# Hardwarové uložení číselných dat

**Doc. Dr. Vladimír Homola, Ph.D.**

Tento text upřesňuje způsoby uložení *numerických* dat v paměti výpočetních systémů, především v kombinaci s procesory řady Intel a jejich klonů. Zabývá se pouze takovými architekturami, které pracují s organizací paměti po bytech - přesněji takovými, kde nejmenší adresovatelnou jednotkou paměti je jeden byte. Nepředpokládá žádné specielní znalosti čtenáře.

## Bit a byte

Základní pojmy níže uvedené jsou částečně vybrány a volně citovány z normy ISO 2382 (poměrně dobře jí odpovídá dřívější ČSN 36 9001).

***Bit*** (z anglického binary digit - dvojková cifra, zkratka **b**): cokoliv, co nabývá pouze dvou hodnot a "počítač" je schopen rozeznat, kterých. Příklad: zrnko materiálu je nebo není zmagnetované; tranzistor proud vede nebo nevede; žárovka svítí nebo nesvítí. Protože tyto dvě hodnoty jsou většinou protikladné, bez ohledu na fyzikální podstatu se označují jako 0 a 1, NE a ANO apod.

***Byte*** (kdysi překládáno jako *slabika*, dnes se "bajt" většinou vůbec nepřekládá, zkratka **B**): posloupnost osmi bitů = dvojkových cifer. Jsou-li bity vyjádřeny jako 0 a 1, pak je jeden byte uspořádanou osmicí nul a jedniček a tvoří tedy zápis osmiciferného čísla ve dvojkové soustavě.. Těchto (různých) osmic je 256, počínaje např. 00000000, pak 00000001, ..., až po 11111110 a 11111111. Dvojková čísla uvedená shora (ve dvojkové soustavě) odpovídají následujícím číslům (v desítkové soustavě): 0, 1, ..., 254, 255. Graficky je možno jeden byte znázornit např. takto:

**Byte:**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Obsah | **O** | **O** | **O** | **I** | **O** | **O** | **I** | **O** |
| bitu č. | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bity v rámci bytu mají svoji adresu (jsou "očíslované"), která odpovídá příslušné mocnině základu soustavy, tj. mocnině 2.

***Paměť***: Uspořádaná množina bytů přístupná výpočetnímu systému. Fyzická realizace je různá: uspořádání ve formě diskety, pevného disku, magnetické pásky, integrovaného obvodu apod. Bez ohledu na fyzickou realizaci jsou paměti nazírány jako linearizovaná (a tedy "očíslovatelná") posloupnost bytů. Očíslování bývá provedeno počínaje nulou, tato "pořadová" čísla se nazývají *adresy* jednotlivých bytů. Na paměť je tedy možno pohlížet např. takto:

**Paměť:**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Obsah | **OIIOIOIOO** | **OIIOIOIOI** | **OOIIIIIO** | **OIIOIOII** | **OIIIIOOIO** | **OIIIOOIO** | **IIIOOOOOI** | **...** |
| bytu s adresou: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... |

***Kilobyte***, ***Megabyte***, ***Gigabyte***, ***Terabyte*** a další: jednotky pro udávání velikosti paměti (tisíc, milion, miliarda, bilion bytů).

*Poznámka 1: V občanském životě je např. předpona kilo- spojena s hodnotou přesně 1000. Ve dvojkovém počítačovém prostředí se však lépe pracuje s hodnotou 1024, protože ta je rovna $2^{10}$, tedy dvojkově 1 a deset nul. Pro rozlišení se "občanské" předpony zapisují malými písmeny, kdežto "počítačové" velkými. "Občanským" se říká "malé kilo" atd, kdežto "počítačovým" se říká "velké kilo"; analogicky ostatním jednotkám. Je tedy **kB**, **mB**, **gB** a **tB** přesně tisíc, milion, miliarda a bilion bytů, kdežto **KB**, **MB**, **GB** a **TB** je 1 024, 1 048 576, 1 073 741 824 a 1 099 511 627 776 bytů, což je $2^{10}$, $2^{20}$, $2^{30}$ a $2^{40}$ bytů.*

*Poznámka 2: Autor si je vědom toho, že někdy kolem roku 2002 byla převzatá norma u nás označovaná jako ČSN-IEC-60027-2, která ponechává "malým" jednotkám jejich původní název a označení (tedy např. kilobyte = kB), kdežto pro "velké" jednotky zavádí názvy "kibibyte", mebibyte", "gibibyte" atd. a označení KiB, MiB, GiB atd. Autor však přísahá, že s tímto označováním se kromě Wikipedie a zmíněné normy v praktickém životě nesetkal.*

***Datový typ***: způsob chápání obsahu jednoho, dvou nebo více po sobě následujících bytů v paměti.

***Semilogaritmický zápis čísla*** je zápis číselné hodnoty ve tvaru součinu celého nebo necelého čísla a mocniny základu (např. 10) - známý např. z kalkulaček, tj. ve tvaru M x $z^E$, např. 1,2 x $10^3$ (=1200). Část M je nazývána ***mantisa***, část E je nazývána ***exponent***.

