# Floating Point

## Jak je reprezentovaný a proč někdy nefunguje

Augustin Žídek

augustin<at>zidek<dot>eu

2. června 2013

# Historie

- Leonardo Torres y Quevedo 1914 – Analytical Engine s floating point

# Historie

- Leonardo Torres y Quevedo 1914 – Analytical Engine s floating point
- Floating point vs fixed point (von Neumann)

# Historie

- Leonardo Torres y Quevedo 1914 – Analytical Engine s floating point
- Floating point vs fixed point (von Neumann)
- Různé implementace až do 1985 – **IEEE 754-1985**

# Post IEEE-754

- **Raketa Ariane 5** (1996), 10 let vývoje, 7 miliard dolarů

# Post IEEE-754

- **Raketa Ariane 5** (1996), 10 let vývoje, 7 miliard dolarů
- Konvertovali 64-bitový `float` na 16-bitový `int`
- “Neočekávaný overflow”

# Post IEEE-754

- **Raketa Ariane 5** (1996), 10 let vývoje, 7 miliard dolarů
- Konvertovali 64-bitový `float` na 16-bitový `int`
- “Neočekávaný overflow”

- **Burza ve Vancouveru** (1983)

# Post IEEE-754

- **Raketa Ariane 5** (1996), 10 let vývoje, 7 miliard dolarů
- Konvertovali 64-bitový `float` na 16-bitový `int`
- “Neočekávaný overflow”

- **Burza ve Vancouveru** (1983)
- Index podhodnocený o 44 %
- Zaokrouhlovací chyba se sbírala 22 měsíců

# Post IEEE-754

- **Raketa Ariane 5** (1996), 10 let vývoje, 7 miliard dolarů
- Konvertovali 64-bitový `float` na 16-bitový `int`
- “Neočekávaný overflow”

- **Burza ve Vancouveru** (1983)
- Index podhodnocený o 44 %
- Zaokrouhlovací chyba se sbírala 22 měsíců

- **Střely Patriot** (1991)

# Post IEEE-754

- **Raketa Ariane 5** (1996), 10 let vývoje, 7 miliard dolarů
- Konvertovali 64-bitový `float` na 16-bitový `int`
- “Neočekávaný overflow”

- **Burza ve Vancouveru** (1983)
- Index podhodnocený o 44 %
- Zaokrouhlovací chyba se sbírala 22 měsíců

- **Střely Patriot** (1991)
- 28 mrtvých, nepřesnost 1/20 s kvůli 24-bitovým `float`

# Post IEEE-754

# Reprezentace – vědecká notace

$$4.2 \times 10^8$$

$4.2$ – mantisa
$8$ – exponent

# Reprezentace floatu

$$+1.11001_2 \times 2^{1011} \ (= 3648)$$

# Reprezentace floatu

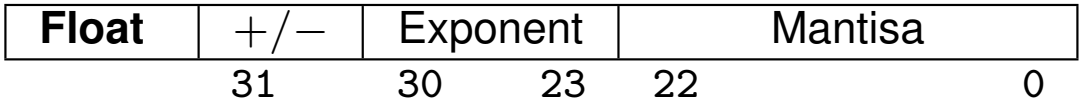
$$+1.11001_2 \times 2^{1011} \ (= 3648)$$

| **Float** | $+/-$ | Exponent | | Mantisa | |
|---|---|---|---|---|---|
| | 31 | 30 | 23 | 22 | 0 |

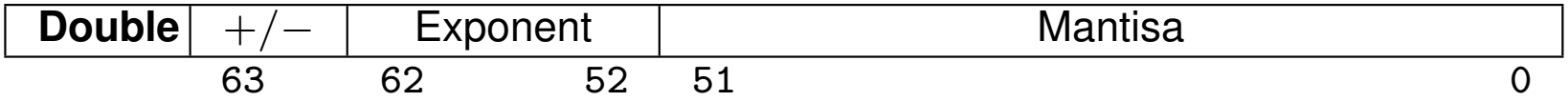
32 (1+8+23) bitů

# Reprezentace floatu

$$+1.11001_2 \times 2^{1011} \ (= 3648)$$




32 (1+8+23) bitů




64 (1+11+52) bitů

# Reprezentace floatu – mantisa

- Mantisa normalizovaná: $1 \leq \texttt{mantisa} < 2$

# Reprezentace floatu – mantisa

- Mantisa normalizovaná: $1 \leq \texttt{mantisa} < 2$
- Vždy $\texttt{1}$ na začátku – není třeba ukládat (hidden bit)

# Reprezentace floatu – mantisa

- Mantisa normalizovaná: $1 \leq \texttt{mantisa} < 2$
- Vždy `1` na začátku – není třeba ukládat (hidden bit)
- `mantisa = 01000...` Jaké číslo je reprezentováno?

