Programiranje 1

Predavanje 04 - Prikaz realnih brojeva u računalu, realna aritmetika

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

19. listopada 2023.

Uvod u prikaz realnih brojeva

Osnovno pitanje: kako pohraniti jako velike ili jako male realne brojeve?

Ideja, koristimo **znanstveni zapis** za prikaz:
$54545000000 = 5.4545 \cdot 10^{10}$, $0.000001234 = 1.234 \cdot 10^{-6}$.

U računalu: broj $=$ predznak $\cdot$ mantisa $\cdot\ 2^{\text{eksponent}}$.
Mantisa se obično pohranjuje u **normaliziranom** obliku
$1 \leq \text{mantisa} < (10)_2$.

Pohrana mantise i eksponenta u **konačno mnogo bitova**.

- Prikaziv **samo raspon** realnih brojeva.
- Neki brojevi unutar prikazivog raspona **nisu prikazivi jer im je mantisa predugačka** (dolazi do zaokruživanja).

Uvod u prikaz realnih brojeva

$1110.11 = 1.11011 \cdot 2^3$
$0.0001011001 = 1.011001 \cdot 2^{-4}$

Vodeći bit se **ne mora pamtiti** ukoliko znamo da je broj$\neq 0$. Možemo iskoristiti za prikaz **dodatne znamenke mantise**. U tom slučaju vodeći bit zovemo **skriveni bit**.

Ovo je pojednostavljeni prikaz, stvarni prikaz u računalu je **još malo složeniji**.

Stvarni prikaz realnih brojeva

Eksponent se prikazuje u **zamaskiranoj (pomaknutoj) formi**. Stvarnom eksponentu e se **dodaje konstanta** takva da je pomaknuti eksponent **uvijek pozitivan za normalizirane brojeve**.

Konstanta ovisi o broju bitova za prikaz eksponenta i bira se tako da je prikaziva **recipročna vrijednost najmanjeg pozitivnog normaliziranog broja**.

Pomaknuti eksponent se zove **karakteristika** a normalizirana mantisa se zove **signifikand**.

Stvarni prikaz se sastoji od tri glavna dijela:

- **Predznaka** s - zauzima jedan bit (**najviši**).
- **Karakteristike** k - zauzima sljedećih w bitova (w - širina pomaknutog eksponenta).
- **Signifikand** m - zauzima sljedećih t bitova (t - završni ili razlomljeni dio od m).

Stvarni prikaz realnih brojeva

Preciznost: $p = t + 1$, **ukupni broj vodećih značajnih** bitova cijele mantise.
Karakteristika: $k \in \{0, \ldots, 2^w - 1\}$, cijeli broj bez predznaka.
Rubne vrijednosti od k označavaju posebna stanja:

- $k = 0$, nula i denormalizirani brojevi.
- $k = 2^w - 1$, beskonačno i nije broj (`NaN`).

Ostale vrijednosti karakteristike se koriste za prikaz **normaliziranih brojeva različitih od nula**.
$k = e + \text{bias}$ (k - karakteristika, e - eksponent, bias$= 2^{w-1} - 1$).
$e \in \{-(2^{w-1} - 2), \ldots, (2^{w-1} - 1)\}$.

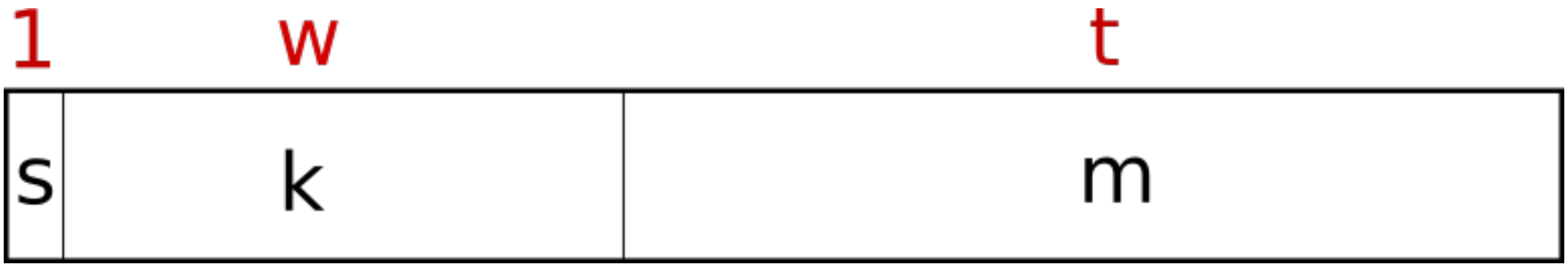
Standardni tipovi (*IEEE* 754 – 2008)