***Normovaný semilogaritmický zápis čísla***: Každou nenulovou hodnotu lze zapsat pomocí semilogaritmického zápisu tak, že celá část mantisy má jedinou, a to nenulovou cifru (stačí příslušně upravit exponent) - např. 123.456 x $10^{20}$ = 1.23456 x $10^{22}$. Takovému zápisu se říká *normovaný* a jeho analogie ve dvojkové soustavě se využívá při ukládání neceločíselných hodnot v paměti.

# Binární minimum

Binární (= dvojková) soustava je takový zápis číselných hodnot, při kterém je jako základ použita hodnota 2. Hexadecimální (= šestnáctková) soustava je takový zápis číselných hodnot, při kterém je jako základu použita hodnota 16. Minimální informace o těchto soustavách podejme na základě analogie s běžně používanou soustavou dekadickou (= desítkovou). Připomeňme jen trochu pozapomínanou skutečnost, že zápis čísla (v jakékoliv soustavě) slouží k vyjádření *počtu* nějakých jednotek resp. jejich zlomků.

## Desítková soustava

Dekadická (= desítková) soustava je takový zápis číselných hodnot, při kterém je jako základ použita hodnota 10. Zopakujme, proč je 348 rovno právě 348: je to proto, že tato hodnota obsahuje 3 stovky (stovka = $10^2$), 4 desítky (desítka = $10^1$) a 8 jednotek (jednotka = $10^0$). Zapsáno jinak:

| $10^2$ (= **100**) | | $10^1$ (= **10**) | | $10^0$ (= **1**) | | celkem |
|---|---|---|---|---|---|---|
| x **3** | + | x **4** | + | x **8** | = | **348** |

Nyní rozšiřme opakování o to, proč je 348,65 rovno právě 348,65:

| $10^2$ (= **100**) | | $10^1$ (= **10**) | | $10^0$ (= **1**) | | $10^{-1}$ (= **0,1**) | | $10^{-2}$ (= **0,01**) | | celkem |
|---|---|---|---|---|---|---|---|---|---|---|
| x **3** | + | x **4** | + | x **8** | + | x **6** | + | x **5** | = | **348.65** |

Podotkněme jen, že správně by všechna čísla v tomto odstavci zapisovaná měla být důsledně označena základem své soustavy; místo 348 by tedy mělo být lépe zapsáno $348_{10}$. Zatím však byla použita jen desítková soustava a tedy se "to nepletlo".

Poslední připomenutí: v zápise čísel může být právě tolik různých hodnot daného řádu, kolik je základ soustavy. To znamená např. žádnou stovku, jednu stovku, dvě stovky, ..., osm stovek nebo devět stovek. Deset stovek už ne, protože to by byla jedna tisícovka ("přechod přes řád"). Pro znázornění těchto deseti různých hodnot je zapotřebí deseti jakýchkoliv, ale různých symbolů. V našich končinách a našem věku bývá zvykem používat symboly 0, 1, 2, ..., 8 a 9.

## Dvojková soustava

Jde o číselnou soustavu se základem 2. Analogii s desítkovou soustavou začněme "od konce": v zápise čísel může být právě tolik různých hodnot daného řádu, kolik je základ soustavy (tj. dvě). To znamená např. žádnou dvojku nebo jednu dvojku; dvě dvojky už ne, protože to by byl "přechod přes řád". Pro vyjádření těchto dvou různých hodnot je zapotřebí dvou různých symbolů. Bývá zvykem používat symboly **0** nebo **O** (pro počet žádný) a **1** nebo **I** (pro počet jeden).

Každý zápis číselné hodnoty ve dvojkové soustavě proto může obsahovat pouze symboly 0 a 1. Je tedy např. 1001 jistě zápis číselné hodnoty ve dvojkové soustavě. Aby bylo zcela jasné, že jde o zápis právě ve dvojkové soustavě, zapisuje se v nejednoznačných případech hodnota jako $1001_2$ nebo $IOOI_2$ na rozdíl např. od $1001_{10}$.