# Reprezentace floatu – mantisa

- Mantisa normalizovaná: $1 \leq \texttt{mantisa} < 2$
- Vždy `1` na začátku – není třeba ukládat (hidden bit)
- `mantisa = 01000...` Jaké číslo je reprezentováno?
  1 + 1/4 = 1.25

# Reprezentace floatu – exponent

| Exp (bin) | Exp (dec) | Hodnota |
|---|---|---|
| 00000000 | 0 | $0$ pokud `mantisa = 0` |
| 00000001 | 1 | $2^{-126}$ |
| ... | ... | ... |
| 01111111 | 127 | $2^{0}$ |
| 10000000 | 128 | $2^{1}$ |
| ... | ... | ... |
| 11111110 | 254 | $2^{127}$ |
| 11111111 | 255 | `inf` pokud `mantisa = 0`, jinak `NaN` |

# Reprezentace floatu – exponent

| Exp (bin) | Exp (dec) | Hodnota |
|---|---|---|
| 00000000 | 0 | $0$ pokud `mantisa = 0` |
| 00000001 | 1 | $2^{-126}$ |
| ... | ... | ... |
| 01111111 | 127 | $2^{0}$ |
| 10000000 | 128 | $2^{1}$ |
| ... | ... | ... |
| 11111110 | 254 | $2^{127}$ |
| 11111111 | 255 | `inf` pokud `mantisa = 0`, jinak `NaN` |

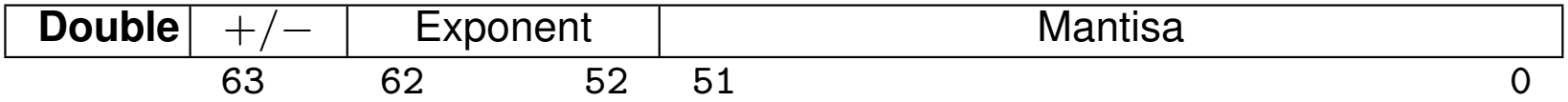
`double`: $2^{-1022} - 2^{1023}$

# Reprezentace floatu – denormalizovaná mantisa

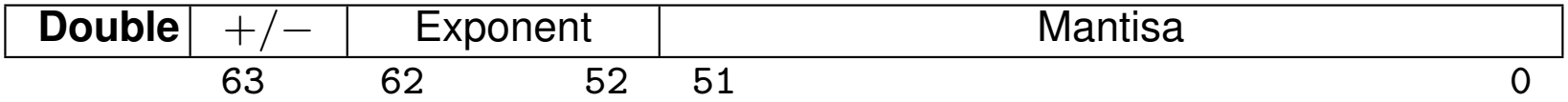
Pokud `exponent = 0`, stává se mantisa denormalizovaná