ime tipa	binary32	binary64	binary128
duljina u bitovima	32	64	128
$t =$	23	52	112
$w =$	8	11	15
$u = 2^{-p}$	2^{-24}	2^{-53}	2^{-113}
$u \approx$	$5.96 \cdot 10^{-8}$	$1.11 \cdot 10^{-16}$	$9.63 \cdot 10^{-35}$
raspon brojeva $\approx$	$10^{\pm 38}$	$10^{\pm 308}$	$10^{\pm 4932}$

u - jedinična greška zaokruživanja.

Prikaz realnih brojeva

- s - predznak. Vrijednost: 0 za pozitivan broj, 1 za negativan broj. Duljina: 1 bit.
- k - karakteristika. Duljina: w bitova.
- m - mantisa (signifikand). Duljina: t bitova.

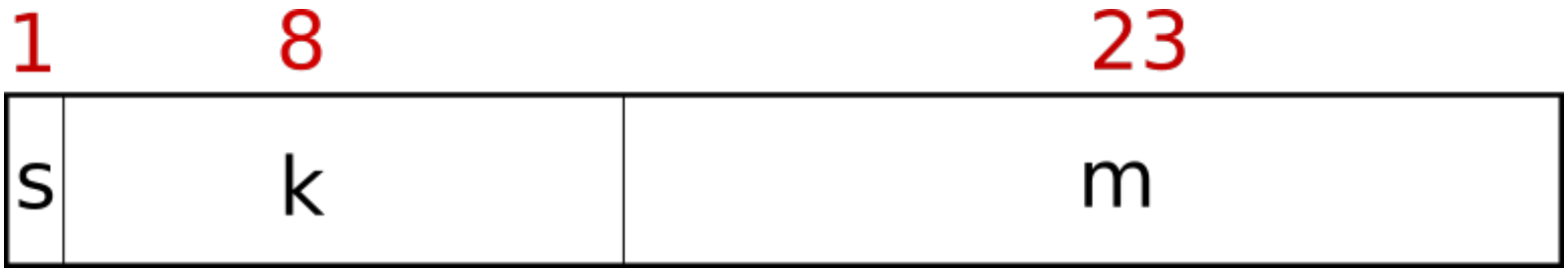


Stvarni prikaz tipa binary32

binary32 je najkraći realni tip (realni broj **jednostruke** točnosti). U programskom jeziku *C* odgovara tipu `float`.

Svojstva:

- duljina 4 byte-a (32 bita).



- U mantisi se ne pamti vodeća jedinica **ukoliko je broj normaliziran**.
- $e \in \{-126, \ldots, 127\}$.
- $k = e + 127, \;\; k \in \{1, \ldots, 254\}$.
- $k = 0$ i $k = 255$ se koriste za posebna stanja.

Prikaz binary32 - primjer

Želimo prikazati broj $(10.125)_{10}$ kao broj u jednostrukoj točnosti. $(10.125)_{10} = (10 + \frac{1}{8})_{10} = (10 + 2^{-3})_{10} = (1010.001)_2 = 1.010001 \cdot 2^3$.