Protože většina z nás je zvyklá chápat počty vyjádřené zápisem čísla jen v desítkové soustavě, vysvětlíme způsob převodu zápisu nějaké (např. uvedené 1001) číselné hodnoty ze dvojkové do desítkové soustavy. Je analogicky jako shora

| $2^3 = 8_{10}$ | | $2^2 = 4_{10}$ | | $2^1 = 2_{10}$ | | $2^0 = 1_{10}$ | | **celkem** |
|---|---|---|---|---|---|---|---|---|
| x **1** | + | x **0** | + | x **0** | + | x **1** | = | $9_{10}$ |

Obdobně zápis necelého čísla - nejde dost dobře říci ani "desetinného" čísla ani čísla s "desetinnou částí" - např. $1001{,}011_2$:

| Celá část | | | | , | "Necelá" část | | | celkem |
|---|---|---|---|---|---|---|---|---|
| $2^3 = 8_{10}$ | $2^2 = 4_{10}$ | $2^1 = 2_{10}$ | $2^0 = 1_{10}$ | | $2^{-1} = 1/2_{10} = 0{,}5_{10}$ | $2^{-2} = 1/4_{10} = 0{,}25_{10}$ | $2^{-3} = 1/8_{10} = 0{,}125_{10}$ | |
| x 1 | x 0 | x 0 | x 1 | | x 0 | x 1 | x 1 | |
| 8 | 0 | 0 | 1 | | 0 | 0,25 | 0,125 | = $9{,}375_{10}$ |

Opačný převod - tj. převod zápisu nějaké číselné hodnoty ze soustavy desítkové do soustavy dvojkové - spočívá v postupném zkoumání, kolikrát se v čísle vyskytují jednotlivé mocniny dvou. Ukažme tento postup např. pro číselnou hodnotu $19_{10}$:

- Nejnižší mocnina dvou, která se v $19_{10}$ už *nevyskytuje*, je pátá: $2^5 = 32_{10}$. Začneme tedy čtvrtou mocninou:
- Čtvrtá mocnina 2 ($2^4 = 16$) se v 19 vyskytuje **1**krát a zbude $3_{10}$.
- Třetí mocnina 2 ($2^3 = 8$) se v 3 vyskytuje **0**krát a zbude $3_{10}$.
- Druhá mocnina 2 ($2^2 = 4$) se v 3 vyskytuje **0**krát a zbude $3_{10}$.
- První mocnina 2 ($2^1 = 2$) se v 3 vyskytuje **1**krát a zbude $1_{10}$.
- Nultá mocnina 2 ($2^0 = 1$) se v 1 vyskytuje **1**krát a zbude 0.

Hodnota $19_{10}$ je tedy rovna **$10011_2$**.

S ohledem na probíranou problematiku ještě jedna poznámka: obsah jednoho bytu může být interpretován jako zápis právě osmiciferného dvojkového čísla. Nejmenší číselná hodnota je osm nul, tedy nula. Největší číselná hodnota je osm jedniček. Postup uvedený shora dává pro hodnotu $11111111_2$ desítkový ekvivalent

$$\mathbf{11111111}_2 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = \mathbf{255}_{10}$$

Všech různých obsahů jednoho bytu je tedy 256.

## Šestnáctková soustava

Jde o číselnou soustavu se základem 16. Pro zápis číselné hodnoty v této soustavě musí tedy být k disposici 16 různých symbolů, označujících počty nula, jedna ... až patnáct. Ve shodě s teorií čísel se tyto symboly nazývají *cifry* a celá současná počítačová civilizace používá následující:

**Cifry šestnáctkové soustavy**

| Symbol | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Počet | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Způsob převodu zápisu hodnoty v šestnáctkové soustavě do soustavy desítkové je zcela analogický předchozímu odstavci - jediný rozdíl je v základu, který je nyní 16.Ukažme to na příkladu hodnoty $2AF3_{16}$ (značí A počet 10 a F počet 15 - viz předchozí tabulka):

| $16^3 = 4096_{10}$ | | $16^2 = 256_{10}$ | | $16^1 = 16_{10}$ | | $16^0 = 1_{10}$ | | celkem |
|---|---|---|---|---|---|---|---|---|
| x 2 | + | x 10 | + | x 15 | + | x 3 | = | $10\ 995_{10}$ |

Šestnáctková soustava je nejčastěji používanou soustavou v počítačovém a vůbec digitálním světě. Velmi usnadňuje a zpřehledňuje zápis stavů paměťových elementů, které jsou fyzikálně tvořeny dvoustavovými konstrukčními prvky (generalizovanými jako *bity*). Platí totiž toto: cifry šestnáctkové soustavy označují počty 0 až 15 podle definice. Ovšem počty 0 až 15 jsou rovněž všemi hodnotami, které tvoří kombinace různých hodnot v zápisu 4-bitového čísla: od $0000_2 = 0_{10}$ do $1111_2 = 8 + 4 + 2 + 1 = 15_{10}$. Je-li tedy např. obsah jednoho bytu chápaný jako osmiciferné dvojkové číslo, pak hodnota v něm uložená lze zapsat právě 2 ciframi šestnáctkové soustavy:

| Byte: | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Obsah | I | O | I | 0 | O | I | I | O |
| šestnáctkově. | A | | | | 6 | | | |

Zápis hodnot jednotlivých bytů je nejčastějším použitím šestnáctkové soustavy. Zatímco ve dvojkové je zapotřebí 8 cifer, v šestnáctkové jen 2 - a to je daleko přehlednější. Místo intervalu hodnot uchovatelných v jednom bytu zapsaných ve dvojkové soustavě <00000000; 11111111> je jistě zápis v šestnáctkové soustavě <00; FF> příjemnější. Z tohoto příkladu jeden důležitý závěr: protože F je nejvyšší cifrou šestnáctkové soustavy, je FF největší dvoucifernou hodnotou zapsanou v šestnáctkové soustavě. Je přitom $FF_{16} = 15 \times 16 + 15 \times 1 = 15 \times 17 = 255_{10}$.

