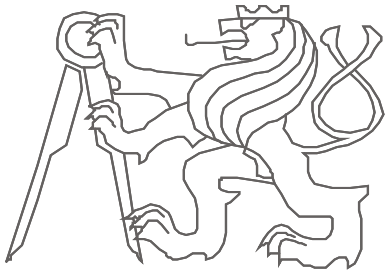
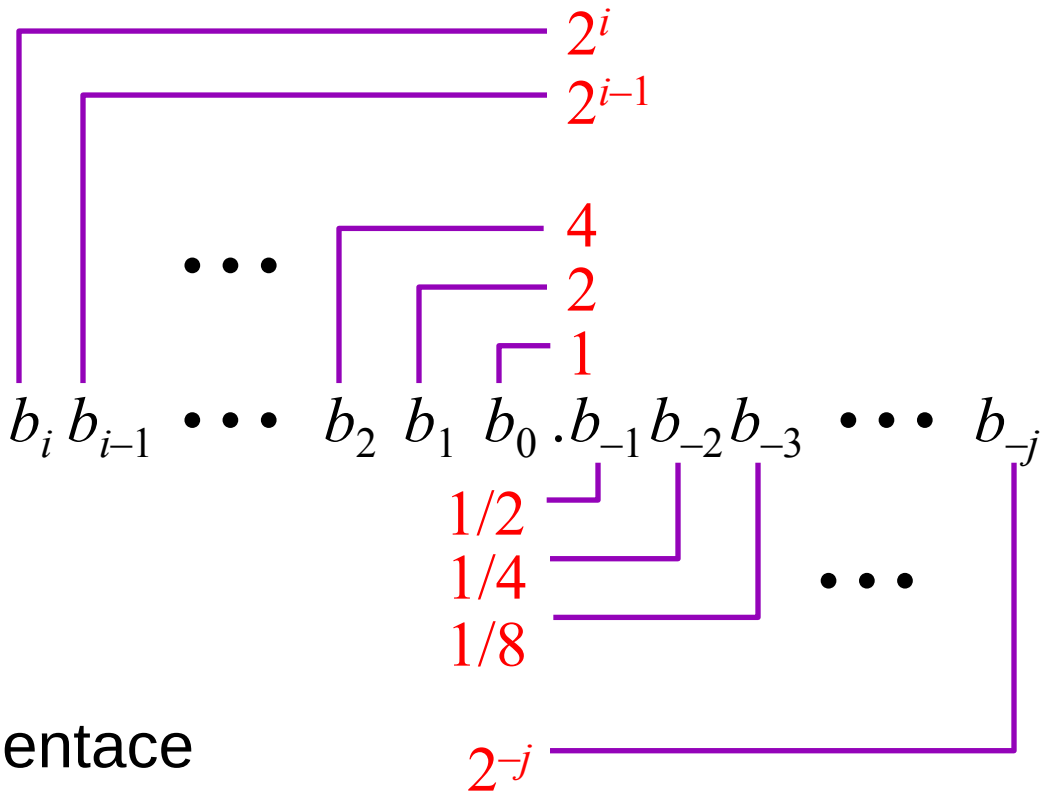
Architektury počítačů

IEEE754



České vysoké učení technické, Fakulta elektrotechnická

Fractional Binary Numbers
(zlomková binární čísla / čísla v pevné řádové čárce)



- Reprezentace
  - bity vpravo od “binary point” udávají zlomky mocnin 2
  - reprezentuje reálná čísla

$$\sum_{k=-j}^{i} b_k \cdot 2^k$$

Zlomková čísla / Pevná řádová čárka

- *Hodnota* *Reprezentace*

  5-3/4 101.11_2

  2-7/8 10.111_2

  63/64 0.111111_2

- *Operace*
  - Dělení 2 posunem vpravo
  - Násobení 2 posunem vlevo
  - Čísla pod $0.111111\ldots_2$ jsou menší než 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Přesná notace $\rightarrow 1.0 - \varepsilon$

Binární➔ Dekadické

$23.47 = 2\times10^{1} + 3\times10^{0} + 4\times10^{-1} + 7\times10^{-2}$

↑ desetinná tečka

$10.01_{two} = 1\times2^{1} + 0\times2^{0} + 0\times2^{-1} + 1\times2^{-2}$

↑ binární tečka

$= 1\times2 + 0\times1 + 0\times\frac{1}{2} + 1\times\frac{1}{4}$

$= 2 + 0.25 = 2.25$

Dekadické ➔ Binární

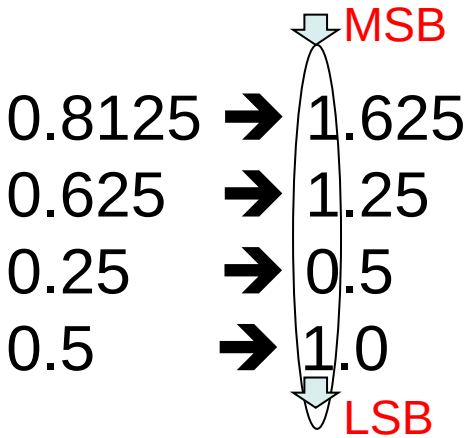
- Možno napsat číslo jako součet mocnin 2