Vrijedi:

- $s = 0$
- $k = e + 127 = (130)_{10} = (2^7 + 2^1)_{10} = 1000\ 0010$
- $m = 0100\ 0100\ 0000\ 0000\ 0000\ 000$

Prikaz broja nula kod binary32

Realni broj **nula** ima dva prikaza:

- mantisa i karakteristika **imaju sve bitove jednake** 0 ($k = 0,\ m = 0$).
- Predznak može biti **pozitivan** ili **negativan**. To daje **pozitivnu** ili **negativnu nulu**.

Vrijednost pozitivne i negativne nule je jednaka (kada se uspoređuje).

Prikaz nule:

- $+0 = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000.$
- $-0 = 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000.$

Denormalizirani brojevi kod binary32

Ukoliko imamo $k = 0$ i $m \neq 0$, kao eksponent se uzima -126 (najmanji dozvoljeni). Takav broj se zove **denormalizirani broj**. Mantisa takvog broja nije normalizirana i počinje s $0.m$.

Promotrimo prikaz binarno zapisanog realnog broja:
$0.000\ 0000\ 0000\ 0000\ 0000\ 1011 \cdot 2^{-126}$

- $s = 0$
- $k = 0000\ 0000$
- $m = 000\ 0000\ 0000\ 0000\ 0000\ 1011$

Prikaz brojeva $\pm\infty$ kod binary32

U slučaju $k = 255$ i $m = 0$ imamo:

- Ako $s = 0$, tada je to prikaz broja $+\infty$ (skraćeno `+Inf`).
- Ako $s = 1$, tada je to prikaz broja $-\infty$ (skraćeno `-Inf`).

Rezultat `Inf` (odnosno `-Inf`) dobivamo ako:

- pokušamo spremiti preveliki broj (*overflow*).
- ako nešto različito od nule podijelimo s **nulom**.

Prikaz broja $+\infty$ $(-\infty)$ je:

- $s = 0$ $(s = 1)$
- $k = 1111\ 1111$
- $m = 000\ 0000\ 0000\ 0000\ 0000\ 0000$

Prikaz vrijednosti `NaN` kod binary32

Ako je $k = 255$ i postoji bar jedan bit mantise **različit od nule**, onda je to oznaka za *nije broj* `NaN` (eng. **N**ot **a** **N**umber).

Rezultat `NaN` označava da je došlo do primjene nedefinirane aritmetičke operacije tijekom izvrednjavanja aritmetičkog izraza. Primjeri su: **dijeljenje nule s nulom**, **računanje drugog korijena negativnog broja** itd.

- $s = 0$
- $k = 1111\ 1111$
- 000 0000 0000 0101 0000 0000

Greške zaokruživanja

Postoje realni brojevi koje ne možemo egzaktno spremiti u računalo čak i kada su unutar prikazivog raspona brojeva **zato što imaju predugačku mantisu**.

$B = 1.0001\ 0000\ 1000\ 0011\ 1001\ 0111$ ima 25 znamenki mantise i **ne može** se egzaktno spremiti u realni broj **jednostruke točnosti** (tip `float` u C-u).

U takvom slučaju se pronalaze dva najbliža susjeda: $B_- < B < B_+$.
Za broj B, takvi susjedi bi bili:
$B_- = 1.0001\ 0000\ 1000\ 0011\ 1001\ 011$
$B_+ = 1.0001\ 0000\ 1000\ 0011\ 1001\ 100$

Greške kod zaokruživanja

Nakon tog koraka se primjenjuje jedna od mogućih tehnika zaokruživanja:

- prema **najbližem** broju (standardno za sve procesore). Ako su dva susjedna broja jednako udaljena od B uvijek bira parni od ta dva (zadnji bit je 0).
- prema **dolje** (prema $-\infty$).
- prema **gore** (prema $+\infty$).
- prema **nuli** (odbacivanjem viška znamenki).

Broj B iz primjera će biti zaokružen u B_+ jer taj broj ima zadnji bit jednak 0 (broj je paran).

Način zaokruživanja može se birati postavljanjem **procesorskih zastavica** ili **postavki prevodioca**.