Známým příkladem použití zápisu hodnoty v šestnáctkové soustavě je kódování barev v HTML souborech: parametr *color* = *#RRGGBB* určuje barvu jako trojici intenzit základních barev (po řadě R = red = červená, G = green = zelená, B = blue = modrá), každá ve stupních intenzit od 0 do 255, zapsaných jako šestnáctkové číslo. Tedy kód #FF00FF znamená: červená složka naplno, zelená není, modrá naplno - dohromady tedy barva fialová (magenta).

# Datové typy

Jednou z nejdůležitějších charakteristik výpočetních systémů je to, s jakými datovými typy dovedou pracovat - jak dovedou pohlížet na obsah jednoho nebo více po sobě jdoucích bytů a tento obsah příslušným způsobem interpretovat. V tomto článku jsou diskutovány pouze *numerické* datové typy. Názvy datových typů uváděných v tomto článku nejsou univerzální a liší se podle kontextu použití; v textu tohoto článku jsou jako názvy typů použity identifikátory z programovacích jazyků C, Basic event. Java, konkrétně **C++**, **Visual Basic (VB)** a **J#** z edice **Microsoft Visual Studio 2010** a novějších.

## Celočíselné datové typy

### Byte bez znaménka

Datový typ označovaný v C++ jako ***unsigned char*** a ve VB jako ***Byte*** je takové chápání obsahu jednoho bytu, při kterém je nazírán jako osmice dvojkových cifer (osmiciferný zápis dvojkové hodnoty) a tedy jako nezáporné celé číslo. Takových různých hodnot je 256, hodnotami datového typu byte jsou všechna čísla od 0 do (desítkově) 255 - viz závěr odstavce *Dvojková soustava*.

| Byte bez znaménka: | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Obsah | I | O | O | O | O | I | O | I |
| bitu č. | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Pozn**.: Desítková hodnota čísla z tohoto příkladu je $1 \times 2^7 + 1 \times 2^2 + 1 \times 2^0 =$ **133**.

### Byte se znaménkem

Datový typ označovaný v C++ jako ***signed char*** nebo také jen ***char*** a ve VB jako ***SByte*** (signed byte) je takové chápání obsahu jednoho bytu, při kterém je první bit zleva (bit č. 7) považován za znaménko a ostatní bity v počtu 7 (v prvním přiblížení) jako absolutní hodnota. Příklad:

| Byte se znaménkem: | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Obsah | I | O | O | O | O | I | O | I |
| bitu č. | 7 = znam. | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Pozn**.: Desítková hodnota čísla z tohoto příkladu je podle striktní interpretace shora uvedené definice následující: absolutní hodnota je $1 \times 2^2 + 1 \times 2^0 =$ **5**, ale protože 7.bit není nulový, měla by být hodnota rovna **-5**. Tvůrci procesorů však uvažovali dál: pokud by to bylo skutečně takto, pak nuly budou dvě: jedna kladná a jedna záporná, což je totéž. Jde tedy jedna kombinace bitů v bytu využít pro další hodnotu. To lze udělat dvěma způsoby - viz níže odstavec *Záporná celá čísla*. Byte se znaménkem se ukládá ve dvojkovém doplňkovém kódu, proto zahrnuje interval hodnot <-128, +127>,

## 2 byty bez znaménka

Datový typ označovaný v C++ jako ***unsigned short*** a ve VB jako ***UShort*** (unsigned short integer) je takové chápání obsahu dvou po sobě jdoucích bytů, při kterém jsou jejich bity nazírány jako šestnáctice dvojkových cifer (šestnácticiferný zápis dvojkové hodnoty) a tedy jako nezáporné celé číslo. Takových různých hodnot je 65 536, proto hodnotami tohoto datového typu jsou všechna čísla od 0 do (desítkově) 65 535. Graficky lze jednu hodnotu tohoto typu znázornit např. takto:

**2 byty bez znaménka:**

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Obsah | I | O | O | O | O | I | O | O | O | O | O | I | O | O | O | I |
| bitu č. | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | O |

**Pozn**.: Desítková hodnota čísla z tohoto příkladu je $1 \times 2^{15} + 1 \times 2^{10} + 1 \times 2^4 + 1 \times 2^0$ = **33 809**.