  $0.8125 = 0.5 + 0.25 + 0.0625$

  $= 2^{-1} + 2^{-2} + 2^{-4}$

  $= 0.1101_2$

- Algoritmus: Opakovaně násobíme zlomkovou část 2, dokud nebude 0 - *násobením provádíme simulaci posunu vlevo*:



Připomínka: Převod celých čísel bez znaménka

- Desítkové číslo: 13

b) dělením 2

13/2	= 6	zbytek 1	LSB
6/2	= 3	zbytek 0	
3/2	= 1	zbytek 1	
1/2	= 1	zbytek 1	MSB

Připomínka: V předmětu SPS se ukazovalo, že celá část čísla se převede dělením 2 a zbytku po dělení - simulace posunu vpravo, a proto je opačné pořadí bitů

Snímek převzatý z 1. přednášky SPS

a) postupným odečítáním *mocnin 2 postupně od největší k nejmenší*

32	16	8	4	2	1	
0						$32=2^5$ větší
0	0					$16=2^4$ větší
0	0	1				$8=2^3$ ok, 13-8=5
0	0	1	1			$4=2^2$ =8+4=12 ok, 5-4=1
0	0	1	1			$2=2^1$ větší než 1
0	**0**	**1**	**1**	**0**	**1**	$1=2^0$ =8+4+1=13 1-1=0

Pozor

- Konečné dekadické číslo
  ➔ konečné binární číslo
- Příklad:

0.1_{ten} ➔ 0.2 ➔ 0.4 ➔ 0.8 ➔ 1.6 ➔ 1.2 ➔ 0.4 ➔
0.8 ➔ 1.6 ➔ 1.2 ➔ 0.4 …

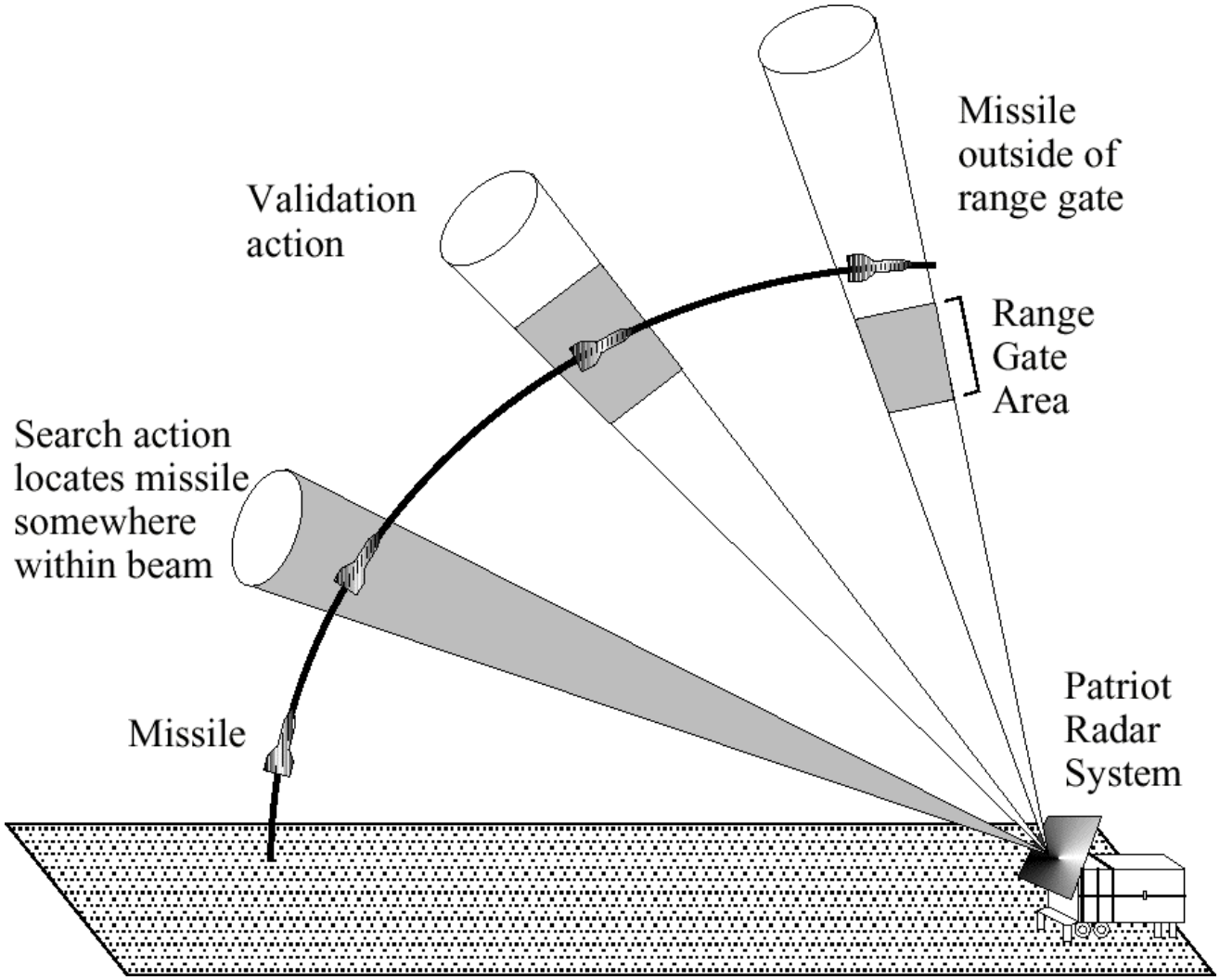
$0.1_{10} = 0.0001100110011001100 11\ldots_2$

