon unosa se ispisuje Binarni zapis predznaka, Binarni zapis realnog broja i Uneseni broj nakon pomicanja zareza sa eksponentom broja 2. Kod za usporedbu eksponenata i izjednjačavanje eksponenata je isti kao i u operaciji zbrajanja. Kod se razlikuje samo po tome što se umanjitelj invertira i zbraja s 1 te ako postoji jednica preljeva ona se briše:

```
for x in range(0,len(varijablaA)):
    vA = varijablaA.replace(".","")
    vB = varijablaB.replace(".","")
    umanjitelj=str('')

    for z in range(0, len(vB)):
        x=z\cdot(-1)-1
        if vB[x]=='0':
            umanjitelj=str('1')+umanjitelj
        elif vB[x]=='1':
            umanjitelj=str('0')+umanjitelj

    jedinica=str('1')
    for i in range(1,len(vB)):
        jedinica=str('0')+jedinica

    print("\nInvertirani binarni broj umanjitelja: ",vB," je:", umanjitelj)
    print("\n\nSad treba invertiranom umanjitelj zbrojiti sa 1")

    print("\n    ", umanjitelj)
    print("+   ", jedinica)
    print("--------------")
```

```
23
24      finalniUmanjitelj=""
25      jedanDalje=int(0)
26
27      for y in range(0,len(umanjitelj)):
28
29          x=y\cdot(-1)-1
30          rezultat=int(umanjitelj[x])+int(jedinica[x])+jedanDalje
31
32          jedanDalje=0
33          if rezultat==0:
34              finalniUmanjitelj=str('0')+ finalniUmanjitelj
35
36          elif rezultat==1:
37              finalniUmanjitelj=str('1')+ finalniUmanjitelj
38
39          elif rezultat==2:
40              finalniUmanjitelj=str('0')+ finalniUmanjitelj
41              jedanDalje=1
42
43          elif rezultat==3:
44              finalniUmanjitelj=str('1')+ finalniUmanjitelj
45              jedanDalje=1
46      if jedanDalje==1:
47          finalniUmanjitelj=str('1')+ finalniUmanjitelj
48          print("   ", finalniUmanjitelj)
49          jedanDalje=0
50          vC=finalniUmanjitelj[:eksponentA+2]+str(".")+finalniUmanjitelj[eksponentA
    +2:]
51      else:
52          print("    ", finalniUmanjitelj)
53          vC=finalniUmanjitelj[:eksponentA+1]+str(".")+finalniUmanjitelj[eksponentA
    +1:]
54
55      finalniRez=""
56      jedanDalje=int(0)
57      print("\n\n Na kraju se zbrajajaju prvi uneseni broj i izracunati umanjitelj")
```

Zbrajanje prvog broja i izračunatoga dvojnoga komplementa je isti kao kod zbrajanja. Realizacija oduzimanja u Pythonu za npr., $7.5625_{(10)} - 3.625_{(10)}$ izgleda kao na sljedećoj slici:



Slika 5: Aritmetička operacija oduzimanja

4.4. Množenje

Množenje dvaju binarnih brojeva je dosta slično kao i kod dekadskih brojeva na način da se rješavanje svodi na zbrajanje. Kod binarnog množenja se gledaju znamenke drugog faktora, ako je znamenka drugog faktora 1 onda se prvi faktor prepisuje ispod za zbrajanje,a ako je znamenka drugog faktora 0 onda se spušta nula. Npr., $14_{(10)} \cdot 101_{(10)}$

$$14_{(10)} = 1110_{(2)}$$

$$101_{(10)} = 1100101_{(2)}$$

$$1110 \cdot 1100101$$

$$\begin{array}{r} 1110 \\ 0000 \\ 0000 \\ 1110 \\ 0000 \\ +1110 \\ \hline 10110000110 = 1414_{(10)} \end{array}$$

Kod množenje realnih brojeva se naglasak se stavlja da broj decimalnih mjesta i predznak. Predznak rješenja je pozitivan ako su predznaci oba faktora + ili ako su predznaci oba faktora -, dok je negativan predznak rješenja ako je prvi faktor pozitivan i drugi negativan ili ako je prvi faktor negativan i drugi pozitivan. Broj decimalnih mjesta se dobiva na način da se broj decimalnih mjesta prvog faktora zbroji sa brojem decimalnih mjesta drugog faktora.

$$3.5_{(10)} = 11.1_{(2)}$$

$$12.625_{(10)} = 1100.101_{(2)}$$

$$111 \cdot 1100101$$

$$\begin{array}{r} 111 \\ 000 \\ 000 \\ 111 \\ 000 \\ +111 \\ \hline 101100.00110.11 = 44.1875_{(10)} \end{array}$$

4.4.1. Programska realizacija dijela za aritmetičku operaciju množenja u Pythonu

Programsko rješenje za aritmetičku operaciju množenja je dosta opširna iz razloga što je većina koda napisana kako bi se prikaz rješavanja i rješenja prikazao na što uredniji i što čitljiviji način. U funkciju množenje() ulaze prethodno izračunate vrijednosti predznaka, binarnog zapisa brojeva i eksponenta unesenih brojeva. Bitan dio funkcije je for petlja u kojoj se iz binarne vrijednosti drugog faktora zasebno provjeravaju znamenke s lijeva na desno.

```
for y in range(1, len(vB)):

    vAZaIspis=str(' ')+vAZaIspis
    akoSeMnoziSNulom=str(' ')+akoSeMnoziSNulom

    if vB[y]=='1':
        print(vAZaIspis)
        znamenka=1
        rezultat=zbroj(rezultat,vA,znamenka)

    elif vB[y]=='0':
        print(akoSeMnoziSNulom)
        znamenka=0
        rezultat=zbroj(rezultat,vA,znamenka)
```

Kada se ustanovi koja je znamenka drugog faktora učitana, ispisuje se broj. Ukoliko je znamenka jedan, na ekranu se ispisuje vrijednost prve varijable u gdje se izvršava zbrajanje, a ako je znamenka nula onda se na ekranu ispisuje niz nula(dužina niza ovisi o količini znamenaka prvog faktora). Program zatim prelazi u funkciju zbroji(). U funkciji zbroji() se zbrajaju trenutni broj ovisno o učitanoj znamenci iz prošle funkcije (za broj jedan zbraja se vrijednost prvog faktora, a za nulu se samo dodaje nula na kraj reda) i rezultat prethodnog zbrajanja. Na taj način se u rezultat sprema zbroj prošlog rezultata i nove varijable te se funkcija zbroji() izvršava onoliko puta koliko ima znamenaka drugog faktora