`0_00000000_10000000_00000000_00000000` $= 0.5 \times 2^{-126}$

# Cvičení

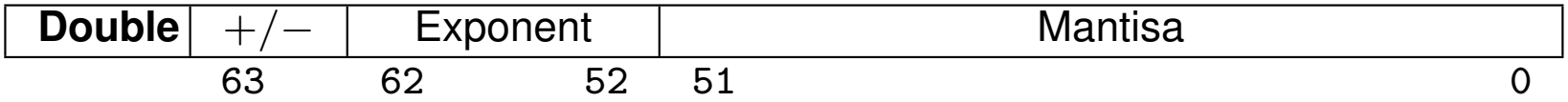


# Cvičení



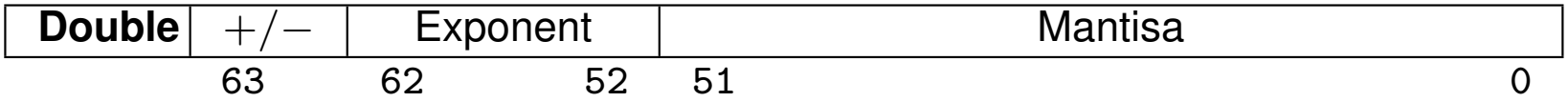

0x8000_0000_0000_0000

# Cvičení



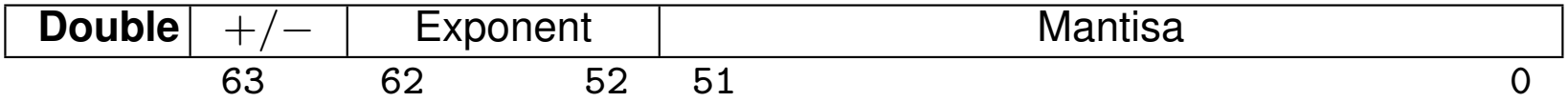

`0x8000_0000_0000_0000 = -0.0`

# Cvičení



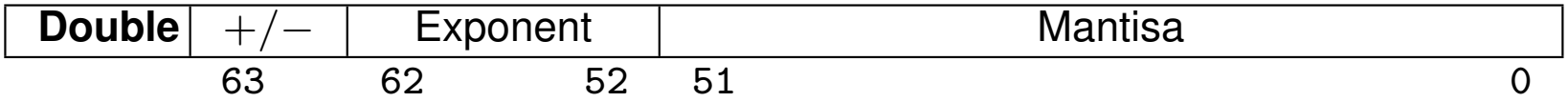

```
0x8000_0000_0000_0000  = -0.0
0x4008_0000_0000_0000
```

# Cvičení



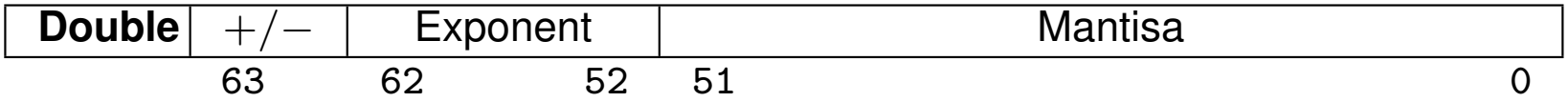

```
0x8000_0000_0000_0000  = -0.0
0x4008_0000_0000_0000  = 1.5 * 2^1
```

# Cvičení



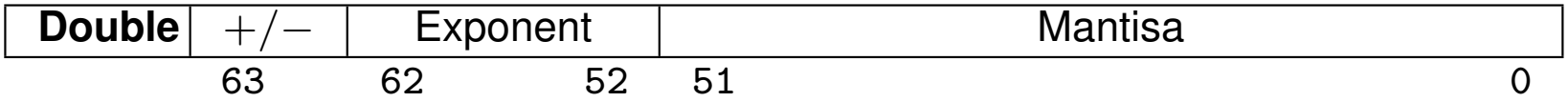

```
0x8000_0000_0000_0000  = -0.0
0x4008_0000_0000_0000  = 1.5 * 2^1
0x7FF0_0000_0000_0000
```

# Cvičení



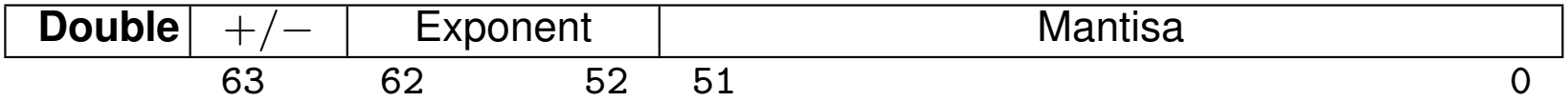

```
0x8000_0000_0000_0000  = -0.0
0x4008_0000_0000_0000  = 1.5 * 2^1
0x7FF0_0000_0000_0000  = +Infinity
```

# Cvičení




```
0x8000_0000_0000_0000  = -0.0
0x4008_0000_0000_0000  = 1.5 * 2^1
0x7FF0_0000_0000_0000  = +Infinity
0x7FF0_0000_000A_0000
```