$= 0.0\underline{0011}_2$ (nekonečný řetězec bitů)

S více bity se zpřesňuje reprezentace 0.1_{10}

Effect of Loss of Precision

According to the General Accounting Office of the U.S. Government, a loss of precision in converting 24-bit integers into 24-bit floating point numbers was responsible for the failure of a Patriot anti-missile battery.



Slide source: UIUC

Effect of Loss of Precision

- During the Gulf War in 1991, a U.S. Patriot missile failed to intercept an Iraqi Scud missile, and 28 Americans were killed.
- A later study determined that the problem was caused by the inaccuracy of the binary representation of 0.10.
  - The Patriot incremented a counter once every 0.10 seconds.
  - It multiplied the counter value by 0.10 to compute the actual time.
- However, the (24-bit) binary representation of 0.10 actually corresponds to 0.099999904632568359375, which is off by 0.000000095367431640625.
- This doesn't seem like much, but after 100 hours the time ends up being off by 0.34 seconds—enough time for a Scud to travel 500 meters!
- UIUC Emeritus Professor Skeel wrote a short article about this.

Roundoff Error and the Patriot Missile. SIAM News, 25(4):11, July 1992.

Slide source: UIUC

Reprezentace

- *Omezení*
  - lze přesně vyjádřit jen čísla $x/2^k$
  - Ostatní čísla jsou uložená nepřesně
- *Hodnota* *Reprezentace*

1/3	$0.0101010101[01]\ldots_2$
1/5	$0.0011001100 11[0011]\ldots_2$
1/10	$0.0001100110011[0011]\ldots_2$

Vědecká notace

- *Dekadické číslo:*

  -123,000,000,000,000 ➔ -1.23×10^{14}

  0.000 000 000 000 000 123 ➔ $+1.23\times 10^{-16}$

- *Binární číslo:*

  110 1100 0000 0000 ➔ 1.1011×2^{14}

  -0.0000 0000 0000 0001 1011 ➔ -1.1101×2^{-16}

HW/SW interfejs

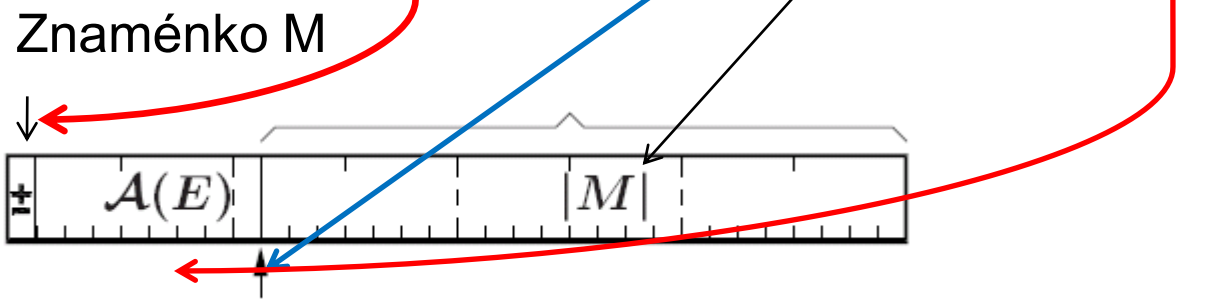
- Normalizováno jako IEEE754 ve verzích
  - Jednoduchá,
  - Dvojnásobná přesnost.
- V programovacím jazyce C se deklaruje jako `float` a `double`.

