Reprezentace čísel v počítači

Tomáš Faltejsek, Luboš Zápotočný, Michal Havelka

2022

Obsah

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1
- **1 byte** = 8 bitů

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1
- **1 byte** = 8 bitů
  - Nejmenší adresovatelná jednotka v paměti počítače

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1
- **1 byte** = 8 bitů
  - Nejmenší adresovatelná jednotka v paměti počítače
  - Nelze tedy od paměti požadovat například 11. bit v pořadí

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1
- **1 byte** = 8 bitů
  - Nejmenší adresovatelná jednotka v paměti počítače
  - Nelze tedy od paměti požadovat například 11. bit v pořadí
  - Musíme si nechat nahrát celý byte (bity 8-16) a z něho poté v programu vybrat 3. bit

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1
- **1 byte** = 8 bitů
  - Nejmenší adresovatelná jednotka v paměti počítače
  - Nelze tedy od paměti požadovat například 11. bit v pořadí
  - Musíme si nechat nahrát celý byte (bity 8-16) a z něho poté v programu vybrat 3. bit
- **Adresa do paměti**

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1
- **1 byte** = 8 bitů
  - Nejmenší adresovatelná jednotka v paměti počítače
  - Nelze tedy od paměti požadovat například 11. bit v pořadí
  - Musíme si nechat nahrát celý byte (bity 8-16) a z něho poté v programu vybrat 3. bit
- **Adresa do paměti**
  - Kladné celé číslo ($\mathbb{N}^+$)

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1
- **1 byte** = 8 bitů
  - Nejmenší adresovatelná jednotka v paměti počítače
  - Nelze tedy od paměti požadovat například 11. bit v pořadí
  - Musíme si nechat nahrát celý byte (bity 8-16) a z něho poté v programu vybrat 3. bit
- **Adresa do paměti**
  - Kladné celé číslo ($\mathbb{N}^+$)
  - Index buňky v paměti

Bit vs. Byte

Adresa	0	1	2	3
Data	137	0**b**10001001	0**x**89	'a'

- **1 bit** je základní a nejmenší jednotkou informace v počítači
  - Nabývá pouze hodnot 0 či 1
- **1 byte** = 8 bitů
  - Nejmenší adresovatelná jednotka v paměti počítače
  - Nelze tedy od paměti požadovat například 11. bit v pořadí
  - Musíme si nechat nahrát celý byte (bity 8-16) a z něho poté v programu vybrat 3. bit
- **Adresa do paměti**
  - Kladné celé číslo ($\mathbb{N}^+$)
  - Index buňky v paměti
  - Operační systém dává programu **virtuální adresy** místo fyzických

Binární soustava

Převod binárního čísla 10001001 do desítkové soustavy

Binární soustava

Převod binárního čísla 10001001 do desítkové soustavy

1	0	0	0	1	0	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Binární soustava

Převod binárního čísla 10001001 do desítkové soustavy

1	0	0	0	1	0	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

$$\begin{aligned} 10001001 &= 1 * 128 \\ &+ 0 * 64 + 0 * 32 + 0 * 16 \\ &+ 1 * 8 \\ &+ 0 * 4 + 0 * 2 \\ &+ 1 * 1 \\ &= 137 \end{aligned} \tag{1}$$

Hexadecimální soustava

Hexadecimální kódování číslic

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111

8	9	A (10)	B (11)	C (12)	D (13)	E (14)	F (15)
1000	1001	1010	1011	1100	1101	1110	1111

Příklad čísla zapsaného v hexadecimální soustavě: 5FE9

Převod z hexadecimální soustavy do binární

Převod binárního čísla 5FE9 do binární soustavy

Převod z hexadecimální soustavy do binární

Převod binárního čísla 5FE9 do binární soustavy

5	F	E	9
0101	1111	1110	1001

Převod z hexadecimální soustavy do binární

Převod binárního čísla 5FE9 do binární soustavy

5	F	E	9
0101	1111	1110	1001

0x5FE9 = 0b0101111111101001

Převod z hexadecimální soustavy do binární

Převod binárního čísla 5FE9 do binární soustavy

5	F	E	9
0101	1111	1110	1001

0x5FE9 = 0b0101111111101001

Což lze také (ekvivalentně) vyjádřit pouze použitím 15 bitů místo 16 vynecháním první nuly, která hodnotu binárního čísla nezmění

0x5FE9 = 0b101111111101001

Reprezentace čísel v počítači

Celočíselné datové typy

- Reprezentace diskrétních jevů
- *"Přesné"* výpočty

Reprezentace reálných čísel

- Reprezentace "spojitých" jevů
- Výpočty jsou *nepřesné* - obsahují zaokrouhlovací chybu