Kada se u funkciji izvrši for petlja za zbrajanje rezultata, dolazi na red određivanje predznaka rješenja i određivanje mjesta na kojem treba biti dodana decimalna točka te na kraju ispis konačnog rezultata.

```
decimalnoMjesto=eksponentA+eksponentB
if (len(vA)\cdot2)== len(rezultat):
    decimalnoMjesto+=1
if decimalnoMjesto<0:
    tocka=decimalnoMjesto\cdot(-1)
    for a in range (0, tocka):
        rezultat=str('0')+rezultat
    ispis=rezultat[:1]+str(".")+rezultat[1:]
else:
    ispis=rezultat[:decimalnoMjesto+1]+str(".")+rezultat[decimalnoMjesto+1:]

if (predznak1==0 and predznak2==0) or (predznak1==1 and predznak2==1):
    predznak=str('0')
else:
```

```
15      predznak=str('1')
16
17  print("     -Predznak: ",predznak, ",a rezulrtat: ",ispis)
```

Za prikaz programskog rješenja za primjer će se uzeti $7.5625_{(10)} \cdot 3.625_{(10)}$

```
Unesite prvi realan broj: 7.5625
Unesite drugi realan broj: 3.625

Pretvaranje realnog broja  7.5625  u računalni IEEE754 zapis:

   - Binarni zapis predznaka:  0
   - Binarni zapis broja:  111.1001
   - Binarni zapis kako računalo sprema realni broj:  0 10000001 11100100000000000000000

Pretvaranje realnog broja  3.625  u računalni IEEE754 zapis:

   - Binarni zapis predznaka:  0
   - Binarni zapis broja:  11.101
   - Binarni zapis kako računalo sprema realni broj:  0 10000000 11010000000000000000000

-------KOD ARITMETIČKIH OPERACIJA, koriste se brojevi orikazani s eksponentom broja 2--------------------


Prvi broj:  7.5625
   - Binarni zapis predznaka: 0
   - Binarni zapis realnog broja:  111.1001
   - Uneseni broj nakon pomicanja zareza sa eksponentom broja 2:  1.111001 • 2^( 2 )

Drugi broj:  3.625
   - Binarni zapis predznaka: 0
   - Binarni zapis realnog broja:  11.101
   - Uneseni broj nakon pomicanja zareza sa eksponentom broja 2:  1.1101 • 2^( 1 )


   - Prvi uneseni broj nakon pomicanja zareza sa eksponentzom broja 2:  1.111001 • 2^( 2 )
   - Drugi uneseni broj nakon pomicanja zareza sa eksponentzom broja 2:  1.1101 • 2^( 1 )


Brojevi su spremni za množenje

 1111001  *  1110100
  1111001
   1111001
    0000000
     1111001
      0000000
       0000000
-------------
11011011010100

  -Potrebno je zbrojiti eksponente te se nekon toga dodaje decimalna točka na mjestu zbroja eksponenata,
    -Predznak:  0 ,a rezulrtat:  11011.011010100
Što bi u dakadskom brojevnom sustavu bilo:  27.4140625
```

Slika 6: Aritmetička operacija množenja

4.5. Dijeljenje

Od aritmetičkih operacija, djeljenje je najkompliciranije operacija za izvođenje u računalu. Dijeljenje se sastoji od dvije aritmetičke operacije: množenja i oduzimanja. Kod dijeljenja binarnih brojeva postoje pravila za oduzimanje binarnih, ali pošto je prethodno je utvrđeno da računalo od aritmetičkih operacija razumije jedino zbrajanje iz tog razloga se djeljenje treba svesti na zbrajanje. Pošto se binarno i dekadsko dijeljenje računaju na isti način. u računalu se oduzimanje zamjenjuje dvojnim komplementom. Npr., $99_{(10)}/11_{(10)}$

$$99_{(10)} = 1100011_{(2)} \qquad 11_{(10)} = 1011_{(2)}$$

Dvojni komlpement drugog broja je:

$$1011_{(2)} = 0101_{(2)}$$

te se djeljenje izvršava na način da se djelitelj oduzme od prvih znamenaka djeljenika:

$$\begin{aligned} &1100011/1011 = 1 \\ -\ &1011 \end{aligned} \tag{4.1}$$

što bi metodom dvojnog komplementa izgledalo sljedeće:

$$\begin{aligned} &1100011/1011 = 1 \\ +\ &0101 \end{aligned} \tag{4.2}$$

nakon zbrajanja se jedinica preljeva ne gleda kao ni nule poslije nje sve do prve jedinice:

$$\begin{aligned} &1100011/1011 = 1 \\ +\ &0101 \\ \hline &1\ 0001 = 1 \end{aligned} \tag{4.3}$$

Kada se zbroje brojevi te kad se obrišu nule i jedinica prijenosa se dobiva krajnji rezultat zbrajanja dvojnog komplementa, to jest, oduzimanja. Idući korak je da se spusti iduća znamenka djeljenika:

$$\begin{aligned} &1100011/1011 = 1 \\ +\ &0101 \\ \hline &10 \end{aligned} \tag{4.4}$$

pošto se može primijetiti kako je broj znamenaka djelitelja veći od broja znamenaka za daljnje oduzimanje, potrebno je spustiti iduću znamenku djeljenika, ali ovaj put je potrebno zapisati nulu u rezultat:

$$
\begin{array}{r}
1100011/1011 = 10 \\
+\,0101 \\
\hline
10 \\
101
\end{array}
\tag{4.5}
$$

Koraci se ponavljaju dok se ne dobije potreban broj znamenaka, a kada se dobije potreban broj znamenaka u rezultat se zapisuje jedinica te je potrebno zbrojiti dobiveni broj i dvojni komplement djelitelja:

$$
\begin{array}{r}
1100011/1011 = 100 \\
+\,0101 \\
\hline
10 \\
101 \\
1011
\end{array}
\tag{4.6}
$$

$$
\begin{array}{r}
1100011/1011 = 100 \\
+\,0101 \\
\hline
10 \\
101 \\
1011 \\
+\,0101 \\
\hline
\end{array}
\tag{4.7}
$$

$$
\begin{array}{r}
1100011/1011 = 1001 \\
+\,0101 \\
\hline
10 \\
101 \\
1011 \\
+\,0101 \\
\hline
1\ 0000
\end{array}
\tag{4.8}
$$

Pošto je ostatak dijeljenja nula, rezultat je cijeli broj. Rezultat dijeljenja je $1001_{(2)} = 9_{(10)}$.