## 2 byty se znaménkem

Datový typ označovaný v C++ jako ***signed short*** nebo také jen ***short*** a ve VB jako ***Short*** (short integer) je takové chápání obsahu dvou po sobě jdoucích bytů, při kterém je první bit zleva (bit č. 15) považován za znaménko a ostatní bity v počtu 15 (v prvním přiblížení) jako absolutní hodnota. Příklad:

**2 byty se znaménkem:**

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Obsah | I | O | O | O | O | I | O | O | O | O | O | I | O | O | O | I |
| bitu č. | 15 = znam. | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | O |

**Pozn**.: Desítková hodnota čísla z tohoto příkladu je podle striktní interpretace shora uvedené definice následující::absolutní hodnota je $1 \times 2^{10} + 1 \times 2^4 + 1 \times 2^0$ = **1 041**, ale protože první bit zleva (bit č. 15) není nulový, měla by být hodnota rovna **-1 041**. Platí tady však stejná poznámka jako u typu *Byte se znaménkem* (viz) a stejný odkaz na odstavec *Záporná celá čísla*. Dva byty se znaménkem se ukládají ve dvojkovém doplňkovém kódu, proto zahrnuje interval hodnot <-32768, +32767>,

## 4 byty bez znaménka

Datový typ označovaný v C++ jako ***unsigned long*** a ve VB jako ***UInteger*** je takové chápání obsahu čtyř po sobě jdoucích bytů, při kterém jsou nazírány jako dvaatřiceticiferné dvojkové číslo (analogie typu *2 byty bez znaménka* - viz). Hodnoty tohoto datového typu se pohybují od nuly až po trochu více než +4 miliardy. Přesně to je interval <0 ; +4,294,967,295>.

Logika uložení je zcela ekvivalentní uložení typů Byte a UShort (viz), ovšem v paměťovém prostoru 4 bytů.

## 4 byty se znaménkem

Datový typ označovaný v C++ jako ***signed long*** nebo také jen ***long*** a ve VB jako ***Integer*** je takové chápání obsahu čtyř po sobě jdoucích bytů, při kterém je první bit zleva (bit č. 31) považován za znaménko a ostatní bity v počtu 31 (v prvním přiblížení) jako absolutní hodnota (analogie typu *2 byty se znaménkem* - viz). Platí tady však stejná poznámka jako u typu *2 byty se znaménkem* a stejný odkaz na odstavec *Záporná celá čísla*. Čtyři byty se znaménkem se ukládají ve dvojkovém doplňkovém kódu, hodnoty tohoto datového typu se pohybují od trochu méně než -2 miliardy až po trochu více než +2 miliardy. Přesně to je interval <-2,147,483,648 ; +2,147,483,647>.

Logika uložení je zcela ekvivalentní uložení typů SByte a Short (viz), ovšem v paměťovém prostoru 4 bytů.

## 8 bytů bez znaménka

Datový typ označovaný v C++ jako ***unsigned long long*** a ve VB jako ***ULong*** je takové chápání obsahu osmi po sobě jdoucích bytů, při kterém jsou nazírány jako šedesátičtyřciferné dvojkové číslo (analogie typu *2 byty bez znaménka* - viz). Hodnoty tohoto datového typu se pohybují od nuly až po trochu více než $1{,}8 \times 10^{19}$ nebo také 16 milionů Terra. Přesně to je interval <0 ; +18,446,744,073,709,551,616>. Takto velká paměť by měla 16 777 216 TB.

Logika uložení je zcela ekvivalentní uložení typů Byte a UShort (viz), ovšem v paměťovém prostoru 8 bytů.

## 8 bytů se znaménkem

Datový typ označovaný v C++ jako ***signed long long*** nebo také jen ***long long*** a ve VB jako ***Long*** je takové chápání obsahu osmi po sobě jdoucích bytů, při kterém je první bit zleva (bit č. 63) považován za znaménko a ostatní bity v počtu 63 (v prvním přiblížení) jako absolutní hodnota (analogie typu *2 byty se znaménkem* - viz). Platí tady však stejná poznámka jako u typu *2 byty se znaménkem* a stejný odkaz na odstavec *Záporná celá čísla*. Osm bytů se znaménkem se ukládá ve dvojkovém doplňkovém kódu, hodnoty tohoto datového typu se pohybují od trochu méně než -9 x $10^{18}$ až po trochu více než +9 x $10^{18}$. Přesně to je interval <-9 223 372 036 854 775 808 ; +9 223 372 036 854 775 807>.

Logika uložení je zcela ekvivalentní uložení typů SByte a Short (viz), ovšem v paměťovém prostoru 8 bytů.

## Záporná celá čísla

Pro úsporu jedné kombinace bitů při uchování celého čísla v paměti (nula je nulou ať kladná nebo záporná) se využívají dvě techniky: dvojkový doplňkový kód a kód posunuté nuly.