Vyjádření čísla x v soustavě z:

$$(x)_z = \pm \sum_{i=0}^{n} b_i z^i, \;\; b_i \in < 0, z-1 >$$

Vyjádření čísla v soustavách

Vyjádření

$$(x)_z = \pm \sum_{i=0}^{n} b_i z^i, \;\; b_i \in < 0, z-1 >$$

- Decimální (z = 10): $(x)_{10} = 200$

Vyjádření čísla v soustavách

Vyjádření

$$(x)_z = \pm \sum_{i=0}^{n} b_i z^i, \;\; b_i \in < 0, z-1 >$$

- Decimální (z = 10): $(x)_{10} = 200$
- Binární (z = 2): $(x)_2 = 11001000$

Vyjádření čísla v soustavách

Vyjádření

$$(x)_z = \pm \sum_{i=0}^{n} b_i z^i, \;\; b_i \in < 0, z-1 >$$

- Decimální (z = 10): $(x)_{10} = 200$
- Binární (z = 2): $(x)_2 = 11001000$
- Hexadecimální (z = 16): $(x)_{16} = C8$

Vyjádření čísla v soustavách

Vyjádření

$$(x)_z = \pm \sum_{i=0}^{n} b_i z^i, \;\; b_i \in < 0, z-1 >$$

- Decimální (z = 10): $(x)_{10} = 200$
- Binární (z = 2): $(x)_2 = 11001000$
- Hexadecimální (z = 16): $(x)_{16} = C8$

Otázka

Proč jsou součástí hexadecimální soustavy charaktery?

Celočíselné datové typy

Jaké znáte celočíselné datové typy?

Celočíselné datové typy

Jaké znáte celočíselné datové typy?

```
int main() {
    char c;                 // 1 byte
    unsigned char uc;       // 1 byte
    short s;                // 2 bytes
    unsigned short us;      // 2 bytes
    int i;                  // 4 bytes
    unsigned int ui;        // 4 bytes
    long l;                 // 8 bytes
    unsigned long ul;       // 8 bytes
    long long ll;           // 8 bytes
    unsigned long long ull; // 8 bytes

    return 0;
}
```

Celočíselné datové typy

Reprezentaci **celočíselných** datových lze rozdělit dle:

- **Přesnosti**
  - short - nižší přesnost
  - long - vyšší přesnost
- **Znaménka** (sign)
  - unsigned - bez znaménka ($\mathbb{Z}^+$)
  - signed - se znaménkem ($\mathbb{Z}$)

Rozsahy celočíselných datových typů

Typ	Paměť	Rozsah	Znaménko	Formátovací řetězec
short *(int)*	2 byte	$< -32,768; 32,767 >$	ano	%hd
unsigned short *(int)*	2 byte	$< 0; 65,535 >$	ne	%hu
int	4 byte	$< -2,147,483,648; 2,147,483,647 >$	ano	%d
unsigned int	4 byte	$< 0; 4,294,967,295 >$	ne	%u
long int	$\geq$ 4 byte	$< -2,147,483,648; 2,147,483,647 >$	ano	%ld
unsigned long int	$\geq$ 4 byte	$< 0; 4,294,967,295 >$	ne	%lu
long long *(int)*	$\geq$ 8 byte	$< -(2^{63}); (2^{63}) - 1 >$	ano	%lld
unsigned long long *(int)*	$\geq$ 8 byte	$< 0; \approx 2^{64} - 1 >$	ne	%llu

**Na 32-bitové architektuře kompilováno skrze gcc*

Rozsahy celočíselných datových typů

Typ	Paměť	Rozsah	Znaménko	Formátovací řetězec
short *(int)*	2 byte	$< -32,768; 32,767 >$	ano	%hd
unsigned short *(int)*	2 byte	$< 0; 65,535 >$	ne	%hu
int	4 byte	$< -2,147,483,648; 2,147,483,647 >$	ano	%d
unsigned int	4 byte	$< 0; 4,294,967,295 >$	ne	%u
long int	$\geq$ 4 byte	$< -2,147,483,648; 2,147,483,647 >$	ano	%ld
unsigned long int	$\geq$ 4 byte	$< 0; 4,294,967,295 >$	ne	%lu
long long *(int)*	$\geq$ 8 byte	$< -(2^{63}); (2^{63}) - 1 >$	ano	%lld
unsigned long long *(int)*	$\geq$ 8 byte	$< 0; \approx 2^{64} - 1 >$	ne	%llu

**Na 32-bitové architektuře kompilováno skrze gcc*

Otázka

Co se stane při výpisu unsigned int pomocí formátovacího řetězce %d ?