4.5.1. Programska realizacija dijela za aritmetičku operaciju dijeljenja u Pythonu

Odabirom četvrte opcije u izborniku aritmetičke operacije otvara se opcija Aritmetičke operacije dijeljenja. Kao i u ostalim aritmetičkim operacijama od korisnika se traži da unese dva realna broja te se nakon toga i ispisuju pojedine informacije za svaki broj te se brojevi pretvaraju u zapis sa pomičnim zarezom. Prije nego što program počne djeljenje unesena dva broja, potrebno je provjeriti dali je broj znamenaka drugog broja veći od broja znamenaka prvog broja. Ako je to slučaj, u zasebnu varijablu se zapisuje prvi broj te se dodavanju nule na kraju te varijable kako bi se dobio broj koji bi bio spreman za dijeljenje, ali se za svaku zapisanu nulu piše i u rezultatu nula. Ako je broj znamenaka prve varijable veći od broja znamenaka druge varijable, onda se u privremenu varijablu zapisuje onoliko znamenaka prve varijable koliko ima znamenaka druge varijable. Nakon tog koraka se provjerava dali je vrijednost privremene varijable veća od vrijednosti druge varijable. U slučaju da je vrijednost privremene varijable veća, program nastavlja dalje, a ako to nije slučaj onda se uzima sljedeća vrijednost prve varijable u privremenu varijablu pa tek onda će program krenuti sa dijeljenjem.

```
privremenaVA=''
razmak=''
brojZnamenakaVa=int(0)
decimalnaTocka=int(eksponentA)- int(eksponentB)
rezultat=''

if len(vA)<len(vB):
    privremenaVA=vA
    for x in range(len(vA), len(vB)):
        privremenaVA=privremenaVA+str('0')
        rezultat=rezultat+str('0')
    if privremenaVA<vB:
        privremenaVA=privremenaVA+str('0')
        rezultat=rezultat+str('0')
    brojZnamenakaVa=len(privremenaVA)

else:
    for x in range(0,len(vB)):
        privremenaVA=privremenaVA+vA[x]
        brojZnamenakaVa+=1

    if privremenaVA<vB and len(vA)<=len(vB):
        privremenaVA=privremenaVA+str('0')
    elif privremenaVA<vB and len(vA)>len(vB):
        privremenaVA=privremenaVA+vA[brojZnamenakaVa]
```

Ulaskom u while() petlju započinje dio programa zadužen za dijeljenje. Prije dodatnog pojašnjavanja bitno je reći kako varijabla razmak je samo estetske prirode te sve obrade varijable razmak ne utječe na točnost rezultata. While() petlja je zadana da se ponavlja maksimalno 10 puta iz tog razloga da se program u slučaju određenih dijeljenja ne bi izvršavao u beskonačnost već da nakon 10 ponavljanja se program zaustavi. Unutar petlje su zadane 3 provjere: dali je privremena varijabla veće vrijednosti od vrijednosti drugog unesenog broja,

dali je privremena varijabla manje vrijednosti od vrijednosti drugog unesenog broja i dali su te varijable jednake. U jednom ciklusu while() petlje se izvrši samo jedna od tih provjera, a nakon pojašnjenja bit će jasno i zašto.

Ako je privremena varijabla veće vrijednosti od vrijednosti drugog unesenog broja to znači da se te vrijednosni mogu oduzeti te se rezultat zapisati u privremenu varijablu s kojom će se računati u idućem ciklusu petlje, a u rezultat upisati broj jedan. Ako je privremena varijabla manje vrijednosti od vrijednosti drugog unesenog broja to znači da se u rezultat zapisuje nula. Ako su varijable jednake u privremenu varijablu se zapisuje nula, a u rezultat se zapisuje jedan. Bitno je za naglasiti da na kraju while() petlje, to jest, prije ulaska u idući ciklus se izvršava program za spuštanje iduće znamenke, što znači da se privremenoj varijabli dodaje iduća znamenka prvog broja ili se zapisuje 0 ukoliko je ostao ostatak, a od prvog broja su iskorištene sve vrijednosti te se započinjem zapisivati decimalni dio rješenja.