Příklady

- -2.34×10^{56} ← normalized
- $+0.002 \times 10^{-4}$ ← not normalized
- $+987.02 \times 10^{9}$ ← not normalized

binárně

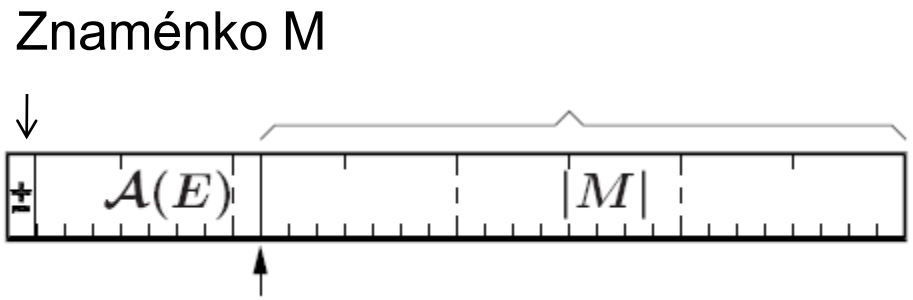
- $\pm 1.xxxxxxx_2 \times 2^{yyyy}$



Poloha řádové čárky (tečky) pro E i M

Formát čísla v pohyblivé řádové čárce

- Kód mantisy: přímý kód — znaménko a absolutní hodnota
- Kód exponentu: aditivní kód (s posunutou nulou - posun +127).



Poloha řádové čárky (tečky) pro E i M

Příklady reprezentace některých důležitých hodnot

Nula

Kladná nula	0 00000000 00000000000000000000000	+0.0
Záporná nula	1 00000000 00000000000000000000000	-0.0

Nekonečno

Kladné nekonečno	0 11111111 00000000000000000000000
Záporné nekonečno	1 11111111 00000000000000000000000

Některé krajní hodnoty

Nejmenší normalizované číslo	* 00000001 00000000000000000000000	$\pm 2^{(1-127)}$
Největší denormalizované číslo	* 00000000 11111111111111111111111	$\pm(1-2^{-23})*2^{-126}$
Nejmenší denormalizované číslo	* 00000000 00000000000000000000001	$\pm 2^{-23}*2^{-126}$

Skrytý bit

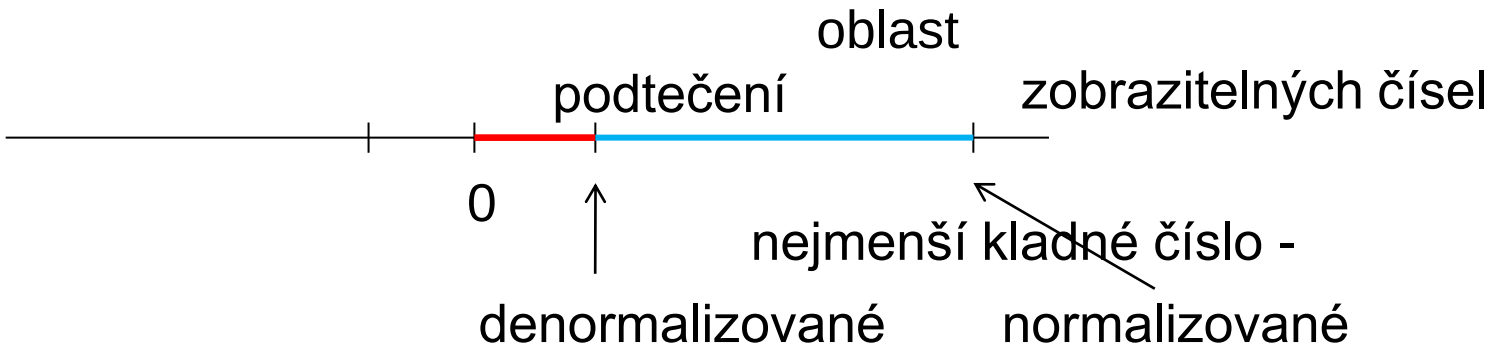
- Nejvyšší platný bit mantisy (který se do bitové reprezentace operandu neukládá) je
- závislý na hodnotě obrazu exponentu. Jestliže je obraz exponentu nenulový, je tento bit
- 1, mluvíme o **normalizovaných** číslech.
- Na druhou stranu jestliže je obraz exponentu nulový, je skrytý bit 0.
- Pak mluvíme o **denormalizovaném** čísle.

Denormalizované číslo?

- Smyslem zavedení denormalizovaných čísel je rozšíření reprezentovatelnosti čísel, která se nacházejí blíže k nule, tedy čísel velmi malých (v následujícím obrázku oblast označena modře).
- Denormalizovaná čísla mají nulový exponent a i skrytý bit před řádovou čárkou je implicitně nulový.
- Cenou je nutnost speciálního ošetření případu nulový exponent, nenulová mantisa.

Podtečení

- Jde o situaci, kdy zobrazované číslo není rovno nule, ale nedosahuje hodnoty nejmenšího zobrazitelného normalizovaného čísla.
- Mnoha podtečením lze předejít právě implementací čísel denormalizovaných.