Datové typy s plovoucí desetinnou čárkou

Jaké znáte datové typy s plovoucí desetinnou čárkou?

Datové typy s plovoucí desetinnou čárkou

Jaké znáte datové typy s plovoucí desetinnou čárkou?

```
int main() {
    float f;          // 4  bytes
    double d;         // 8  bytes
    long double ld; // 16 bytes

    return 0;
}
```

Přímý kód

Číslo je v počítači uloženo v binárním tvaru

$$(x)_2 = \pm \sum_{i=0}^{n} b_i 2^i, \;\; b_i \in < 0, 1 >$$

- **bit na první pozici (MSB) vymezen pro znaménko**
  - **0** $= +$ (*kladné číslo*), **1** $= -$ (*záporné číslo*)
- Problémy:
  1. Dvojí reprezentace nuly: $P(0)_{10} = 00000000, P(-0)_{10} = 10000000$
  2. Není zachována (ne)rovnost:
     $P(100)_{10} = (01100100) < P(-100)_{10} = (11100100)$

Otázka

Jakého rozsahu nabývá 8-bitové číslo reprezentované přímým kódem?

Přímý kód

Číslo je v počítači uloženo v binárním tvaru

$$(x)_2 = \pm \sum_{i=0}^{n} b_i 2^i, \;\; b_i \in < 0, 1 >$$

- **bit na první pozici (MSB) vymezen pro znaménko**
  - **0** $= +$ (*kladné číslo*), **1** $= -$ (*záporné číslo*)
- Problémy:
  1. Dvojí reprezentace nuly: $P(0)_{10} = 00000000, P(-0)_{10} = 10000000$
  2. Není zachována (ne)rovnost:
     $P(100)_{10} = (01100100) < P(-100)_{10} = (11100100)$

Otázka

Jakého rozsahu nabývá 8-bitové číslo reprezentované přímým kódem?

$$x \in < -2^m - 1; 2^m - 1 >, \quad m = n - 1, \quad n = 8$$

Inverzní kód

Rozsah

$$x \in < -2^m - 1; 2^m - 1 >, \quad m = n - 1$$

- Záporné číslo je **negací** (jedničkovým doplňkem) kladného čísla
- **Výhody**:
  1. $I(100)_{10} = (01100100)_2 > I(-100)_{10} = \mathbf{!}(01100100)_2 = (10011011)_2$
     - ★ Nyní platí (ne)rovnost, odpadá problém se zachováním relace
- **Problémy**:
  1. Dvojí reprezentace nuly:
     $I(0)_{10} = 00000000, I(-0)_{10} = \mathbf{!}00000000 = 11111111$

Doplňkový kód

- Připočtením **1** k *jedničkovému doplňku* získáváme **dvojkový doplněk**
- $D(-100)_{10} = !(01100100)_2 + 1 = (10011100)_2$
- **Výhody**:
  1. Jediná reprezentace 0:
     $D(0)_{10} = 00000000 = D(-0) = 11111111 + 1 = 00000000$
  2. Zachovává relace:
     $D(100)_{10} = 01100100 > D(-105)_{10} = I(-105)_{10} + 1 = (10011100)_2$
- **Problémy**:
  1. Nesymetrický interval (nelze vyjádřit absolutní hodnotu nejzápornějšího čísla apod.)

Reálné datové typy

Pevná řádová čárka

- Reprezentace složením: **n** bitů pro *celou část*, **m** bitů pro desetinnou část a **1** bit pro znaménko (sign)

Obecný zápis

$$(x)_2 = (x_c + x_d), \quad x_c = \sum_{i=0}^{n} b_i 2^i, \quad \sum_{i=-1}^{m} b_i 2^i$$

$$\begin{aligned} 10.01_2 &= 1 * 2^1 + 0 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} \\ &= 1 * 2 + 0 * 1 + 0 * \frac{1}{2} + 1 * \frac{1}{4} \\ &= 2 + 0.25 \\ &= 2.25_{10} \end{aligned} \tag{2}$$

Plovoucí řádová čárka

Semilogaritmický tvar čísla

$$x = m \cdot z^e$$

Pokud číslo splňuje *normalizační podmínku*, nazýváme ho **normalizované**:

$$1 \leq m < z$$

Tedy:

- Mantisa vždy začíná binární číslicí **1**
- Mantisa leží v intervalu $< 1, z)$

	exponent e						mantisa m						
$\pm$	2^{n-1}	$\ldots$	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	$\ldots$	2^{-m}