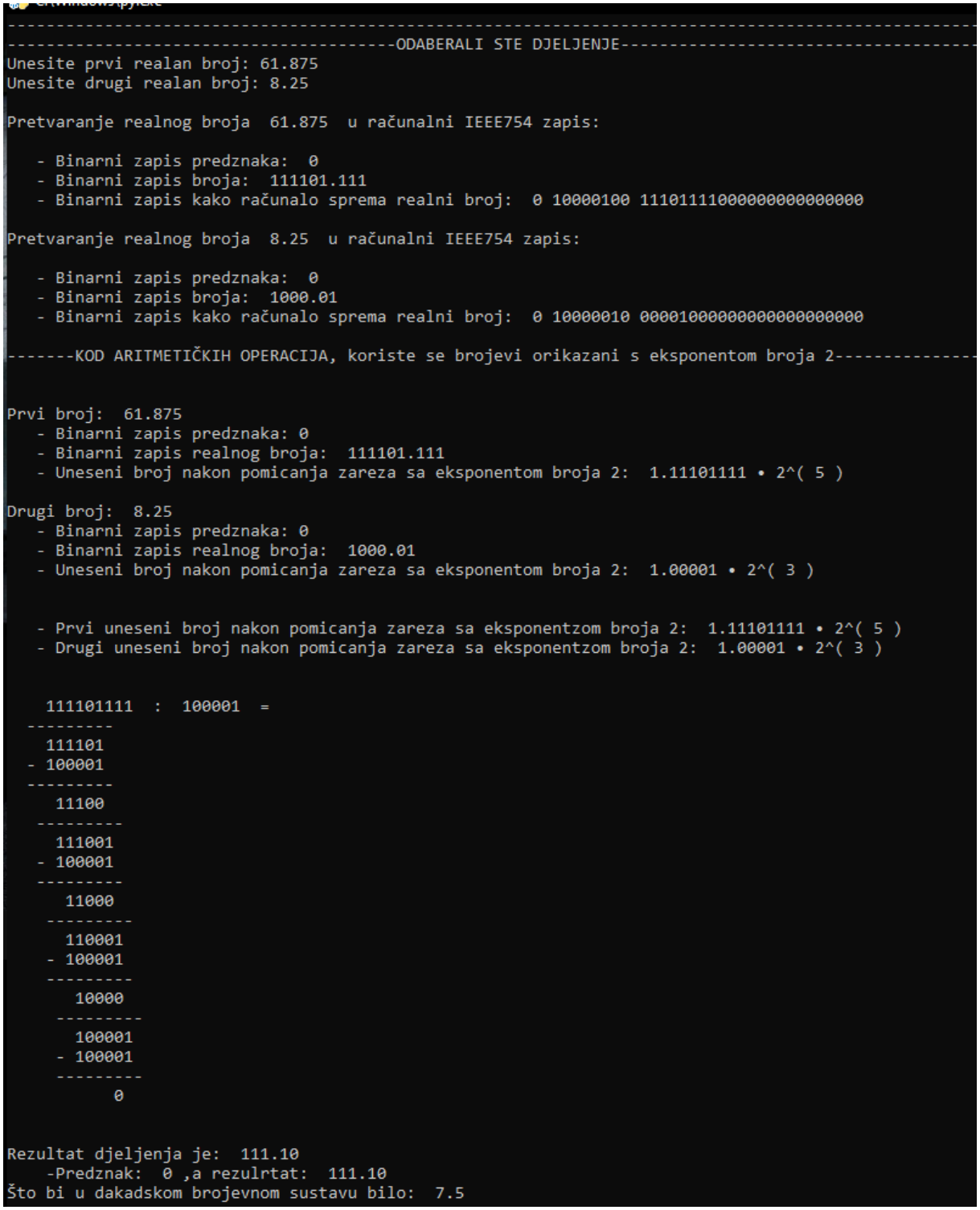
Između provjere i dopisivanja znamenaka u privremenu varijablu se nalazi dio koda koji je zadužen za provjeru dali je dijeljenje završeno u smislu da su sve vrijednosti prvog unesenog broja iskorištene i dali je ostatak nula, ako su obje tvrdnje istinite onda se zapisuje konačan rezultat i prekida se izvršavanja while() petlje.

```
while gotovo<10:
    if privremenaVA>vB:
        print(razmak,"---------")
        print(razmak," ",privremenaVA)
        print(razmak,"-",vB)
        print(razmak,"---------")

        privremenaVA=(int(privremenaVA,2)-int(vB,2))
        privremenaVA=str(bin(privremenaVA)).replace('0b','')

        for x in range (len(privremenaVA),len(vB)):
            razmak=str(" ")+razmak

        print(" ",razmak,privremenaVA)
        rezultat=rezultat+str('1')


    elif privremenaVA==vB:
        print(razmak,"---------")
        print(razmak," ",privremenaVA)
        print(razmak,"-",vB)
        print(razmak,"---------")

        for x in range (0,len(privremenaVA)):
            razmak=str(" ")+razmak

        privremenaVA=str('0')
        rezultat=rezultat+str('1')
        print(razmak,privremenaVA)

    elif privremenaVA<vB:
        if len(vA)>brojZnamenakaVa:
```

```
            print(" ",razmak,privremenaVA)
            rezultat=rezultat+str('0')
        else:
            print(" ",razmak,privremenaVA)
            rezultat=rezultat+str('0')
```

```
if int(privremenaVA)==0 and len(vA)<=int(brojZnamenakaVa):

    if decimalnaTocka<0:
        rezultat=str("0")+str(".")+rezultat[decimalnaTocka:]
    elif decimalnaTocka==0:
        rezultat=rezultat[:decimalnaTocka+1]+str(".")+str("0")
    elif decimalnaTocka>0:
        rezultat=rezultat[:decimalnaTocka+1]+str(".")+rezultat[decimalnaTocka+1:]+
    str("0")
    print("\n\nRezultat djeljenja je: ",rezultat)
    break

if len(vA)>brojZnamenakaVa:
    privremenaVA=privremenaVA+vA[brojZnamenakaVa]
else:
   privremenaVA=privremenaVA+str('0')
brojZnamenakaVa+=1
```

Za primjer unesimo $61.875/8.25$ u program:



Slika 7: Aritmetička operacija dijeljenja

5. Greške

Korištenjem aritmetike s pomičnim zarezom, računalo sprema sve obrađene informacije ili informacije iz svijeta u memoriju. Svaki tip podataka koristi nekakav oblik pomičnog zareza za pohranu podataka te iz tog razloga svaki operacijski sustav treba biti spreman da reagira na iznimke pomičnog zareza. Sabiranje beskonačno mnogo realnih brojeva u ograničeni broj bitova je nemoguć te je potrebno rješenje kako bi to bilo moguće. Beskonačni zapisi realnog broja se ne mogu spremiti u konačan binarni zapis te se približnim prikazom broj treba zaokružiti kako bi se prikazao konačan zapis,to jest, svaki rezultat svake operacije se mora moći biti prikazati. Približnim prikazanom često dolazi grešaka prilikom zaokruživanja.

Za prikaz realnih brojeva najviše se koristi prikaz s pomičnim zarezom. Parametri koji definiraju prikaz s pomičnim zarezom su baza β (u računalu se koristi baza 2) i preciznost p. U svrhu obrade teme završnog rada se upotrebljava IEEE754 standard za izračunavanje s pomičnog zareza, to jest, standard jednostruke preciznosti što znači da je baza β=2 i preciznost p=24. IEEE–754 standard propisuje da sve četiri osnovne aritmetičke operacije vrijedi ista ocjena greške zaokruživanja i da izračunati rezultat ima malu relativnu grešku. Postoje dvije vrste realnih brojeva koje se ne mogu prikazati kao broj s pomičnim zarezom iz dva razloga, prvi je da broj u binarnom obliku ima beskonačan niz ponavljajućih znamenki (npr. $0.1_{(10)} = 0.100110011001...$), a drugi razlog je da je broj van raspona koji se može prikazati.

Kod višestrukih operacija u kojima nastaje mali problem događa se širenje greške te greške postaju veće greške te je potrebno procijeniti grešku u rezultatu. U računalnoj aritmetici ne vrijedi asocijativnost zbrajanja i množenja ni distributivnost množenja prema zbrajanju. Vrijedi samo komutativnost zbrajanja i množenja.

