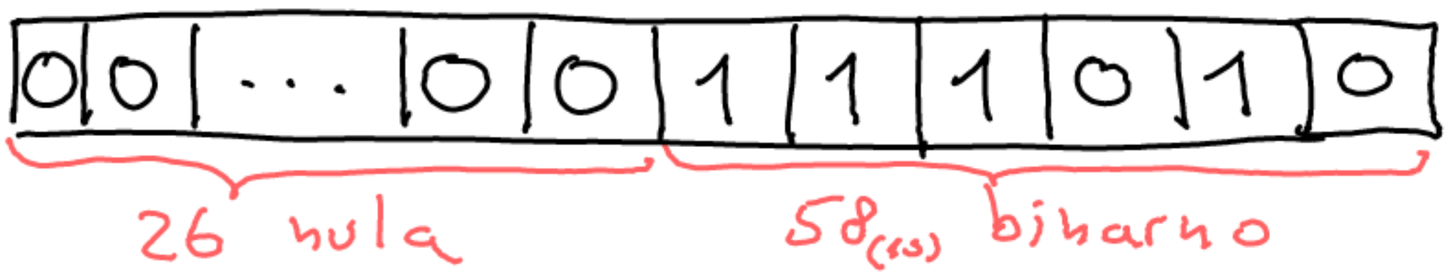
CIJELI BROJEVI U RAČUNALU

→ binarni sustav, ali sa FIKSNIM brojem bitova

→ tipično 4B = 32b za cijele brojeve, iako ovisi o arhitekturi računala i tipu podataka ...

→ tako da u računalnoj memoriji npr. $58_{(10)}$ NIJE

111010 nego je



To vrijedi samo za POZITIVNE cijele brojeve, a kako su zapisani NEGATIVNI ... ??

⇒ DVOJNI KOMPLEMENT $(\overline{N}+1)$

komplement binarnog broja = zamjena „0" s „1" i obrnuto

npr.

$N = 00111010 = +58_{(10)}$

$\overline{N} = 11000101$

$+\ 00000001$

$11000110 = -58_{(10)}$ (u računalima, ne matematički)

primjetite: pozitivni broj „počinje" s „0",
negativni broj „počinje" s „1"

Zašto bi to tako radili ???

→ <u>zbog oduzimanja</u> (tj. zbrajanja) !

primjer:

```
  1     1 1
  00110011   (51)
+ 00101001   (41)
------------
  01011100   (92)
```

(tu su 8-bitni brojevi
iako su u stvarnosti
u pravilu 32-bitni)

```
  00101001   (41)
------------
  11010110
+ 00000001
------------
  11010111   (-41)
```

2. komplement

```
  00101000
+ 00000001
------------
  00101001   (41)
```

2. komplement

```
  1 1   1   1 1   1
  00110011   (51)
+ 11010111   (-41)
------------
1 00001010   (10)
```

8 bitova

višak (tzv. overflow, tj. preljev)

→ 2. komplement 2. komplementa poništava operaciju (kako i treba... $-(-x)=x$)

Znači, zato što smo ograničeni brojem bitova za zapis cijelih brojeva, zbrajanje sa 2. komplementom funkcionira kao oduzimanje, odnosno NEGATIVNE brojeve možemo zapisivati 2. komplementom pozitivnih brojeva i onda nam treba samo operacija zbrajanja !

Posljedica: za zapis sa **n** bitova max broj je $2^{n-1}-1$, a min broj je -2^{n-1} !

npr. za n=8: $N_{max} = 2^7 - 1 = 127 \rightarrow 0111\ 1111$

$N_{min} = -2^7 = -128 \rightarrow 1000\ 0000$

Kako znamo da je broj negativan ili pozitivan?

→ prema prvom (najljevijem) bitu !

Na primjer:

$$10001001 \xrightarrow{\text{2. komplement}} \begin{array}{r} 01110110 \\ +\ 00000001 \\ \hline 01110111 \end{array} = 119_{(10)}$$

1 → negativan broj!

→ dakle $\boxed{10001001 = -119_{(10)}}$

... pa istom logikom i ...

$$10000000 \xrightarrow{\text{2. komplement}} \begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1} \\ 01111111 \\ +\ 00000001 \\ \hline 10000000 \end{array} = 128_{(10)}$$

1 → negativan broj!

→ dakle $\boxed{10000000 = -128_{(10)}}$

$$00110110 \quad \boxed{= 54_{(10)}}$$

0 → pozitivan broj!

Što ako bi zbrajanjem dobili broj veći od najvećeg mogućeg, odnosno izvan raspona $[-2^{n-1}, 2^{n-1}-1]$??

⇒ prelijev preko kraja raspona na početak ⇒

⇒ negativni broj ⇒ <u>nebulozni rezultati</u> !