Přehled

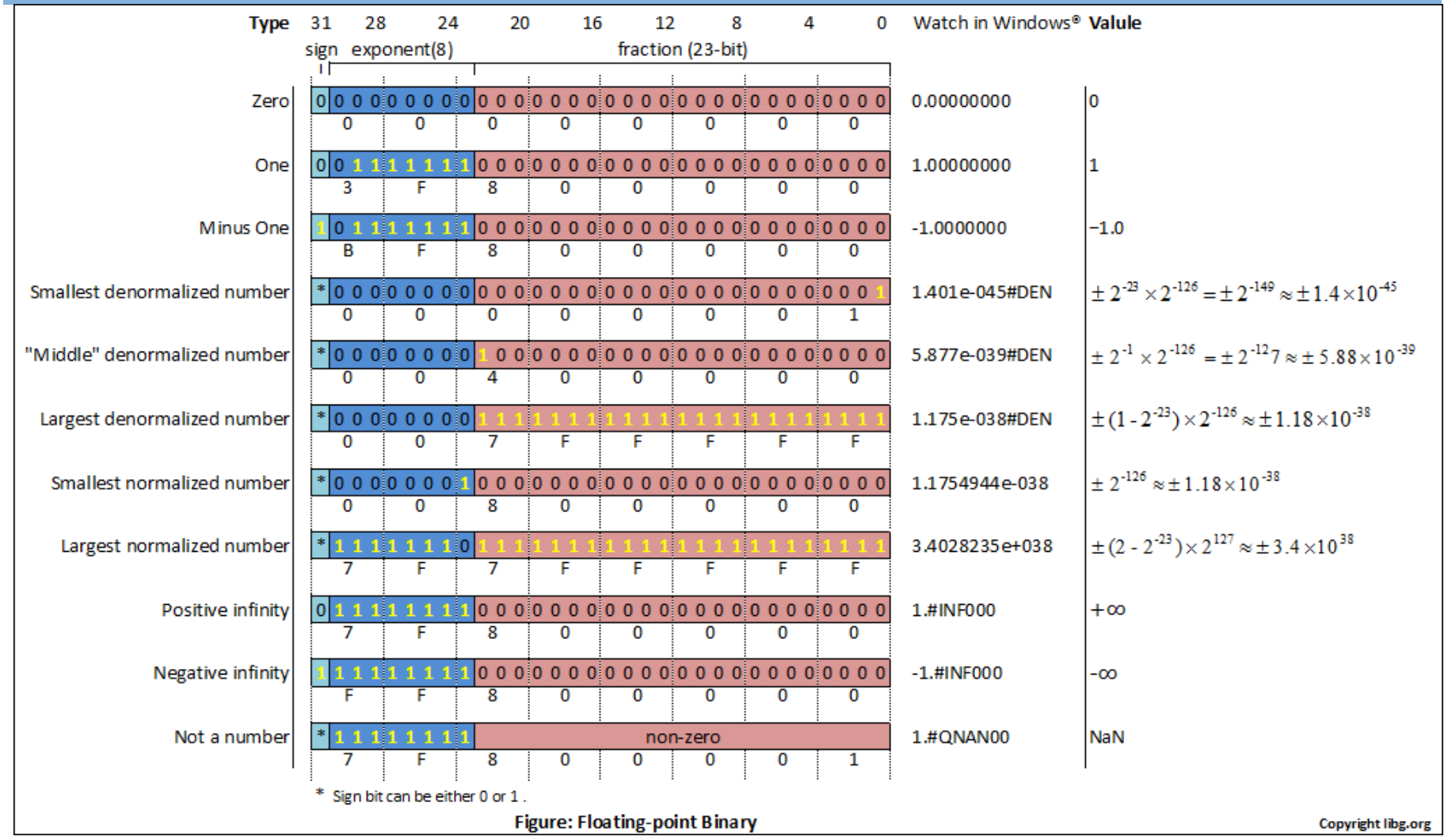


Figure: Floating-point Binary

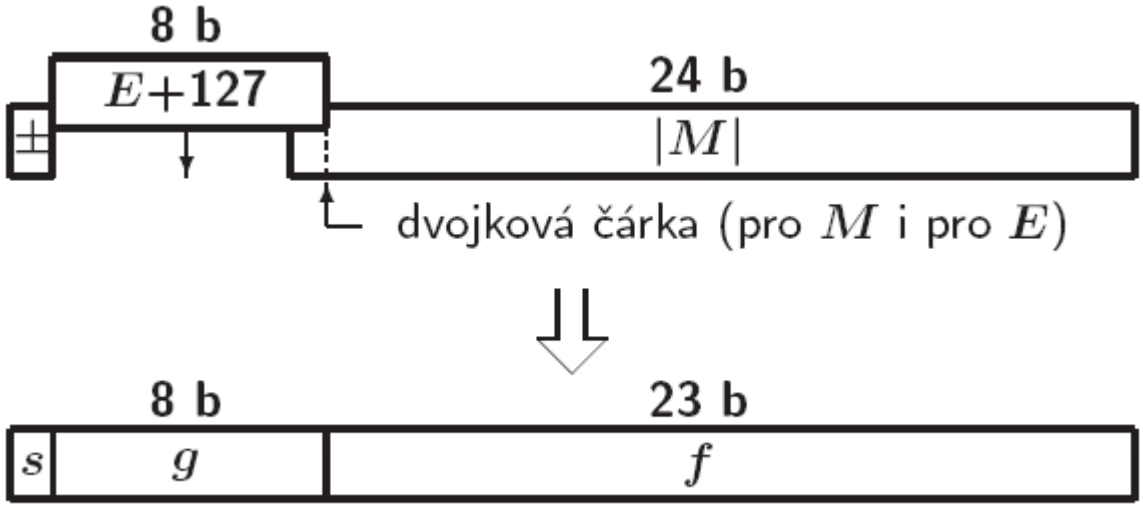
NaN

- Samé jedničky v exponentu,
- Mantisa je nenulová.
- V tom případě není bitový řetězec obrazem žádného čísla, anglicky Not-a-Number, Nan.

Shrnutí – zobrazitelná čísla a výjimky

s-bit	Obraz exponentu	m	význam
0	0<e<255	>0	normalizované kladné číslo
1	0<e<255	>0	normalizované záporné číslo
0	0	>0	denormalizované kladné číslo
1	0	>0	denormalizované záporné číslo
0	0	0	kladná nula
1	0	0	záporná nula