5.1. Greške zaokruživanja

Greška zaokruživanja je razlika između realne predodžbe broja i izgleda broja nakon zaokruživanja. Drugim riječima, broj π danas ima široku primjenu te se taj broj stalno koristi. Iako svima broj π je jednak broju 3.14 iz razloga što je broj π lakše zaokružiti i zapisati kao broj sa dvije decimale te s tom vrijednosti je lakše računati, ali zapravo broj π je beskonačan niz brojeva. Kada bi osoba uzela papir i olovku te krenula zapisivati broj π, započela bi niz sa 3,14159 26535 89793... sve dok osoba ne bi ostala bez mjesta na papiru. Iz tog razloga se broj π zapisuje kao 3.14 te se uzima dodatno decimalnih mjesta po želji.

Tako i računalo neke brojeve kada pretvori iz dekadskog brojevnog sustava u binarni brojevni sustava bi dobio beskonačan niz te bi taj broj zapisivao sve dok ne bi ostao bez memorije. Za primjer uzmimo 0.1_{10} u binarnom sustavu taj broj je 0.00011 0011 0011 0011 0011 0011... beskonačan niz brojeva koji teži u 0.1, ali nikad neće biti 0.1.

Svi realni brojevi se mogu zapisati u računalo, ali brojevi koji su jako veliki ili su beskonačan niz brojeva trebaju se zaokružiti. Broj 1.1_{10} kada bi se zapisivao u IEEE754 standardu broj bi izgledao 0 0111 1111 0001 1001 1001 1001 1001 100 1 1001... Može se vidjeti kako je broj nemoguće zapisati unutar 23 bita mantise. U tom slučaju je potrebno naći dva najbliža

prikaziva susjeda, to jest, po 24 bitu tog niza odrediti za jedan veći ili jedan manji. Na zadanom primjeru bi to izgledalo sljedeće:

$$\text{zadani niz} \;=\; B \;=\; 1.0001\ 1001\ 1001\ 1001\ 1001\ 100\ \underline{1}$$

$$\text{za jedan manji} \;=\; B_{-} \;=\; 1.0001\ 1001\ 1001\ 1001\ 1001\ 011$$

$$\text{za jedan veći} \;=\; B_{+} \;=\; 1.0001\ 1001\ 1001\ 1001\ 1001\ 101$$

Nakon toga se rezultat zaokružuje, a to se vrši na način da se uzme onaj koji je najbliži zadanom broju što je ujedno i standardan način zaokruživanja, ali ako su oba jednako udaljena onda se izabire parni broj od dobivena dva, to jest, gleda se onaj koji ima zadnji bit 0. U primjeru su prikazana dva načina zaokruživanja: prema dolje $(-\infty)$, prema gore (∞) i ostao je treći način a to je prema nuli (odbacivanje viška znamenaka)

5.2. Relativna greška i Ulps

Recimo da dekadski broj 0.012344 želimo predstaviti u sustavu s pomičnim zarezom sa preciznošću $p = 3$, tada se dobije $1.23 \cdot 10^{-2}$ dok je greška posljednjeg mjesta .44. Ljudima se to ne čini kao velika greška te će zaokruženu vrijednost uzeti za dalju obradu gdje takva greška neće predstavljati nikakav problem, ali za izračune u kojima je preciznost izračuna ključna, greška 0.000044 može imati čak i kobne posljedice. Iz tog razloga je bitno imati način mjerenja pogreški zaokruživanja.

Zašto broj $1/3$ koji je u decimalnom obliku 0.33333... te zbroj $1/3 + 1/3 + 1/3$ ljudima je jednak broju jedan dok računalo taj zbroj ne ne vidi kao broj 1 već kao broj 0.999999... Gledati rezultat zbroja $1/3+1/3+1/3$ kao 1, a ne kao 0.999999 i nije nekakva greška, dapače, ni nije greška. Pošto se broj 9 ponavlja u beskonačnost, samo za premjer uzmimo da se rješenje zbroja konačan broj 0.999. Ako sad usporedimo grešku zaokruživanja broja 0.999 sa brojem 1, može se vidjeti da je greška 0.001. S tim saznanjem vratimo se na broj 0.9999... sa znamenkom broja 9 koji se ponavlja u beskonačnost, greška od 0.001 bi se smanjivala u beskonačnost te bi vrijednost te greške toliko smanjila da bi bila totalno zanemariva. Broj s pomičnim zarezom ako se koristi za predstavljanje nekog jako velikog ili beskonačnog broja koji se ne može prikazati unutar IEEE754 standarda potrebno je zaokružiti broj, a zatim i izračunati vrijednost pogreške.

Za početak će se greške zaokruživanja primijeniti na dekadskim brojevnom sustavu kako bi se lakše pojasnila tema, Za izračun greške se koristi izraz ulp (engl. units in the last place). Ako je rezultat izračuna broj s pomičnim zarezom jako blizu točnog rezultata, i dalje bi moglo doći do pogreške za čak 0,5 ulp. Drugi način za računanje razlike između broja s pomičnim zarezom i realnog broja kojem se približava je razlika između dva broja podijeljena s realnim brojem. Recimo za dekadski broj 1.2343 prikazan s preciznošću $p = 3$ dobije se $1.23 \cdot 10^0$. Razlika tih brojeva je $1.2343 - 1.23 = 0.0043$. Zatim kada se podjeli razlika i broj prije zaokruživanja dobije se da je greška $0.0043/1.2343 \approx 0.003$. Tako mali brojevi se mogu izbjeći zbog toga što se relativna pogreška obično zapisuje kao faktor puta ε koji se računa sljedećom

formulom:

$$\varepsilon = (\beta/2)(\beta)^{-p}$$

Za prijašnji primjer vrijedi

$$\beta = 10$$

$$p = 3$$

$$\varepsilon = (10/2)(10)^{-3} = 5(10)^{-3} = 0.005$$

i tada se relativna greška računa:

$$((0.0043 \ / \ 1.2343) \ / \ 0.005)\varepsilon \approx 0.7\varepsilon$$

Kako bi se razumila razlika između ulp-a i relativne pogreške, npr., broj x = 12.35 kada pretvorimo u broj s pomičnim zarezom i preciznost p=3 te zaokružimo broj dobije se $1,24 \cdot 10^1$. Može se vidjeti kako je greška 0.005 ulp, relativna greška je:

$$((0.005/1.235)/0.005) \approx 0.8\varepsilon$$

Kada bi se povećao broj x za 8 puta, to jest $12.35 \cdot 8$, za rezultat se dobije 98.8, a kada bi se isti broj s pomičnim zarezom i preciznost p=3 pomnožio s 8, to jest $1.24 \cdot 8$, rezultat je $9.92 \cdot 10^1$. Greška je 4.0 ulp, ali je relativna greška i dalje 0.8:

$$((0.04/9.92)/0.005) \approx 0.8\varepsilon$$

te se može vidjeti iako je relativna greška, greška izmjerena u ulp je 8 puta veća.