Ocjena greške za standardno zaokruživanje je **dva puta manja** od ocjene greške ostalih načina zaokruživanja.

Jedinična greška zaokruživanja

Ako je $x \in \mathbb{R}$ unutar raspona brojeva prikazivih u računalu, umjesto x se sprema **zaokruženi prikazivi broj** $fl(x)$. Time dolazi do greške pri zaokruživanju od $\leq \frac{1}{2}$ zadnjeg bita mantise ($\leq \frac{1}{2} \cdot 2^{-t} = 2^{-t-1} = 2^{-p}$). Ta **gornja ocjena** se zove **jedinična greška zaokruživanja** (oznaka u).
Za `float`: $u = 2^{-24} \approx 5.96 \cdot 10^{-8}$.
Vrijedi: $fl(x) = (1 + \varepsilon) \cdot x$, $|\varepsilon| \leq u$, gdje je ε relativna greška napravljena zaokruživanjem. Kod `float`-a imamo **malu relativnu grešku**.

Preporuka: oprezno koristiti `float`. Koristiti `double` ukoliko je moguće (veća preciznost i raspon).

Raspon tipa `float`

$FLT_{MAX} = (1 - 2^{-24}) \cdot 2^{128} \approx 3.40282347 \cdot 10^{38}$.
Prikaz:

- $s = 0$
- $k = 1111\ 1110$
- $m = 111\ 1111\ 1111\ 1111\ 1111\ 1111$

$FLT_{MIN} = 2^{-126} \approx 1.17549435 \cdot 10^{-38}$.
Prikaz:

- $s = 0$
- $k = 0000\ 0001$
- $m = 000\ 0000\ 0000\ 0000\ 0000\ 0000$

Raspon tipa `float`

Simboličke konstante `FLT_MAX`, `FLT_MIN` definirane su u datoteci `float.h` i mogu se koristiti u *C* programima.

- $\frac{1}{FLT\_MIN}$ je egzaktno prikaziv.
- $\frac{1}{FLT\_MAX}$ nije egzaktno prikaziv (denormalizirani broj).

Najmanji prikazivi denormalizirani broj je
$2^{-126} \cdot 2^{-23} = 2^{-149} \approx 1.40129846 \cdot 10^{-45}$.

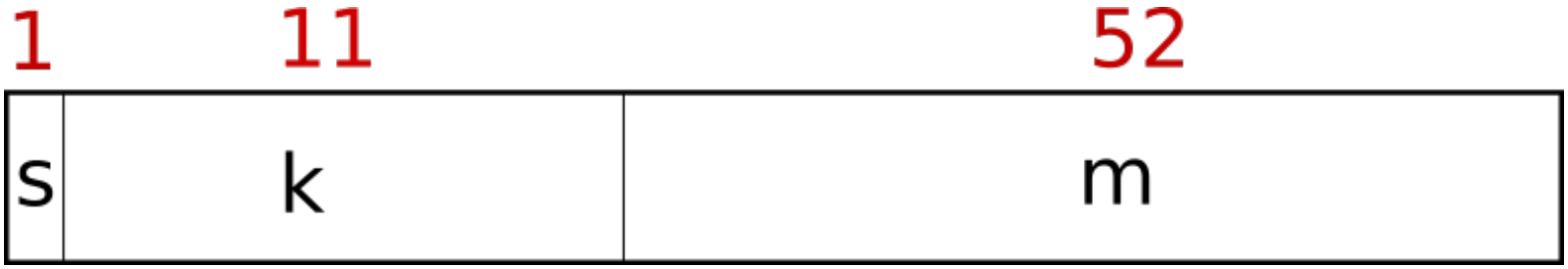
- $s = 0$
- $k = 0000\ 0000$
- $m = 000\ 0000\ 0000\ 0000\ 0000\ 0001$

Stvarni prikaz tipa binary64

binary64 je realni tip **dvostruke točnosti**. U *C*-u ga reprezentira tip `double`.

Svojstva:

- duljina 8 byte-a (64 bita).