0	255	0	kladné nekonečno (overflow)
1	255	0	záporné nekonečno (overflow)
0	255	≠0	NaN – not a number – nečíselná hodnota
1	255	≠0	NaN – not a number – nečíselná hodnota

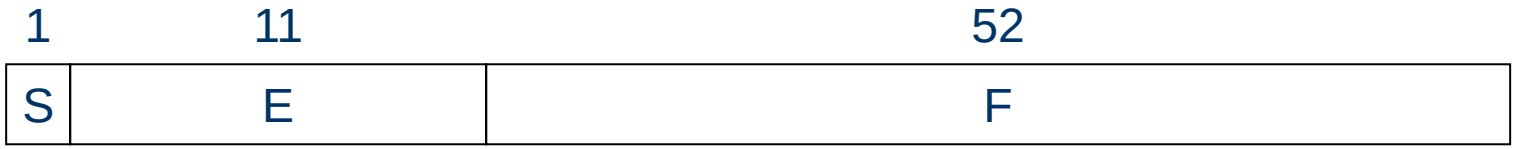
ANSI/IEEE Std 754-1985 formáty – 32b a 64b



ANSI/IEEE Std 754-1985 — dvojitý formát — 64b

g . . . 11b *f* . . . 52b

double



- E není 00…0 (dekadicky 0)
  nebo 11…1 (dekadicky 2047)
- Normalized rule: number represented is

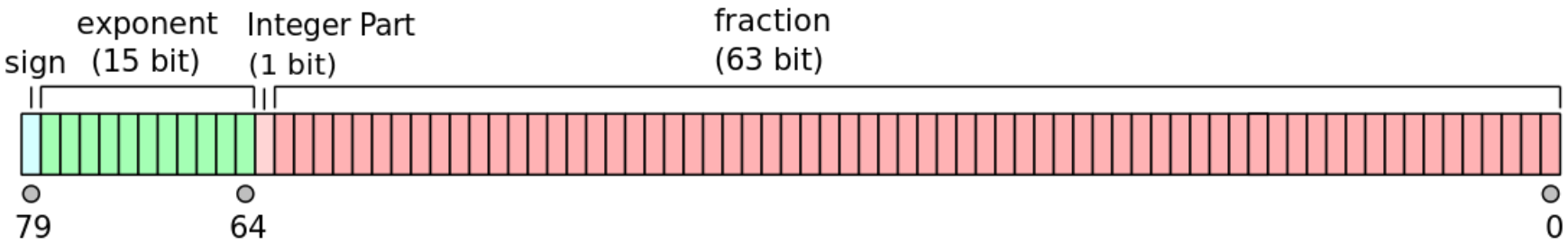
$$(-1)^{S} \times 1.F \times 2^{E-1023}$$

