Zapis realnih brojeva u računalu - single precision

x-bitna aritmetika (gdje je x djeljivo sa 8 u našem slučaju) je komplicirani nacin za reći: “brojeve (bilo cijele bilo realne) trpamo u x bitova” po od ređenim pravilima ☺

kada spremamo realni broj u 32 bita, onda se exponent-cijeli broj sprema u 8 bitova (ostalo pravilo je - prvi bit je predznak realnog broja, ostalih 23(=32-9) je za prikaz mantise) i tada se takav način zapisa zove single precision, olitiga jednostruka točnost
kada spremamo realni broj u 64 bita, onda se exponent-cijeli broj sprema u 11 bitova (ostalo pravilo je analogno - prvi bit je predznak realnog broja, ostalih 52(=64-12) je za prikaz mantise) i tada se takav nacin zapisa zove double precision, olitiga dvostruka točnost

❶ normalizirani brojevi

normalizirani broj je broj oblika: **1**,blabla · $2^{exponentt}$
normaliziranom mantisom zovemo **1**,blabla i označavamo sa **m**
broj najčešće spremamo u 4 bajta = 32 bita i to ovako:
1 2 --------------9 10 --- 32
|_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_|
1. bit služi za predznak broja kojeg spremamo u računalo
u idućih 8 bitova se sprema exponent (od 2. do 9. bita)
i u iduća 23 bita dolazi mantisa (od 10. do 32. bita), od koje zapravo dolazi samo blabla

broj oblika ±1,blabla · $2^{exponent}$ ćemo spremiti ovako:
|_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_|

predznak je 0 ili 1, 0 je ako je broj pozitivan ($(-1)^0 = 1$), 1 je ako je broj negativan ($(-1)^1 = -1$)
exponent je cijeli broj, no kako u 8 bitova stanu brojevi od 0 do 255, a mi ipak trebamo i negativne brojeve, to dogovorno (za normalizirane brojeve) uzimamo da ćemo ih imati od e_{min} (u oznaci e) do e_{max} (u oznaci E) (te vrijednosti su u našem slučaju, kad imamo zapis broja u 32 bita, e = -126 i E = 127)
no prikaz exponenta NIJE isti kao prikaz cijelog broja u računalu (makar bi vam se tako moglo učiniti ☺)
naime, ideja je da nema negativnih brojeva u exponentu, pa se prema tome primjenjuje sljedeće pravilo/metoda (IEEE standard, više o tome na www.google.com upisati IEEE floating point, ili nešto slično)

> imamo exponent (koji je veći od -126, a manji od 127), mi ćemo mu dodati 127. dobit ćemo neki broj (u svakom slučaju pozitivan i strogo manji od 255) koji ćemo, u binarnom zapisu, utrpati u 8 bitova za exponent. kako je pozitivan, to ga samo pretvorimo u binarni nekom od metoda, npr. dijeljenjem s 2 ili tako nekako

dakle, zapravo, mi u memoriji pamtimo exponente od 1 do 254, ali znamo da su ti exponenti, u stvarnosti, za 127 manji
mantisa je broj oblika 1.blabla, no, upravo zato što je dogovor da je to broj oblika 1.nešto, mi znamo koja je prva znamenka i nećemo ju pamtiti, već pamtimo samo ono nakon točke

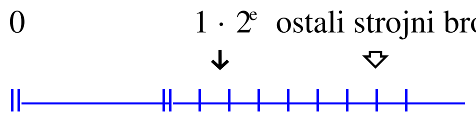
ovo s exponentom je možda još uvijek malo konfuzno
dakle, idem još jednom ukratko probati ponoviti ☺
imamo neki normalizirani broj tipa 1.blabla * 2^{exp}, gdje je exp broj između -126 i 127
mi ćemo u računalo (tj. u onih 8 bitova za exponent) staviti pozitivan broj exp + 127
dakle, ono što stvarno piše u 8 bitova (predvidjenih za exponent) je broj izmedju 1 i 254, a mi znamo da se u stvarnosti radi o broju za 127 manjem

dakle npr. broj -1,10011 · 2^{-7} (gdje je 1,10011 već u bazi 2, a -7 ćemo prebaciti u bazu 2) ćemo spremiti ovako:

|1|0|1|1|1|1|0|0| |0|1|0|0|1|1|0|0| |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0|

pri tome smo s exponentom napravili sljedeće:
dodamo 127 » -7 + 127 = 120
to dobiveno prebacimo u binarni zapis » 111 1000
dopunimo nulama do 8 bitova » 0111 1000
(one koje zanima o kojm je dekadskom broju riječ: -0.012451172 ☺)
dakle, exponenti koji dolaze u obzir za normalizirane brojeve su veći od -126, a manji od 127; u računalo, za eksponente, zapisujemo brojevi od 1 do 254

najmanji normalizirani broj zapisiv u računalu je 1,0000000...000 · 2^e = 1 · 2^e
pogledajmo na brojevnom pravcu kako bi to otprilike izgledalo:



❷ denormalizirani brojevi

denormalizirani broj je broj oblika: **0**,blabla · **2**^{e}
denormaliziranom mantisom zovemo **0**,blabla i označavamo sa **m**

što će nam denormalizirani brojevi?
da bismo dobili veću preciznost među brojevima između 0 i 2^e, valjda ☺

sad se postavlja pitanje kako taj broj zapisati u računalu
naime, spremamo li normalizirani broj, onda to izgleda ovako:

broj **1**, blabla · 2^e

|0|0|0|0|0|0|0|0| |1|b|l|a|b|l|a|0| |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0|

prva ideja koja se nameće kako bismo spremili broj oblika **0**, blabla · 2^e bi možda bila ovo:

|0|0|0|0|0|0|0|0| |1|b|l|a|b|l|a|0| |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0|

no, kad bolje pogledamo, vidimo da imamo identični prikaz za dva potpuno različita broja
(1, blabla · 2^e i **0**, blabla · 2^e)

a to nije ono što smo mi htjeli kad smo dopustili i denormalizirane mantise
no, sjetimo se, da u 8 bitova možemo pospremiti 256 brojeva, a u rasponu od e do E (tj od e_{min}= -126 do e_{max}= 127) smo "potrošili" tek 254 broja, ostala su nam još dva, -127 = e_{min}-1 i 128 = e_{max}+1
dogovorno je da će se, u slučaju denormaliziranih mantisa kao OZNAKA da bismo znali da se radi upravo o denormaliziranom broju, upotrebljavati e-1, tj. e_{min} -1, tj. -127 (u računalo upisujemo broj za 127 veći, isto je pravilo kao i kod normaliziranih, dakle upisujemo 0)

znači, broj u računalu zapisan kao
|0|0|0|0|0|0|0|0| |0|b|l|a|b|l|a|0| |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0|

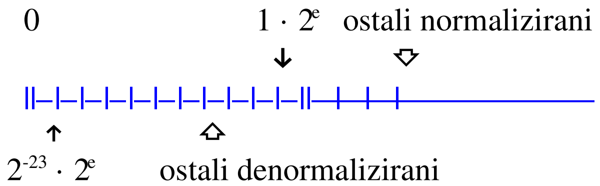
je u stvari broj **0**, blabla · 2^e

dakle u polja |_|_|_|_|_|_|_| |_|
u prikazu na način |_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_|
se **UPISUJE** e_{min} -1(tj. prikaz broja -127 binarno u 8 bitova – tj -127 + 127 = 0 u 8 bitova)
a **ZNAČI** da se radi o denormaliziranom broju čiji je exponent **NUŽNO** $\mathbf{2^e}$

kad u poljima |_|_|_|_|_|_|_| |_| vidite broj 0000000 0, to znači
da imate posla s denormaliziranim brojem i da mu je mantisa:
0,broj u poljima |_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_|
Ako je mantisa, npr, blebl, onda je taj broj 0,blebl · 2^e

svaki denormalizirani broj je oblika mantisa · 2^e

najmanji denormalizirani upisivi broj je
0,00000...0001· 2^e što je, kad ga ljepše zapišemo, u stvari broj $2^{-23} \cdot 2^e$
pogledajmo opet malo brojevni pravac



također, razlika dva susjedna broja (npr broj i njegov sljedbenik) je očito 2^{-23} ☺ (raspisano je)
|x|y|y|y|y|y|y|y| |y|b|l|a|b|l|a|z| |z|z|z|z|z|z|z|z| |z|z|z|z|z|z|z|0| ← broj
|x|y|y|y|y|y|y|y| |y|b|l|a|b|l|a|z| |z|z|z|z|z|z|z|z| |z|z|z|z|z|z|z|1| ← sljedbenik

znači, u ovom primjeru sve im je isto osim zadnje znamenke mantise
kako mantisa ima 23 mjesta, i kad bi od sljedbenika oduzeli broj, dobili bismo upravo mantisu koja ima 22 nule i 1 na kraju, što je mantisa "reda" 2^{-23}

prikaz nule: |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0|

nula je u stvari broj **0**,0..0 · 2^e

no, i ovo je nula (zapravo -0 ☺)

|1|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0| |0|0|0|0|0|0|0|0|

par exponenata:

0000 0001 = -126 = e = e_{min}	1111 1110 = 127 = E = e_{max}
0000 0000 = -127 = e_{min} – 1	1111 1111 = 128 ← upotrebljava se za prikaz ∞

idemo dalje,

$x = m \cdot 2^{exponent}$

$x_s = (m + 2^{-23}) \cdot 2^{exponent}$

$x_p = (m - 2^{-23}) \cdot 2^{exponent}$

ovo nam je trebalo da bi pričali malo o relativnom razmaku

$$\frac{|x_s - x|}{|x|} = \frac{(m+2^{-23})\cdot 2^{exponent} - m\cdot 2^{exponent}}{m\cdot 2^{exponent}} = \frac{m\cdot 2^{exponent} + 2^{-23}\cdot 2^{exponent} - m\cdot 2^{exponent}}{m\cdot 2^{exponent}} =$$

$$= \frac{2^{-23}\cdot 2^{exponent}}{m\cdot 2^{exponent}} = \frac{2^{-23}}{m}$$

za normalizirane brojeve:

$$\frac{2^{-23}}{m}\{\text{ jer je } m \geq 1\} \leq 2^{-23}$$

(ovo znači da je 2^{-23} gornja ograda za taj kvocijent, što vrijedi jer je kvocijent najveći kad ga dijelimo sa najmanjim brojem, a najmanji mogući u ovom slučaju je upravo 1tad je mantisa 1,0...0)

za denormalizirane brojeve je (ovo se kao navodno može pokazati)

$$\frac{2^{-23}}{m}\{\text{ jer je } m < 1\} = 1$$

što će nam taj relativni razmak?

pa, to znači da je greška zaokruživanja sigurno ⩽ od pola toga, jer naime, sav taj prikaz je ograničen, i skup brojeva sa kojima mi uopće raspolažemo je konačan (a znamo da realnih brojeva ima beskonačno i još malo više ☺)

ukratko, pregled o realnim brojevima (sa http://www.psc.edu/general/software/packages/ieee/ieee.html)

Single Precision

The IEEE single precision floating point standard representation requires a 32 bit word, which may be represented as numbered from 0 to 31, left to right. The first bit is the sign bit, S, the next eight bits are the exponent bits, 'E' and the final 23 bits are the fraction 'F':

```
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
0 1      8 9                    31
```

The value V represented by the word may be determined as follows:

- If E=255 and F is nonzero, then V=NaN ("Not a number")
- If E=255 and F is zero and S is 1, then V=-Infinity
- If E=255 and F is zero and S is 0, then V=Infinity
- If 0<E<255 then V=(-1)**S * 2 ** (E-127) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-126) * (0.F) These are "unnormalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

In particular,

```
0 00000000 00000000000000000000000 = 0
1 00000000 00000000000000000000000 = -0

0 11111111 00000000000000000000000 = Infinity
1 11111111 00000000000000000000000 = -Infinity

0 11111111 00000100000000000000000 = NaN (Not a Number)
1 11111111 00100010001001010101010 = NaN (Not a Number)

0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2
0 10000001 10100000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5
1 10000001 10100000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5

0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)
0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)
0 00000000 00000000000000000000001 = +1 * 2**(-126) *
                                     0.00000000000000000000001 =
                                     2**(-149)  (Smallest positive value)
```

Double Precision

The IEEE double precision floating point standard representation requires a 64 bit word, which may be represented as numbered from 0 to 63, left to right. The first bit is the sign bit, S, the next eleven bits are the exponent bits, 'E' and the final 52 bits are the fraction 'F':

```
S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
0 1        11 12                                                63
```

The value V represented by the word may be determined as follows:

- If E=2047 and F is nonzero, then V=NaN ("Not a number")
- If E=2047 and F is zero and S is 1, then V=-Infinity
- If E=2047 and F is zero and S is 0, then V=Infinity
- If 0<E<2047 then V=(-1)**S * 2 ** (E-1023) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-1022) * (0.F) These are "unnormalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

Zapis cijelih brojeva u računalu

recimo da ga prikazujemo u 16 bitova, dakle u ovako nečemu:
|_|_|_|_|_|_|_|_| |_|_|_|_|_|_|_|_|

pozitivni cijeli broj prikazujemo tako da ga samo pretvorimo u binarni i dopunimo nulama, npr
50 » 11 0010 » 0000 0000 0011 0010
dakle |0|0|0|0|0|0|0|0| |0|0|1|1|0|0|1|0|

negativni broj prikazujemo pomoću dvojnog komplementa

za one koji su malo zaboravili ☺, **dvojni komplement** se radi ovako:

broj -x (gdje je x pozitivan broj) želimo prikazati u npr 8 bitova
[npr -7]
x prebacimo u binarni zapis u 8 bitova (samo ga pretvorimo u binarni i dodamo ispred onoliko nula koliko treba da bi popunili svih 8 bitova)
[znači 7 » 111 » 0000 0111]
e, sad mu tražimo dvojni komplement tako da zamijenimo 0 sa 1 i 1 sa 0, (znači 0 «-» 1)
[0000 0111 » 1111 1000]
i dodamo tom broju 1
[1111 1000 » 1111 1001 ← ovo je broj -7 prikazan u 8 bitova]

dakle, recimo, sad hoćemo -50 prikazati u računalu (u 16 bitova)
50 » 11 0010 » 0000 0000 0011 0010 » 1111 1111 1100 1101 » 1111 1111 1100 1110
|1|1|1|1|1|1|1|1| |1|1|0|0|1|1|1|0|

that's all folks! ☺