Nakon pojašnjenja na dekadskom brojevnom sustavu, vrijeme je da se pojasne greške zaokruživanja u IEEE754 standardu. Vratimo se na primjer 1.1_{10} koji u IEEE754 standardu izgleda 0 0111 1111 0001 1001 1001 1001 1001 100 1 1001.... Pošto znamenke decimalnog djela idu u beskonačnost potrebno je zaokružiti broj.

$$\text{za jedan manji} = B_- = 1.0001\ 1001\ 1001\ 1001\ 1001\ 011$$

$$\text{za jedan veći} = B_+ = 1.0001\ 1001\ 1001\ 1001\ 1001\ 101$$

nakon zaokruživanja prema dolje $(-\infty)$ i prema gore (∞) potrebno je odrediti koji je zaokruženi broj bliži unesenom broju 1.1:

$$B_- = 1.0001\ 1001\ 1001\ 1001\ 1001\ 011 =$$

$$2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-22} + 2^{-23} =$$

$$1.09999978542$$

$$B_+ = 1.0001\ 1001\ 1001\ 1001\ 1001\ 101$$

$$2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-21} + 2^{-23} =$$

$$1.10000002384$$

$$1.1 - B_- = 0.00000021458$$

$$B_+ - 1.1 = 0.00000002384$$

Može se vidjeti kako je B_+ bliži broju 1.1_{10} te se umjesto broja 1.1 u memoriju sprema broj B_+. Time se dogodila greška zaokruživanja "zadnjeg bita" mantise te se taj broj zove jedinična greška zaokruživanja. Može se vidjeti kako je greška zaokruživanja 0.00000002384, to jest, greška zaokruživanja je $2.384 \cdot 10^{-8}$ ili u binarnom brojevnom sustavu kako to računalo vidi znači da se greška nalazi u 2^{-22} bitu mantise što predstavlja vrlo malu relativnu grešku

5.3. Programska realizacija greške zaokruživanja u Pythonu

Odabirom četvrte opcije na glavnom izborniku, program ulazi u funkciju greske(). Funkcija greske() daje korisniku opciju da upiše realni broj. Nakon što se upiše željeni broj, program računa predznak, eksponent i mantisu kao i kod IEEE754 funkcije samo što u ovom slučaju je dodan dodatni prostor za provjeru dali se kod unesenog broja događa greška zaokruživanja. Ukoliko se u tom zapisu ne nalazi ništa, to znači da se kod unesenog broja ne javlja greška te, nakon ispisa IEEE754 zapisa unesenog broja, program izlazi iz funkcije na glavni izborni. Ukoliko program vidi da uneseni broj ne stane u 23-bitni prostor mantise, program će prvo javiti da greška zaokruživanja postoji te će ispisati IEEE754 oblik broja plus dio niza koji se preljeva. Program pamti prvi bit preljeva. Zatim program na temelju ispisanoga broja zaokružuje broj za bit više i za bit manje te ispisuje te brojeve u binarnom obliku. Dobiveni brojevi se pretvaraju u dekadski i oduzimanju od unesenog kako bi se vidio koji od dobivenih brojeva daje manju grešku te se nakon toga ispisuje broj koji se zapravo zapisuje u memoriju i kolika je greška zaokruživanja. Navedeni dio koda izgleda sljedeće:

```
print("   - Kod unesenog broja se javlja greska te je uneseni broj je niz brojeva
    koji ne stane unutar 24 bita mantise")
print("   - Uneseni broj bi u IEEE754 standardu izgledao:  ", str(predznak),
    eksponentB2, mantisa2[:23],"[",mantisa2[23:31],"...( i tako dalje )]")
print("   - Bitno je zapisati mantisu te sljedecu znamenku nakon 23-bitne mantise (
    znaci 24 bit tog niza) iz razloga sto se tom 24 bitnom broju dodaje i oduzima
    broj 1")
print("\n     -originalni dio mantise te sljedeca znamenka:        ", mantisa2[:23],
     mantisa2[24])

b=str('00000000000000000000001')
jedanVise=zbrajanje2(mantisa2[:24],b)
print("     -kada se mantisa poveca za jedan, dobije se:         ",jedanVise)
c=str('11111111111111111111111')
```

```
jedanManje=zbrajanje2(mantisa2[:24],c)
print("     -kada se mantisa smanji za jedan, dobije se:         ",jedanManje)

print("\n   -zatim se gleda kada mantisu povecamo za jedan!");
veci=manjiVeci(predznak,eksponentB2,jedanVise)

print("\n   -zatim se gleda kada mantisu smanjimo  za jedan!");
manji=manjiVeci(predznak,eksponentB2,jedanManje)

V=veci-n
M=n-manji

print("\n   -Razlika unesenog broja i broja koji je dobiven povecanjem mantise za 1:
     ",V)
print("   -Razlika unesenog broja i broja koji je dobiven smanjenjem mantise za 1: "
    ,M)
if V<M:
    print("\n   -Moze se vidjeti kako je  ",veci," blizi unesenom broju te je greska
     zaokruzivanja ", V)
elif V>M:
    print("\n   -Moze se vidjeti kako je  ",manji," blizi unesenom broju te je
    greska zaokruzivanja ", M)
else:
    if jedanVise[23]==0:
        print("\n  -Posto su oba broja jednako udaljeni od unesenog, uzima se broj",
     veci, "iz razloga sto ima parnu zadnju znamenku mantise"  )
    elif jedanManje[23]==0:
        print("\n  -Posto su oba broja jednako udaljeni od unesenog, uzima se broj",
     manji, "iz razloga sto ima parnu zadnju znamenku mantise"  )
    else:
        print("\n  -Posto su oba broja jednako udaljeni od unesenog i oba broja
    imaju parnu zadnju znamenku"  )
```