ANSI/IEEE Std 754-1985 — dvojitý formát — 64 bitů

E . . . 11 bitů *f* . . . 52 bitů

x86 Extended Precision Format

používá aritmetický koprocesor x86



3.65×10^{-4951} to 1.18×10^{4932}

Zdroj: http://en.wikipedia.org/wiki/Extended_precision

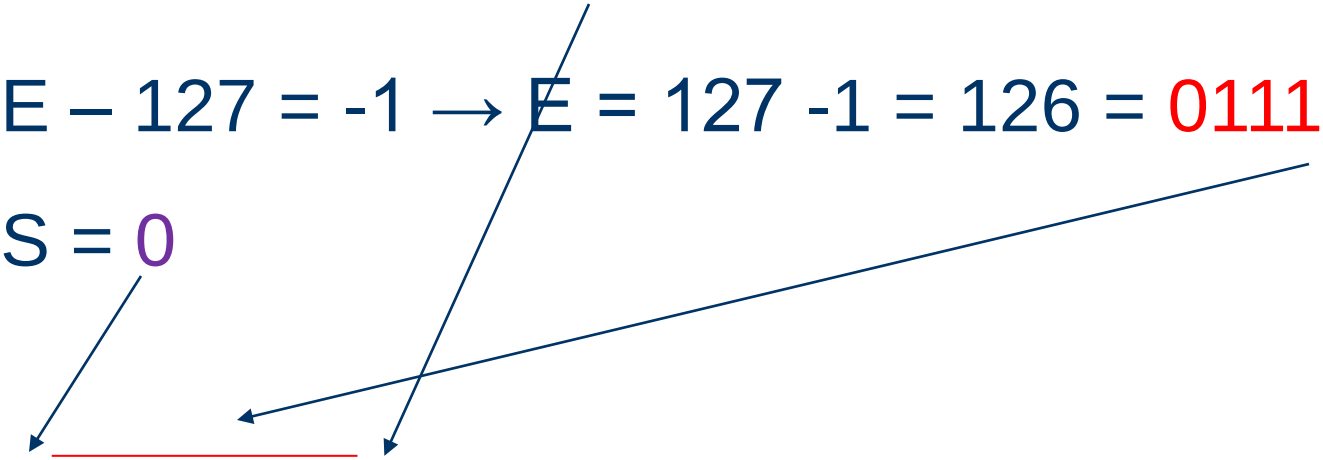
Příklad: 0.75

$0.75_{10} = 0.11_{2} = 1.1 \times 2^{-1}$

$1.1 = 1.\ F \rightarrow F = 1$

$E - 127 = -1 \rightarrow E = 127 - 1 = 126 = 01111110_2$

$S = 0$



$00111111010000000000000000000000 = 0x3F400000$

Příklad 0.1_{10} - Převeďte na float

$0.1_{10} = 0.\underline{0001100 11}...._2 =$

0.0001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011...

Příklad 0.1_{10} - Převeďte na float

$0.1_{10} = 0.000110011...._2$

$= 1.10011_2 \times 2^{-4} = 1.F \times 2^{E-127}$

F = 10011 -4 = E – 127

E = 127 -4 = 123 = 01111011_2

0011 1101 1100 1100 1100 1100 1100 1100 *1100 11..*

0x3DCCCCCD, proč je D ?

IEEE-754 konverze

- Převeďte -12.625_{10} IEEE-754 float formát.
- Krok #1: Převeďte $-12.625_{10} = -1100.101_2$
- Krok #2: Normalizujte $-1100.101_2 = -1.100101_2 \times 2^3$
- Krok #3: Vyplňte pole, znaménko je záporné -> S=1.
  Exponent + 127 -> 130 -> 1000 0010 .
  Úvodní bit 1 mantisy je skrytý ->

1 1000 0010 . 1001 0100 0000 0000 0000 000

Kvíz: Rozhodněte o platnosti vztahů

```
int x = …;
float f = …;
double d = …;
```

Předpokládejme, že `d` a `f` nejsou NAN

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0 ⇒ ((d*2) < 0.0)`
- `d > f ⇒ -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

Odpovědi na kvíz

```
int x = …;
float f = …;
double d = …;
```

Předpokládejme,
že `d` a `f` nejsou NAN

- `x == (int)(float) x` — Ne: 24 významných bitů
- `x == (int)(double) x` — Ano: 53 významných bitů
- `f == (float)(double) f` — Ano : zvýšení přesnosti
- `d == (float) d` — Ne: ztráta přesnosti
- `f == -(-f);` — Ano : pouhá změna znaménka
- `2/3 == 2/3.0` — Ne: 2/3 == 0
- `d < 0.0 ⇒ ((d*2) < 0.0)` — Ano!
- `d > f ⇒ -f > -d` — Ano!
- `d * d >= 0.0` — Ano!
- `(d+f)-d == f` — Ne: Není asociativní