- U mantisi se ne pamti vodeća jedinica **ukoliko je broj normaliziran**.
- $e \in \{-1022, \ldots, 1023\}$.
- $k = e + 1023, \;\; k \in \{1, \ldots, 2046\}$.
- $k = 0$ i $k = 2047$ se koriste za posebna stanja (na isti način kao i kod binary32).

Jedinična greška i raspon tipa `double`

Jedinična greška zaokruživanja za `double` je
$u = 2^{-53} \approx 1.11 \cdot 10^{-16}$.
Broj $1 + 2u$ je **najmanji prikazivi broj strogo veći od** 1.

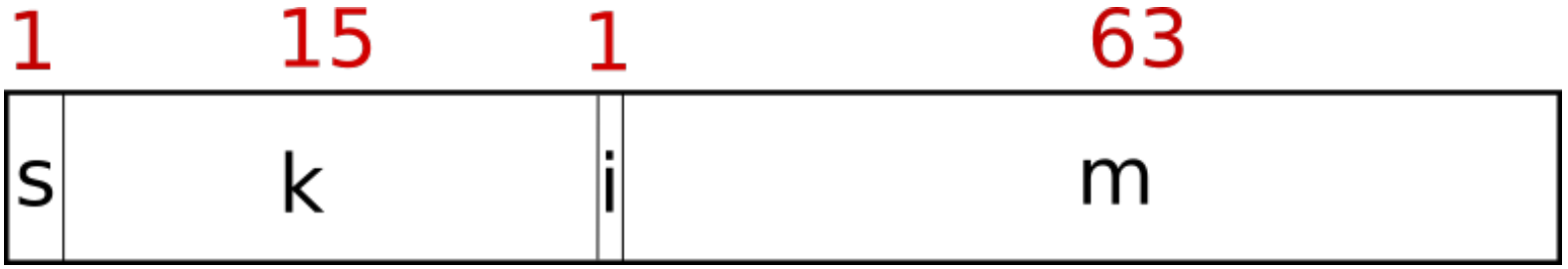
Najveći prikazivi pozitivni broj:
$DBL_{MAX} = (1 - 2^{-53}) \cdot 2^{1024} \approx 1.7976931348623157 \cdot 10^{308}$
Najmanji prikazivi normalizirani pozitivni broj:
$DBL_{MIN} = 2^{-1022} \approx 2.2250738585072014 \cdot 10^{-308}$

Tip `extended`

U *C*-u **možda** dohvatljiv kao `long double`.

Svojstva:

- duljina: 10 byte-a (80 bita).



- u mantisi se **pamti** vodeći bit i
- $e \in \{-16382, \ldots, 16383\}$
- $k = e + 16383, \;\; k \in \{1, \ldots, 32766\}$
- $k = 0$ i $k = 32767$ se koriste za posebna stanja.

Realna aritmetika računala - zbrajanje

Primjer: Pretpostavimo da imamo računalo koje radi u bazi 10 s $p = 4$ značajne dekadske znamenke. Treba naći rezultat **zbrajanja** brojeva: $x = 9.937 \cdot 10^0$, $y = 8.165 \cdot 10^{-2}$.

Koraci:

- Izjednačimo eksponent na onaj **veći**.
- Mantisu manjeg broja **pomaknemo udesno** ukoliko je potrebno.
- Mantise se **zbroje**.
- Dobiveni rezultat se **normalizira** i ako treba **zaokružuje**.

$x = 9.937 \cdot 10^0$
$y = 0.08165 \cdot 10^0$
$x + y = 10.01865 \cdot 10^0$
$x + y = 1.002 \cdot 10^1$
$fl(x + y) = 1.002 \cdot 10^1$

Realna aritmetika računala - standard

Realna aritmetika računala nije egzaktna!
Razlog je što rezultat svake operacije mora biti prikaziv zbog čega **dolazi do zaokruživanja**.