### Dvojkový doplňkový kód

V tomto režimu je první bit zleva považován za znaménkový. Je-li roven nule, je číslo kladné a zbývající bity vyjadřují přímo jeho hodnotu. Má-li se zaznamenat číslo záporné, pak se nejprve provede operace binární negace (záměna nul a jedniček) absolutní hodnoty, načež se výsledek zvětší o 1.

Příklad pro hodnotu N = $-9_{10}$ na jednom bytu: abs (N) = $9_{10}$ = $00001001_2$, jehož dvojková negace je $11110110_2$ a hodnota o 1 větší je $11110111_2$. To je tedy zobrazení hodnoty -9 na jednom bytu ve dvojkovém doplňkovém kódu.

Procesory Intel a další používají tento způsob např. pro zpracovávaná, přímo adresovatelná data.

### Kód posunuté nuly

V tomto režimu není uložena přímo číselná hodnota N, ale tato hodnota zvětšená o vhodnou konstantu K Uložena je tedy hodnota T = N + K.. Konstanta je volena tak, aby výsledek byl už vždy nezáporné číslo. Pro získání skutečné hodnoty N je tedy naopak třeba provést operaci N = T - K.

Volba konstanty se řídí velikostí paměťového prostoru určenému pro uložení celého čísla. Typicky se za konstantu volí hodnota získaná vyplněním určeného paměťového prostoru samými 1 kromě prvního bitu zleva, který se nastavuje na 0.

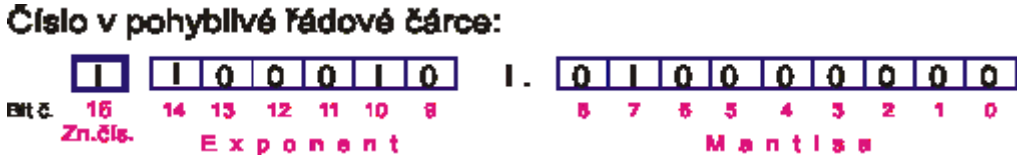
Příklad pro hodnotu N = $-9_{10}$ na jednom bytu: konstanta K pro 8 bitů je rovna $01111111_2$ = $127_{10}$. Uložena je tedy hodnota T = -9 + 127 = $118_{10}$ = $01110110_2$. To je tedy zobrazení hodnoty -9 na jednom bytu v kódu posunuté nuly

Procesory Intel a další používají tento způsob např. pro exponentovou část při uchování čísla v pohyblivé řádové čárce.

# Neceločíselné datové typy

## Obecně o formátu uložení

Složitější je takový datový typ, který má vyjadřovat také ne-celé číslo (záměrně se vyhýbáme terminu desetinné číslo). V počítačovém prostředí nejrozšířenější je způsob definovaný standardem IEEE 754 (Institute of Electrical and Electronics Engineers) pro binární aritmetiku v pohyblivé řádové čárce. Je to takové chápání obsahu několika po sobě jdoucích bytů, při kterém jsou byty nazírány jako tři skupiny bitů. První skupinou je první bit zleva chápaný jako *znaménko* celého čísla. Druhou skupinou je několik bitů chápaných jako celé číslo a označovaných jako *exponent*. Třetí skupinou je několik bitů chápaných (a označovaných) jako *mantisa*. Ve finální interpretaci jde o zápis dvojkového čísla v normovaném semilogaritmickém tvaru (viz nahoře). Graficky lze logiku uložení jedné hodnoty tímto způsobem znázornit na příkladu dvoubytového uložení čísla např. takto:



První bit zleva je chápán jako znaménko **Z** celého čísla: O znamená kladné, I záporné číslo. V příkladu na obrázku shora tedy jde o číslo záporné.

Další skupina bitů slouží k uchování exponentu **E**. Jako formát uložení je použit způsob s kódem posunuté nuly (viz předchozí kapitola). Konstanta K posunu je rovna kladné celočíselné hodnotě získané vyplněním všech bitů exponentové části jedničkami kromě prvního bitu zleva, který je vyplněn nulou. Na obrázku shora je tedy konstanta K rovna $011111_2 = 31_{10}$. Protože hodnota uložená v exponentové části je rovna $100010_2 = 34_{10}$, je exponent E = 34 - 31 = 3.

Nyní k části **M** pro mantisu: jak bylo uvedeno shora v kapitole *Bit a byte*, lze každou hodnotu v jakékoliv soustavě zapsat v normovaném semilogaritmickém tvaru - tedy takovém, kdy celá část mantisy má jedinou, a to nenulovou cifru. Ve dvojkové soustavě tedy touto cifrou musí být jednička. Jestliže má popisovaný způsob vést k normovanému semilogaritmickému tvaru, pak každá uchovaná hodnota (kromě nuly) musí mít jako celou část mantisy jedničku - ale v tom případě je zbytečné ji explicitně v datech uchovávat. Proto tato jednička je pouze "myšlená", uvažuje se s ní při převodech, ale sama se nezaznamenává. Na obrázku shora je tedy mantisa M rovna $1.01_2 = 1 + 1/2^{-2} = 1{,}25_{10}$.