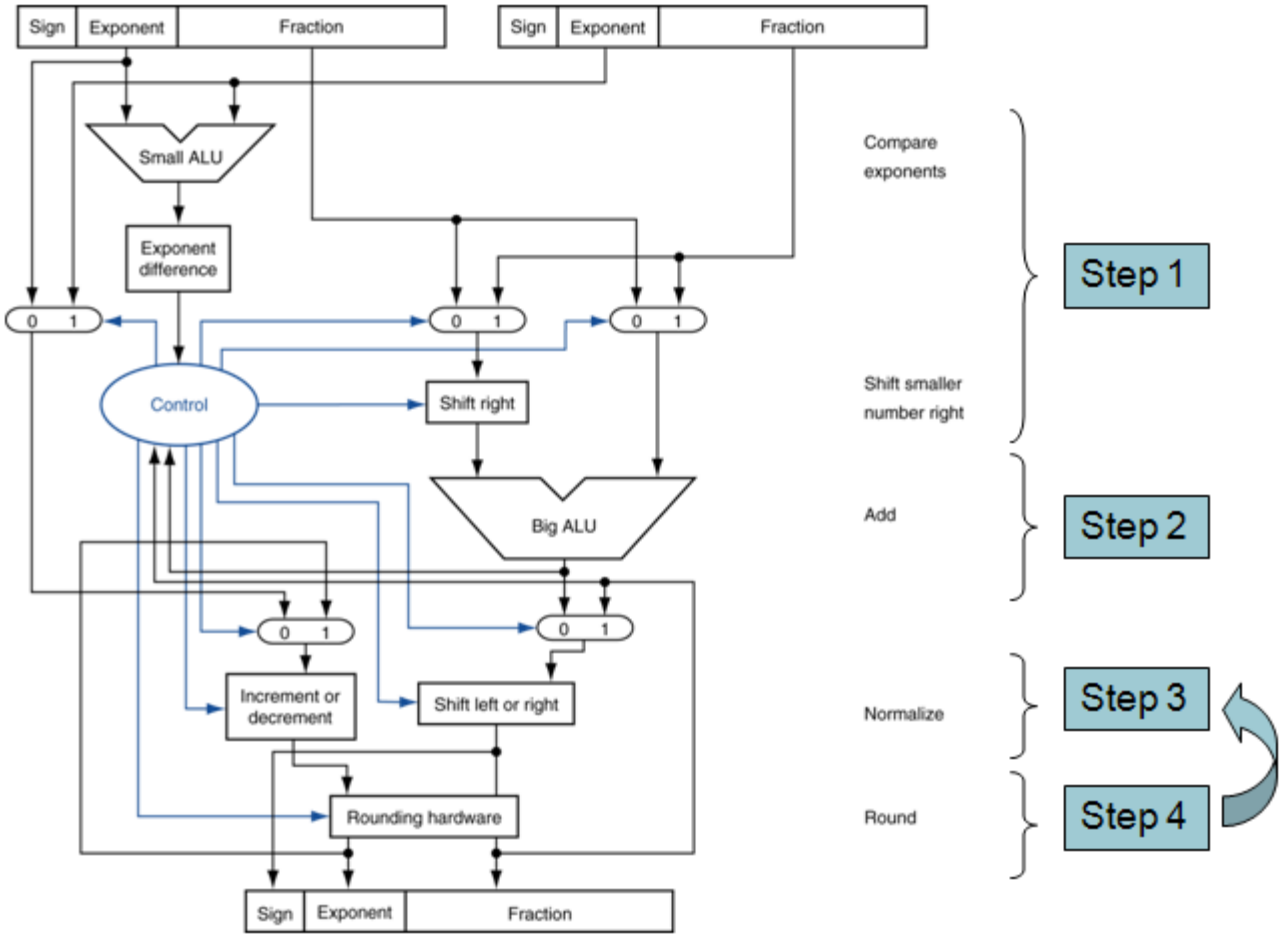
Porovnání dvou čísel ve FP

- Porovnání: je-li $A \geq B \Leftrightarrow A - B \geq 0$.
- Obrazy čísel A a B se odečtou jako čísla v přímém kódu a v pevné řádová čárce.
- To je výhodou zvoleného zobrazení čísel.

Algoritmus sčítání v pohyblivé řádové čárce

- Odečteme exponenty.
- Mantisu čísla s menším exponentem posuneme doprava o počet bitů, který je roven rozdílu exponentů.
- Sečteme mantisy obou čísel.
- Určíme počet nul mezi řádovou čárkou a první platnou číslicí součtu mantis.
- Posuneme součet doleva o tolik míst, kolik nul bylo nalezeno za řádovou čárkou.
- Zmenšíme původní exponent o počet nalezených nul.
- Zaokrouhlíme.

HW FP sčítačky



Násobení čísel v pohyblivé řádové čárce

- Exponenty sečteme.
- Mantisy vynásobíme.
- Normalizujeme.
- Zaokrouhlíme.

- HW FP násobičky je srovnatelně složitý, jako FP sčítačky. Jen má namísto sčítačky násobičku.