u prethodno ispisanom dijelu programa se mogu primijetiti dvije funkcije: zbrajanje2() i manjiVeci(). Funkciju zbrajanje2() nije potrebno dodatno pojašnjavati iz razloga što se u toj funkciji samo broj zaokružuje na jedan više ili jedan manje, odnosno oduzima se od glavne mantise i prvog bita preljeva broj jedan ili se broj jedan zbraja sa glavnom mantisom i brojem 1. Dok s druge strane, potrebno je detaljnije objasniti funkciju manjiVeci(). U funkciji manjiVeci() se prvo IEEE754 zapis broja pretvara u decimalni binarni broj te se rezultat ispisuje na ekran:

```
eksponent=int(eksponent, 2)
eksponent=eksponent-127
mantisa=mantisa[:23]

if eksponent==0:
    broj=str('1.')+mantisa

elif eksponent>0:
    broj=str('1')+mantisa[:eksponent]+str('.')+mantisa[eksponent:]

elif eksponent<0:
    dodatak=str('1')
```

```
    for y in range (1, -(eksponent)):
        dodatak=str('0')+dodatak
    broj=str('0.')+dodatak+mantisa
```

Zatim program taj broj pretvara u dekadski zapis i vraća taj zapis nazad u funkciju greske() da daljnju obradu.

```
if eksponent<0:

        cijeli, decimalni = str(broj).split(".")
        cijeli=int(cijeli, 2)
        for x in range(0, len(decimalni)):
            if decimalni[x]==str("1"):
                e=x\cdot(-1)-1
                rezultat=rezultat+pow(2,e)

    else:
        broj=str('1')+mantisa[:eksponent]+str('.')+mantisa[eksponent:]
        cijeli, decimalni = str(broj).split(".")
        cijeli=int(cijeli, 2)

        for x in range(0, len(decimalni)):
            if decimalni[x]==str("1"):
                e=x\cdot(-1)-1
                rezultat=rezultat+pow(2,e)
    finalni=cijeli+rezultat
    if predznak==0:
        print("     -Decimalni zapis broja u dekadskom brojevnom sustavu:" ,cijeli+
    rezultat);
    elif predznak==1:
        print("     -Decimalni zapis broja u dekadskom brojevnom sustavu: -" ,cijeli
    +rezultat);
    return finalni
```

Za primjer ću uzeti broj 1.1 kako bi prikazao rad programa kakav bi korisniku bio prikazan

```
1. PRETVARANJE REALNOG BROJA U BINARNI
2. PRETVARANJE REALNOG BROJA U ZAPIS KOJI RAČUNALO KORISTI ZA SPREMANJE REALNOG BROJA (IEEE754 ZAPIS)
3. PRIKAZ KAKO RAČUNALO OBRAĐUJE ARITMETIČKE OPERACIJE S REALNIM BROJEVIMA
4. PRIKAZ KAKO RAČUNALO OBRAĐUJE GREŠKE
9. PREKINI PROGRAM


ODABIR: 4
Unesite realan broj: 1.1
   - Kod unesenog broja se javlja greška te je uneseni broj je niz brojeva koji ne stane unutar 24 bita mantise
   - Uneseni broj bi u IEEE754 standardu izgledao:   0 01111111 00011001100110011001100 [ 11001100 ...( i tako dalje )]
   - Bitno je zapisati mantisu te sljedeću znamenku nakon 23-bitne mantise(znači 24 bit tog niza) iz razloga što se tom 24 bitnom broju dodaje i oduzima broj 1

     -originalni dio mantise te sljedeća znamenka:        00011001100110011001100 1
     -kada se mantisa poveča za jedan, dobije se:         00011001100110011011010
     -kada se mantisa smanji za jedan, dobije se:         00011001100110011011000

   -zatim se gleda kada mantisu povečamo za jedan!
     -Decimalni zapis broja u binarnom brojevnom sustavu: 1.00011001100110011001101
     -Decimalni zapis broja u dekadskom brojevnom sustavu: 1.100000023841858

   -zatim se gleda kada mantisu smanjimo  za jedan!
     -Decimalni zapis broja u binarnom brojevnom sustavu: 1.00011001100110011001100
     -Decimalni zapis broja u dekadskom brojevnom sustavu: 1.0999999046325684

   -Razlika unesenog broja i broja koji je dobiven povečanjem mantise za 1:  2.3841857821338408e-08
   -Razlika unesenog broja i broja koji je dobiven smanjenjem mantise za 1:  9.536743172944284e-08

   -Može se vidjeti kako je   1.100000023841858  bliži unesenom broju te je greška zaokruživanja  2.3841857821338408e-08
```

Slika 8: Greške zaokruživanja

5.4. Greške zaokruživanja u stvarnom životu

Postoje mnogi zapisi u stvarnom životu kako je greška zaokruživanja koja je jakom mala može rezultirati od proglašenja pogrešnog predsjednika zbog grešaka zaokruživanja postotka postotka glasova po saveznim državama pa sve do potonuća naftne platforme. Jedan od takvih primjera je i Patriot. Patriot je bio vojni sustav protuzračne obrane koji zbog pogreške nisu uspjele oboriti Iračku raketu u Saudiskoj Arabiji. Rezultat te greške je taj da iračka raketa Scud Pukim slučajem pala na američku vojnu bazu usmrtivši 28 vojnika dok je nešto manje od stotinu bilo ranjeno. Raketni obrambeni sustav Patriot tijekom operacije Desert Storm u Dhahranu u Saudijskoj Arabiji 25.2.1991. nije popratio kretanje nadolazećeg Scud-a kako bi ga presreo.

Raketa Patriot je napravljena krajem 70-ih godina kao protuzračno oružje te se 80-ih premodificirao sa svrhom obrane nadolazećih kratko dometnih projektila. Vjeruje se kako se problem pojavio iz dizajna pošto je patriot bio zamišljen kao protuzračni, a ne protu raketni sustav te su inžinjerki koji su radili na tom sustavu napravili određena ograničenja. Jedno od ograničenja je bio taj što se vjerovalo da sustav Patriot neće raditi duže od nekoliko sati te da će raditi u mobilnoj, to jest, pokretnoj jedinici, a ne na fiksnom mjestu. Sustav koji je upravljao Patriot raketama je vrijeme brojilo u desetinama sekunde koje proteknu od uključivanja sustava. Desetina sekunde je 0.1, a kako se moglo vidjeti 0.1 u binarnom brojevnom sustavu je beskonačan niz 0.00011 gdje se 0011 ponavlja u beskonačnost. Računalo na kojem se sustav pokretao je koristio mantisu duljine 23 bita što rezultira greškom od $9.5 \cdot 10^{-8}$. Naravno ne čini se kao velika greška, ali nakon 100 sati rada ta greška je dosegla 0.34 sekunde. Scud raketa putuje od prilike 1.6km/s što je rezultiralo time što je Patriot sustav tražio Scud-a oko pola kilometra od njegove stvarne pozicije. Sustav je to klasificirao kao lažni alarm pošto nije nalazio prijetnju te ga je zanemario što je rezultiralo katastrofom.