# Cvičení




```
0x8000_0000_0000_0000  = -0.0
0x4008_0000_0000_0000  = 1.5 * 2^1
0x7FF0_0000_0000_0000  = +Infinity
0x7FF0_0000_000A_0000  = NaN
```

# Největší, nejmenší

- Největší `float`: $1.111...111 \times 2^{127} \doteq 3 \times 10^{38}$

# Největší, nejmenší

- Největší `float`: $1.111...111 \times 2^{127} \doteq 3 \times 10^{38}$
- Největší `double`: $1.111...111 \times 2^{1023} \doteq 2 \times 10^{308}$

# Největší, nejmenší

- Největší `float`: $1.111...111 \times 2^{127} \doteq 3 \times 10^{38}$
- Největší `double`: $1.111...111 \times 2^{1023} \doteq 2 \times 10^{308}$
- Nejmenší záporné analogicky

# Největší, nejmenší

- Největší `float`: $1.111...111 \times 2^{127} \doteq 3 \times 10^{38}$
- Největší `double`: $1.111...111 \times 2^{1023} \doteq 2 \times 10^{308}$
- Nejmenší záporné analogicky
- Co nejmenší pozitivní? Pozor na denormalizované!

# Největší, nejmenší

- Největší `float`: $1.111...111 \times 2^{127} \doteq 3 \times 10^{38}$
- Největší `double`: $1.111...111 \times 2^{1023} \doteq 2 \times 10^{308}$
- Nejmenší záporné analogicky
- Co nejmenší pozitivní? Pozor na denormalizované!
- Nejmenší `float`: $0.000...1 \times 2^{-126} \doteq 1.4 \times 10^{-45}$

# Největší, nejmenší

- Největší `float`: $1.111...111 \times 2^{127} \doteq 3 \times 10^{38}$
- Největší `double`: $1.111...111 \times 2^{1023} \doteq 2 \times 10^{308}$
- Nejmenší záporné analogicky
- Co nejmenší pozitivní? Pozor na denormalizované!
- Nejmenší `float`: $0.000...1 \times 2^{-126} \doteq 1.4 \times 10^{-45}$
- Nejmenší `double`: $0.000...1 \times 2^{-1022} \doteq 5 \times 10^{-324}$

# Největší, nejmenší

- Největší `float`: $1.111...111 \times 2^{127} \doteq 3 \times 10^{38}$
- Největší `double`: $1.111...111 \times 2^{1023} \doteq 2 \times 10^{308}$
- Nejmenší záporné analogicky
- Co nejmenší pozitivní? Pozor na denormalizované!
- Nejmenší `float`: $0.000...1 \times 2^{-126} \doteq 1.4 \times 10^{-45}$
- Nejmenší `double`: $0.000...1 \times 2^{-1022} \doteq 5 \times 10^{-324}$
- Viz proměnné `Float` a `Double`

# Machine epsilon

- **Definice:** Rozdíl mezi `1.0` a nejmenším reprezentovatelným číslem, které je větší než jedna.

# Machine epsilon

- **Definice:** Rozdíl mezi `1.0` a nejmenším reprezentovatelným číslem, které je větší než jedna.
- Minimální rozdíl mezi čísly aby byla považována za nestejná.

# Machine epsilon

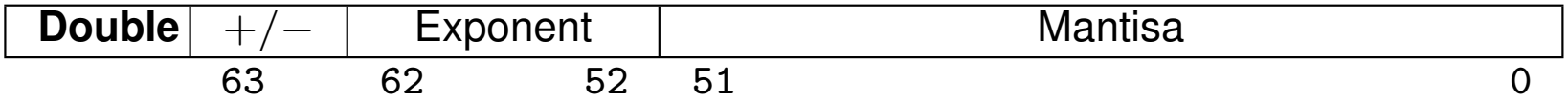
- **Definice:** Rozdíl mezi `1.0` a nejmenším reprezentovatelným číslem, které je větší než jedna.
- Minimální rozdíl mezi čísly aby byla považována za nestejná.
- `Float:` 1.19209290E-7
- `Double:` 2.2204460492503131E-16

# Machine epsilon

- **Definice:** Rozdíl mezi `1.0` a nejmenším reprezentovatelným číslem, které je větší než jedna.
- Minimální rozdíl mezi čísly aby byla považována za nestejná.
- `Float:` 1.19209290E-7
- `Double:` 2.2204460492503131E-16
- Jednoduché vypočítat

# Porovnávání

`float` lze porovnávat bitově jako `int`



# První úskalí

Jak je reprezentováno `0.1`?