Samotná uložená hodnota je pak rovna $\mathbf{H} = \pm \mathbf{M} \times 2^{\mathbf{E}}$. Na obrázku shora tedy je uložena hodnota $-1.25 \times 2^3 = -1{,}25 \times 8 = \mathbf{-10_{10}}$.

## Formáty FPU procesorů Intel a jejich klonů

Následující tabulka upřesňuje shora uvedený obecný model uložení pro konkrétní typ procesoru. Datové typy *Single* (z angl. Single Precision Floating Point) a *Double* používá většina programovacích jazyků, typ Extended je použit také např. jako interní pracovní datový typ FPU procesorů Pentium a dalších.

| Název v C++ / VB | Délka [bytů] | Část exponentu [bity] | Konstanta K posunu | Část mantisy [bity] | Maximální hodnota | Minimální nenulová hodnota | Max. počet platných cifer |
|---|---|---|---|---|---|---|---|
| float / Single | 4 | 30 - 23 | 127 | 22 - 0 | 3.40E+38 | 1.4E-45 | 7 |
| double / Double | 8 | 62 - 52 | 1023 | 51 - 0 | 1.79E+308 | 4.9E-324 | 17 |
| nemá / nemá | 10 | 78 - 64 | 16383 | 63 - 0 | 1.18E+4932 | 3.6E-4951 | 20 |

# Typ Decimal

Tento datový typ se používá pro přesné uložení specielní podmnožiny racionálních čísel. Vychází vstříc finančníkům, pro které je nemožnost přesného uložení např. jednoho haléře jako 0.01[Kč] zcela nepředstavitelná. Je bázováno desítkovou soustavou, ovšem v binárním uložení. Typ Decimal je často realizovaný softwarově a tedy zpracovávaný neskonale déle než předchozí hardwarové typy.

Mějme celočíselnou hodnotu C, mějme nezápornou celočíselnou hodnotu D. Vytvořme racionální číslo X takto: $X = C / 10^D$. Je-li D=0, je tedy X=C, jinak je X rovno hodnotě C s **desetinnou** tečkou (čárkou) v jeho **desítkovém** zápisu "posunutou doleva" o D míst.

Hardwarové uložení hodnot čísla X tohoto typu využívá 128 bitů = 16 bytů paměti. Těchto 16 bytů je logicky rozděleno na 2 části:

- 12 bytů (= 96 bitů) celočíselná hodnota N bez znaménka jako absolutní hodnota C,
- 4 byty (=32 bitů) obsahující mj. desítkový faktor D a znaménko čísla C.

Část N celočíselné hodnoty má 96 bitů (12 bytů) a je absolutní hodnotou čísla C. Hodnota N je tedy větší nebo rovna nule a menší nebo rovna $2^{96}-1$.

Část F obsahující desítkový faktor a znaménko má 32 bitů (4 byty, bity číslovány zleva) je rozdělena takto:

- 31. bit je znaménkový; 0 pro kladná C a nulu, 1 pro záporná C a nulu.
- Bity 30 až 24 jsou nevyužité.
- Bity 23 - 16 (tj. druhý byte zleva) obsahuje hodnoty desítkového faktoru D z intervalu <0;28> (tj. $00_{16}$ - $1C_{16}$).
- Bity 15 - 0 (poslední 2 byty zprava) jsou nevyužité.

Interval nenulových absolutních hodnot Decimal je tedy od $(2^{96}-1) \times 10^{-28}$ (pro D=28) do $2^{96}-1$ (pro D=0).

Část N lze logicky rozdělit na 2 skupiny celočíselných hodnot bez znaménka: 32 nejvyšších bitů (4-bytové číslo bez znaménka) a 64 nejnižších bitů (8-bytové číslo bez znaménka). Označme je $N_{high}$ a $N_{low}$. Při práci s hodnotami typu Decimal používá Microsoft tento způsob uložení:

- Část F,
- část $N_{high}$,
- část $N_{low}$.

Všechny tři části jsou uloženy jako little endian (viz dále).

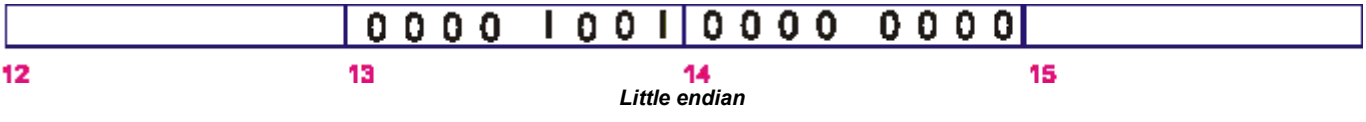
# Endians

Tento etymologicky zajímavý pojem použitý již Johnatanem Swiftem je mezi počítačovou veřejností znám spíše ve spojení Little-endian a Big-endian.