(tako se detektira da se to dogodilo, od strane jezgre, računalu je tu sve ok)

npr.

$$\begin{array}{rl} 0111\,1111 & = 127_{(10)} \\ +\ 0000\,0001 & \\ \hline 1000\,0000 & = -128_{(10)} \end{array}$$

⇒ raspon je cikličan !

$$\begin{array}{rl} \overset{1\ 1\ 1}{0}0111001 & = 57_{(10)} \\ +\ 01110010 & = 114_{(10)} \\ \hline 10101011 & = -85_{(10)} \end{array}$$

, a trebalo bi biti 171, što je 44 preko 127, pa dalje

$-128 + 43 = 85$

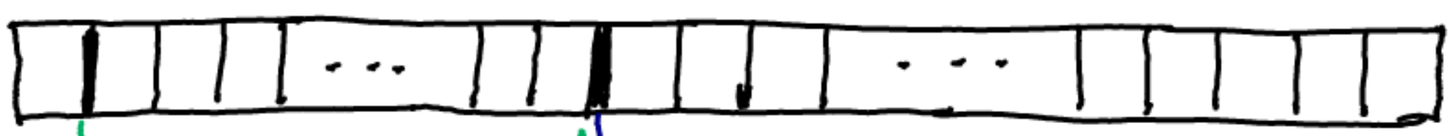
$$\begin{array}{l} \downarrow \\ 01010100 \\ +\qquad\quad 1 \\ \hline 01010101 = 85_{(10)} \end{array}$$

DECIMALNI BROJEVI U RAČUNALU

od „floating point number", jer dec. točka ovisi o eksponentu

- IEEE 754 standard
  - single precision (32 bita = 4B) - tzv. FLOAT
  - double precision (64 bita = 8B) - tzv. DOUBLE

zapis u memoriji:



PREDZNAK (1b): $0 = „+", 1 = „-"$, tj. $\times (-1)^b$

EKSPONENT (8/11b): „exp" = $N - (2^{n-1} - 1)$

- N = prirodni binarni broj zapisan sa tih 8/11 bitova
- n = broj bitova (8 ili 11)

BROJ (preciznost; 23/52 b): $1.b_{22}b_{21}\dots b_1b_0$

decimalni binarni broj zapisan sa tih 23/52 bitova...

→ SVE SKUPA:

$$B = (-1)^b \times 2^{(N-2^n+1)} \times 1.b_x \dots b_1 b_0$$!

Na primjer: 1:01111001 0:10101000000000000000000

$$= (-1)^1 \cdot \left(1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{32}\right) \cdot 2^{(64+32+16+2-127)} = -1.65625 \cdot 2^{-13} \approx -2.02179 \cdot 10^{-4}$$

– u DOUBLE formatu (64b = 8B):

→ najveći mogući broj (po iznosu): $\pm 1.\underbrace{99\ldots9}_{52} \cdot 2^{1023} \approx 1.7977 \cdot 10^{308}$

→ najmanji mogući broj (po iznosu): $\pm 1.0 \cdot 2^{-1022} \approx 2.2251 \cdot 10^{-308}$

(odnosno, za posebne „subnormal" brojeve, $\pm 2^{-1074} \approx 4.94 \cdot 10^{-324}$)

→ posebni brojevi: $\pm\infty$ → kada je exp: 11111111111, a sve ostalo 000…000

NaN → kada je exp: 11111111111, a ostalo je ≠ 000…000

↳ NOT A NUMBER – kada se dogodi greška u računu, npr. dijeljenje s nulom i slično…

!

ZNAKOVI U RAČUNALU

→ svaki SIMBOL je KODIRAN s točno određenom količinom i kombinacijom bitova ⇒ KODNE TABLICE !

→ ASCII – najstariji standard (i dalje aktualan)
- engleska abeceda i razni posebni znakovi ...
- svaki simbol = 7 bitova tj. 1B (1. bit 0)

npr. „0" = $48_{(10)}$ = $00110000_{(2)}$
„9" = $57_{(10)}$ = $00111001_{(2)}$
„A" = $65_{(10)}$ = $01000001_{(2)}$
„a" = $97_{(10)}$ = $01100001_{(2)}$
„#" = $35_{(10)}$ = $00100011_{(2)}$

nevidljivi i ostali „posebni" znakovi se u programiranju označavaju s „\"

„\n" = $10_{(10)}$ = $00001010_{(2)}$ → novi redak
„ " = $32_{(10)}$ = $00100000_{(2)}$ → razmak

to se zove BACKWARD COMPATIBLE

→ UTF-8 – moderni standard (opće prihvaćen)
- identičan ASCII-ju za tih $2^7 = 128$ znakova
- ako je prvi bit „1" onda nije ASCII znak ...
- ne-ASCII znakovi se zapisuju s 2, 3 ili 4 bajta ...
- mogu se zapisati „SVI MOGUĆI POZNATI ZNAKOVI"

→ kada se neka datoteka pokušava pročitati kao tekstualna (dakle bitovi interpretirati kao znakovi) NUŽNO je da računalo zna po kojoj kodnoj tablici (encoding) su ti znakovi zapisani, inače ih neće točno pročitati / prikazati (osim ako su ASCII - to će pretpostaviti)

→ dakle treba paziti na to kada se radi sa tekstom / datotekama (čita i piše)