# První úskalí

Jak je reprezentováno `0.1`?

`1_11111101110_0110011001100110011001100...`

# Aritmetika – násobení a dělení

$$(a \times 2^n) \cdot (b \times 2^m) = a \cdot b \times 2^{n+m}$$
$$(a \times 2^n) / (b \times 2^m) = a / b \times 2^{n-m}$$

+ normalizace

# Aritmetika – sčítání a odčítání

# Aritmetika – sčítání a odčítání

1. Denormalizuj na větší číslo
   $1 \times 2^1 + 1.9 \times 2^3 = 0.125 \times 2^3 + 1.9 \times 2^3$

# Aritmetika – sčítání a odčítání

1. Denormalizuj na větší číslo
   $1 \times 2^1 + 1.9 \times 2^3 = 0.125 \times 2^3 + 1.9 \times 2^3$
2. Sečti/odečti
   $0.125 \times 2^3 + 1.9 \times 2^3 = 2.025 \times 2^3$

# Aritmetika – sčítání a odčítání

1. Denormalizuj na větší číslo
   $1 \times 2^1 + 1.9 \times 2^3 = 0.125 \times 2^3 + 1.9 \times 2^3$
2. Sečti/odečti
   $0.125 \times 2^3 + 1.9 \times 2^3 = 2.025 \times 2^3$
3. Normalizuj
   $2.025 \times 2^3 = 1.0125 \times 2^4$

# Je sčítání asociativní?

```
(1.0 + 1.0E40) - 1.0E40 = 1.0 + (1.0E40 - 1.0E40)
```

# Je sčítání asociativní?

```
(1.0 + 1.0E40) - 1.0E40 = 1.0 + (1.0E40 - 1.0E40)
```

Hmm, někdy není...

# Sčítání řad

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n} \neq \frac{1}{n} + \ldots + \frac{1}{4} + \frac{1}{3} + \frac{1}{2} + \frac{1}{1}$$

# IEEE-754 zaokrouhlování

# IEEE-754 zaokrouhlování

- Zaokrouhlování s preferencí sudé číslice (unbiased rounding)

# IEEE-754 zaokrouhlování

- Zaokrouhlování s preferencí sudé číslice (unbiased rounding)
- $1.5 \doteq 2.0$, $2.5 \doteq 2.0$, $3.5 \doteq 4.0$

# IEEE-754 zaokrouhlování

- Zaokrouhlování s preferencí sudé číslice (unbiased rounding)
- $1.5 \doteq 2.0$, $2.5 \doteq 2.0$, $3.5 \doteq 4.0$
- Výhoda: Statisticky stejně často $.5$ zaokrouhleno nahoru a dolu

# IEEE-754 zaokrouhlování

- Zaokrouhlování s preferencí sudé číslice (unbiased rounding)
- $1.5 \doteq 2.0$, $2.5 \doteq 2.0$, $3.5 \doteq 4.0$
- Výhoda: Statisticky stejně často $.5$ zaokrouhleno nahoru a dolu
- Ukázky `RoundingProblem` a `RoundingProblem2`

# Co s tím

# Co s tím

- `print(random())` – alespoň intelektuálně upřímné

# Co s tím

- `print(random())` – alespoň intelektuálně upřímné
- Použití knihoven (Apfloat, JScience), nástrojů (Matlab, Octave)

# Co s tím

- `print(random())` – alespoň intelektuálně upřímné
- Použití knihoven (Apfloat, JScience), nástrojů (Matlab, Octave)
- V Javě `BigDecimal`

# BigDecimal

- Libovolná přesnost, nástroje pro manipulaci

# BigDecimal

- Libovolná přesnost, nástroje pro manipulaci
- Naprostá kontrola zaokrouhlování – ideální pro finanční matematiku

# BigDecimal

- Libovolná přesnost, nástroje pro manipulaci
- Naprostá kontrola zaokrouhlování – ideální pro finanční matematiku
- `docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html`

# Dotazy, řešení problémů?

augustin<at>zidek<dot>eu

Děkuji za pozornost