Dva tjedna prije katastrofe je uočena greška te je ured Patriota bio obaviješten. Nakon nekoliko dana je problem bio riješen te se softver sustava bio ažuriran i spreman za instalaciju na Patriot sustave. Za vrijeme napada na Dhahran, ažuriranje softvera još nije stiglo u Dhahran, stiglo je dan kasnije.

6. Zaključak

Za zaključak se može reći kao je binarni brojevni sustav izgledom dosta drugačiji od dekadskog, ali pretvaranje brojeva iz jednog u drugi sustav nije teško. Isto tako, ni binarna aritmetika nije teška ako se savladaju osnovna znanja o binarnom sustavu, što više, dosta učenika i studenata koji u svom programu školovanja ili studiranja imaju predmet ili kolegiji posvećena binarnom sustavu i binarnoj aritmetici takve teme su im čak i zanimljive i lagane za shvatiti, naravno dok brojevi nisu jako veliki ili jako mali. Savladavanjem binarnog brojevnog sustava se za korak približavamo razumijevanju kako računalo obrađuje podatke. Kroz završni rad se opisuje kako računalo vidi podatke vanjskog svijeta, kako ih obrađuje te kako ih zapisuje u memoriju.

Svi znamo kako računalo može pohraniti jako velike i jako male brojeve unutar ograničenog memorijskog prostora. Za to je zaslužan standard pomičnog zareza. U radu se radilo sa IEEE754 standardom koji broj zapisuje unutar 32-bitnog prostora. Iako postoje standardi sa većim memorijskim prostorom, u svrhu završnog rada se koristi 32-bitni standard radi jednostavnosti pojašnjenja.

Ponekad za jako veliki ili jako mali brojevi zbog duljine mantise niti taj prostor nije dovoljan, na red dolazi do greške prilikom zaokruživanja. Greška zaokruživanja sprema broj najbliži unesenom broju pri tome se zna i koliko je odstupanje od prvobitnog broja. Iako se takve sitne greške normalnom čovjeku čine smiješne da uopće obraćao pažnju na njih, inženjerima i programskim stručnjacima te male greške nisu za zanemariti. Kao što je primjer dokazao, takva mala odstupanja znaju u određenim situacijama dovesti do velikih promjena. Napisani završni rad prikazuje osnove binarnog sustava, prikaza realnih brojeva u računalu, binarne aritmetike i greške zaokruživanja te kroz primjere i programskim rješenjem se pobliže pojašnjavaju navedene teme. Savladavanjem i potpunim razumijevanjem aritmetike pomičnog zareza izgrađuju se stabilni temelji za detaljno razumijevanje i nadograđivanje stečenoga znanja.

Popis literature

[1] T. Strek, *Reliable computing: numerical and rounding errors*, [pristupano: 21.08.2021.]

[2] A. Klisuric, „POVIJESNI POČETCI NASTAJANJA I RAZVOJA BROJEVA," [pristupano: 20.08.2021.]

[3] L.-K. Wang i M. J. Schulte, „Decimal floating-point division using Newton-Raphson iteration," *Proceedings. 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004.*, IEEE, 2004., str. 84–95, [pristupano: 15.07.2021.]

[4] A. D. Booth, „A signed binary multiplication technique," *Computer Arithmetic: Volume I*, str. 123, 2015., [pristupano: 03.08.2021.]

[5] B. Brown, „Binary arithmetic," *Computer science*, str. 1–9, 1999., [pristupano: 10.07.2021.]

[6] P. Markstein, „The New IEEE-754 Standard for Floating Point Arithmetic," *Numerical Validation in Current Hardware Architectures*, A. Cuyt, W. Krämer, W. Luther i P. Markstein, ur., serija Dagstuhl Seminar Proceedings, Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008. adresa: `http://drops.dagstuhl.de/opus/volltexte/2008/1448`, [pristupano: 15.07.2021.]

[7] G. Sng, C. Hien i D. L. Ho-hon, *Construction Of The Real Number System*, [pristupano: 07.07.2021.]

[8] M. Brain, C. Tinelli i T. Wahl, *An Automatable Formal Semantics for IEEE-754 Floating-Point Arithmetic*, [pristupano: 19.07.2021.]

[9] J. M. Lodder, *Binary Arithmetic: From Leibniz to von Neumann*, [pristupano: 15.07.2021.]

[10] P. A. McQuaid, „Software disasters—understanding the past, to improve the future," *Journal of Software: Evolution and Process*, sv. 24, br. 5, str. 461–463, 2012., [pristupano: 31.08.2021.]

[11] W3Schools, *Python Tutorial - W3Schools*. adresa: `https://www.w3schools.com/python/`.

[12] M. Ninković i V. Ovčina. (). „Matematika osnovna razina," adresa: `https://www.algebra.hr/drzavna-matura/wp-content/uploads/sites/4/2018/08/Matematika-B-ni%C5%BEa.pdf`. [pristupano: 07.07.2021.]

[13] „Binarni brojevni sustav," *Hrvatska enciklopedija, mrežno izdanje.*, Leksikografski zavod Miroslav Krleža, 2021. adresa: `https://www.enciklopedija.hr/natuknica.aspx?ID=7696`, [pristupano: 07.07.2021.]

[14] M. K. Bakula, „Numericka analiza," 2008. adresa: https://mapmf.pmfst.unist.hr/~milica/Numericka_analiza/Folije_za_predavanja/NA_Aritmetika.pdf, [pristupano: 19.07.2021.]

[15] *The Foating-Point Guide*. adresa: https://floating-point-gui.de/, [pristupano: 20.08.2021.]

[16] N. C. Guide, „Appendix D What Every Computer Scientist Should Know About Floating-Point Arithmetic,"

Popis slika