Standard `IEEE 754-2008` za realnu aritmetiku propisuje da sljedeća svojstva moraju vrijediti za sve četiri osnovne aritmetičke operacije:

- **ista ocjena greške zaokruživanja** kao i za prikaz broja.
- **izračunati rezultat** mora imati **malu relativnu grešku**.

Isto vrijedi za neke matematičke funkcije poput **drugog korijena** dok za funkcije kao `sin` (oko 0), `ln` (oko 1) ta svojstva ne vrijede.

Realna aritmetika računala - zaokruživanje

Neka je $\circ \in \{+, -, *, /\}$ i neka su x i y prikazivi operandi. Ako su x i y u dozvoljenom, **normaliziranom**, rasponu i ako se egzaktni rezultat operacije $x \circ y$ također nalazi u normaliziranom rasponu (nije nužno prikaziv) tada za prikaziv rezultat $fl(x \circ y)$ vrijedi: $fl(x \circ y) = (1 + \varepsilon)(x \circ y), \ |\varepsilon| \leq u$. u je jedinična greška zaokruživanja, a ocjena odgovara zaokruživanju egzaktnog rezultata.

Prava **relativna greška** ε ovisi o: x, y, operaciji $\circ$ i stvarnoj realizaciji aritmetike računala.

Napomena: za gornju ocjenu greške je **ključna pretpostavka o normaliziranom rasponu**, inače greška može biti i **puno veća**.

Realna aritmetika računala - zaokruživanje

Zbog zaokruživanja u realnoj aritmetici računala ne vrijede uobičajni zakoni aritmetičkih operacija iz $\mathbb{R}$.
a) **nema asocijativnosti zbrajanja i množenja**, b) **nema distributivnosti množenja prema zbrajanju**.

Poredak izvršavanja operacija je bitan!

Standardno pravilo koje vrijedi i u realnoj aritmetici računala je: **komutativnost za zbrajanje i množenje.**

Širenje grešaka zaokruživanja

Gotovo svaki rezultat u realnoj aritmetici računala ima nekakvu **grešku**.
Kod izračuna koji koriste puno aritmetičkih operacija očekujemo **akumulaciju** (tzv. **širenje**) grešaka[1].
Nužno je znati za učinkovito računanje **koliko brzo rastu** greške toga tipa.

Najopasnija operacija u realnoj aritmetici: **oduzimanje bliskih brojeva (kraćenje)**. Kod takvih operacija **iz potencijalno velikih (bliskih) brojeva** dobivamo **male** brojeve (može drastično povećati grešku).

[1] Više informacija o greškama zaokruživanja ćete naučiti na kolegiju Numerička matematika.

Primjer neasocijativnosti zbrajanja

Primjer: Asocijativnost zbrajanja u realnoj aritmetici računala **ne vrijedi**.

Izračunajmo konačnu sumu $S_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}$.
Prva stvar koju trebamo utvrditi je **kojim redosljedom računati**?
Zbrajanje je binarna operacija stoga moramo nekako grupirati parove pribrojnika da dobijemo konačnu sumu.

U primjeru imamo $n - 1$ primjenu binarne operacije zbrajanja. Kod realnih brojeva možemo računati u proizvoljnom redosljedu jer vrijedi asocijativnost zbrajanja.

Primjer neasocijativnosti zbrajanja

Promotrimo načine na koje možemo izračunati ovu sumu **bez primjene** svojstva komutativnosti zbrajanja.

- **Zbrajanje unaprijed:**
  $S_{n,1} = (\ldots((1 + \frac{1}{2}) + \frac{1}{3}) + \cdots + \frac{1}{n-1}) + \frac{1}{n}$,
- **Zbrajanje unatrag:**
  $S_{n,2} = 1 + (\frac{1}{2} + (\frac{1}{3} + \cdots + (\frac{1}{n-1} + \frac{1}{n})\ldots))$.

Promotrimo rezultat u realnoj aritmetici računala za $n = 1\ 000\ 000$ u tri standardne `IEEE` točnosti: **single**, **double** i **extended**.