Přenost binárně uloženého reálného čísla

Čím více bitů má mantisa, tím vyšši přesnost čísla

Pro uložení exponentů stačí menší počet bitů

Semilogaritmický tvar: dekadické a binární číslo

- **Dekadické číslo**:
  $-123,000,000,000,000 = -1.23 \times 10^{14}$
  $0.000\ 000\ 000\ 000\ 000\ 123 = 1.23 \times 10^{-16}$
- **Binární číslo**:
  $110\ 1100\ 0000\ 0000 = 1.1011 \times 2^{14}$

Přesnost desetinných čísel v počítači

```
#include <stdio.h>

// Co se vytiskne na stdout?

int main() {
        float a = 0.1;

        printf("%f", a);


    return 0;
}
```

Přesnost desetinných čísel v počítači

```
#include <stdio.h>

// Co se vytiskne na stdout?

int main() {
        float a = 0.1;

        printf("%f", a);


    return 0;
}
```

Odpověď

0.100000

Číslo 0.1 v binární soustavě

- Konečné číslo v dekadické soustavě $\rightarrow$ nekonečné číslo v binární soustavě

$$0.1_{10} = 0.00011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011$$
$$0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011$$
$$0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011$$
$$0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011$$
$$0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011$$
$$0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011$$
$$0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011$$
$$\ldots$$
$$0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011_2$$

$\rightarrow$ **nekonečná** sekvence bitů

Nepřesné zobrazení realných čísel

$$\frac{1}{3} \approx 0.0101\ 0101\ 0101\ \ldots\ 01_2$$

$$\frac{1}{5} \approx 0.0011\ 0011\ 0011\ \ldots\ 0011_2$$

$$\frac{1}{10} \approx 00011\ 0011\ 0011\ \ldots\ 0011_2$$

Omezení

Přesně lze vyjádřit pouze čísla ve tvaru $\frac{x}{2^k}$
Všechna ostatní čísla se ukládají jako **nepřesná**

Uložení čísla dle normy IEEE754

- Jednoduchá přesnost (32 bitů) v jazyce C: **float**
- Dvojnásobná přesnost (64 bitů) v jazyce C: **double**

Poznámka

Ve verzi *IEEE 754-2008* představena *plovoucí přesnost*

- 16 bitová přesnost (využíváno při grafice)
- 128 a 256 bitová přesnost (vědecké výpočty)
- Definuje rozložení bitů mezi mantisou a exponentem

Aritmetika čísel s desetinou čárkou

```
#include <stdio.h>

// Co se vytiskne na stdout?

int main() {
    if (0.1 + 0.2 == 0.3) {
        printf("Rovna se \n");
    } else {
        printf("Nerovna se \n");
    }

    return 0;
}
```

Odpověď

Nerovna se

Aritmetika čísel s desetinou čárkou - epsilon

```
#include <stdio.h>
#include <float.h>
#include <stdlib.h>

// Co se vytiskne na stdout?

int main(void) {
        if (abs(0.1 + 0.2 - 0.3) < DBL_EPSILON) {
            printf("Je mensi \n");
        } else {
            printf("Neni mensi \n");
        }

        return 0;
}
```

Modulární aritmetika

Operace modulo - zbytek po dělení

modulo (%) je binární operátor, který dává zbytek po celočíselném dělení

Modulární aritmetika

Operace modulo - zbytek po dělení

modulo (%) je binární operátor, který dává zbytek po celočíselném dělení

Příklady

$5 \% 3 = ?$
$9 \% 4 = ?$
$1024 \% 2 = ?$

Modulární aritmetika

Operace modulo - zbytek po dělení

modulo (%) je binární operátor, který dává zbytek po celočíselném dělení

Příklady

$5 \% 3 = ?$
$9 \% 4 = ?$
$1024 \% 2 = ?$

Výsledky

$5 \% 3 = 2$
$9 \% 4 = 1$
$1024 \% 2 = 0$

Posun bitů

Posun doleva

Posun bitů o jednu pozici **vlevo** je stejná operace jako **násobení 2**

```
#include <stdio.h>

int main() {
    char a = 0b00000100;        // 4
    printf("%d\n", a);
    a = a << 1;                 // 4 * 2 ==> 8
    printf("%d\n", a);
    return 0;
}
```

Posun doprava

Posun bitů o jednu pozici **vpravo** je stejná operace jako **dělení 2 a následné zaokrouhlení dolů**

```
#include <stdio.h>

int main() {
    char a = 0b00000101;        // 5
    printf("%d\n", a);
    a = a >> 1;                 // floor(5 / 2) ==> 2
    printf("%d\n", a);
    return 0;
}
```