Algoritmi za računanje $S_{n,1}$ i $S_{n,2}$ su:

- **unaprijed:** $S_{n,1} = 1,\ S_{n,1} = S_{n,1} + \frac{1}{i},\ i = 2, \ldots, n,$
- **unatrag:** $S_{n,2} = \frac{1}{n},\ S_{n,2} = \frac{1}{i} + S_{n,2},\ i = n-1, \ldots, 1.$

Primjer neasocijativnosti zbrajanja

Crveno je označena prva pogrešna znamenka.

tip i suma	vrijednost	relativna greška
`single` $S_{n,1}$	14.3573579788208007812	$2.45740 \cdot 10^{-3}$
`single` $S_{n,2}$	14.3926515579223632812	$5.22243 \cdot 10^{-6}$
`double` $S_{n,1}$	14.3927267228647810526	$6.54899 \cdot 10^{-14}$
`double` $S_{n,2}$	14.3927267228657544962	$-2.14449 \cdot 10^{-15}$
`extended` $S_{n,1}$	14.3927267228657233553	$1.91639 \cdot 10^{-17}$
`extended` $S_{n,2}$	14.3927267228657236467	$-1.08475 \cdot 10^{-18}$

Pošto su izračunate vrijednosti različite u svim točnostima, očito asocijativnost zbrajanja **ne vrijedi**.

Zbrajanje unatrag u svim točnostima daje **točniji** rezultat:

- Kod zbrajanja unatrag suma se polako akumulira.
- Kod zbrajanja unaprijed zbroj puno brže raste. Pred kraj, mali članovi jedva utječu na rezultat (prisjetimo se implementacije zbrajanja, **izjednačavanje eksponenta na veći**).

Primjer katastrofalnog kraćenja

Primjer: Zaokruživanjem ulaznih podataka dolazi do male relativne greške (prema konstrukciji). Zanima nas **kako ona utječe na konačni rezultat**.

Promatrajmo realnu aritmetiku računala u bazi 10. Za mantisu imamo $p = 4$ dekadske znamenke, a za eksponent imamo 2 znamenke. Neka je:
$x = 8.8866 = 8.8866 \cdot 10^0$
$y = 8.8844 = 8.8844 \cdot 10^0$
x i y nisu prikazivi u memoriji uz $p = 4$ pa spremamo $fl(x)$ i $fl(y)$.
$fl(x) = 8.887 \cdot 10^0$
$fl(y) = 8.884 \cdot 10^0$
Relativna greška zaokruživanja je mala: $u = 2^{-p} = 5 \cdot 10^{-5}$.
Računamo: $fl(x) - fl(y)$.

Primjer katastrofalnog kraćenja

$fl(x) - fl(y) = 8.887 \cdot 10^0 - 8.884 \cdot 10^0 = 0.003 \cdot 10^0 = 3.??? \cdot 10^{-3}$.
? dobivamo što nakon normalizacije ne možemo rekonstruirati znamenke broja. Računalo umjesto ? zapisuje 0 da rezultat bude točan ukoliko su ulazni argumenti točni.
Konačni rezultat: $fl(x) - fl(y) = 3.000 \cdot 10^{-3}$.
Pravi rezultat:
$x - y = 8.8866 \cdot 10^0 - 8.8844 \cdot 10^0 = 0.0022 \cdot 10^0 = 2.2 \cdot 10^{-3}$.

Uočimo da je već prva značajna znamenka u $fl(x) - fl(y)$ pogrešna. Relativna greška je **velika** ($\varepsilon = 0.3636\dot{3}\dot{6}$). Ukoliko kod narednih aritmetičkih operacija dođe i do kraćenja trojke, dolazi do jako netočnog rezultata.
Veličinu relativne greške ne uzrokuje operacija oduzimanja već **greške u prikazu polaznih operanada**. Uz egzaktno prikazive ulazne operande dolazi do kraćenja ali je rezultat **egzaktan**. Takvo kraćenje zovemo **benigno kraćenje**.

Primjer - kvadratna jednadžba

Treba riješiti realnu kvadratnu jednadžbu $ax^2 + bx + c = 0$ za zadane a, b, c i $a \neq 0$.

Matematički gledano, problem rješavamo računanjem:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Gledano s aspekta numerike realne aritmetike računala:

- ne možemo pretpostaviti da ćemo uopće moći izračunati rješenje po ovoj formuli (moguć **overflow** kod računanja b^2),
- ne možemo pretpostaviti da će dobiveni korijeni biti točni (moguća **kraćenja** kod računanja $b^2 - 4ac$).

Primjer - kvadratna jednadžba

Problem s numerikom možemo malo reducirati dijeljenjem s $a \neq 0$. Time dobijemo normalizirani oblik jednadžbe:
$x^2 + px + q = 0,\ p = \frac{b}{a},\ q = \frac{c}{a}$ Rješenja ove jednadžbe su:

$$x_{1,2} = \frac{-p \pm \sqrt{p^2 - 4q}}{2}$$

U praksi se računanje radi po formuli: $x_{1,2} = -\frac{p}{2} \pm \sqrt{(\frac{p}{2})^2 - q}$.
$p/2$ ili $-p/2$ se izračuna na početku i ne mora se množiti s 4.

Za oduzimanje u diskriminanti s velikim kraćenjem **nema jednostavnog rješenja**. To je odraz **nestabilnosti** problema. Rješenja mogu biti dva po vrijednosti bliska korijena, male promjene u vrijednosti parametra c mogu rezultirati velikom promjenom korijena.

Primjer - kvadratna jednadžba

Primjer: Rješavamo kvadratnu jednadžbu $x^2 + 56x + 1 = 0$.

U dekadskoj aritmetici s 5 značajnih znamenki dobijemo:
$x_1 = 28 - \sqrt{783} = 28 - 27.982 = 0.018000$,
$x_2 = 28 + \sqrt{783} = 28 + 27.982 = 55.982$

Točna rješenja su: $x_1 = 0.0178628\ldots$, $x_2 = 55.982137\ldots$.
Primjetimo: manji korijen po apsolutnoj vrijednosti ima relativnu grešku $7.7 \cdot 10^{-3}$ (**samo dvije točne znamenke**). Veći korijen po apsolutnoj vrijednosti je **apsolutno točan**.

Što možemo učiniti da popravimo rezultat?
Prvo izračunamo veći korijen po apsolutnoj vrijednosti po formuli:
$x_2 = \frac{-(b+sign(b)\sqrt{b^2-4ac})}{2a} = -\frac{p}{2} - sign(p)\sqrt{(\frac{p}{2})^2 - q}$, a manji korijen onda izračunamo korištenjem Viète ovih formula:
$x_1 \cdot x_2 = \frac{c}{a} = q \;\Rightarrow x_1 = \frac{q}{x_2}$ (izbjegli smo opasno kraćenje!).

Promašaj raketa Patriot

U prvom Zaljevskom ratu, rekete **Patriot** nisu uspjele srušiti iračku **Scud** raketu.

Razlog: aritmetika realnih brojeva!

- Računalo koje je upravljalo sustavom je brojilo vrijeme u desetinkama sekunde proteklom od paljenja sustava. $0.1_{10} = (0.0\dot{0}01\dot{1})_2$.
- To računalo je prikazivalo realne brojeve korištenjem normalizirane mantise duljine 23 bita.
- Spremanjem broja 0.1 u registar takvog računala radi se apsolutna greška $\approx 9.5 \cdot 10^{-8}$ sekundi.
- Računalo je bilo u upotrebi 100 sati pa je ukupna greška iznosila $100 \cdot 60 \cdot 60 \cdot 10 \cdot 9.5 \cdot 10^{-8} = 0.34$s.
- Scud raketa putuje brzinom ≈ 1.6 km/s, stoga je Patriot tražio raketu **više od pola kilometra** od svog stvarnog položaja.