Jde o následující problém: Celý tento článek mluví o několika bytech umístěných v paměti "vedle sebe". Ony byty však mají konkrétní adresy. Mluvíme-li např. o 2-bytové hodnotě $9_{10} = 1001_2$ typu Unsigned Short Integer = 0000 0000 0000 $1001_2$, jak je tato hodnota umístěna v paměti třeba od adresy 13? Takto


***Big endian***

nebo takto:



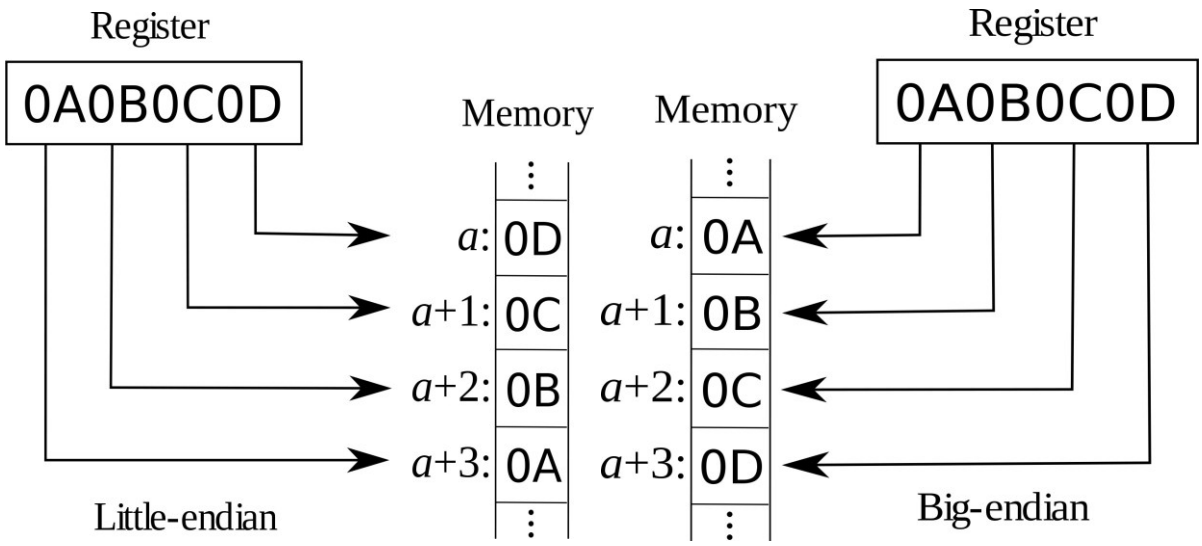

***Little endian***

jinak řečeno, je první umístěný byte ten nejvíce či nejméně významný? Je dobré si v tomto případě uvědomit, že druhý uvedený případ je ekvivalentní tomuto:


***Little endian***

tj. jenom "kreslíme" adresaci paměti opačně než v prvním případě. Ve všech třech případech už jsou pod schémata uvedena označení pomocí "endian". Toto není jen otázka umístění vícebytových dat v paměti. Například při přenosu dat USB (= sériovým) portem je nejdříve z každého bytu vysílán bit 7 nebo bit 0? Uvědomme si dále, že i v hovorové řeči používáme např. dvacátý první (=Big endian) nebo jedenadvacátý (=Little endian).

Registry procesorů Intel jsou tvořeny řadou bitů s nejvyšším bitem vlevo. Při přenosu z a do paměti pak rozdíl mezi Little-endian a Big-endian pěkně znázorňují schémata uvedená v [1]:




Nutno uvést, že procesory Intel a programovací jazyky a systémy minimálně Microsoftu používají Little endian. Celá řada operačních systémů pracujících na procesorech x86 a x86-64 používá Little-endian, stejně tak řada jiných systémů pracujících např. na procesorech SPARC, Motorola a dalších používá Big-endian.

# Přesnost uložení hodnot

## Obecné úvahy

Malé opakování učiva z mateřské a základní školy: čísla máme *přirozená*, *celá*, *racionální* a *reálná*. Uložit v počítačovém prostředí např. přirozenou trojku žádný problém nečiní - viz výše. Uložit tam však racionální 1/3 = 0.33333..... je problém primárně filozofický. Už Alexandre Dumas vložil do úst svého d'Artagnana kouzelný výrok:

> Jedním ze základních pravidel matematických je, milý Porthe, že nádoba musí býti větší obsahu!

Aplikováno na nádobu (=konečné počítačové prostředí) a obsah (=nekonečné číslo) vidíme zřetelný spor. Vyplývá z toho, že žádné číslo s nekonečným rozvojem v konečném počítačovém prostředí přímo neuložíme. Uložíme sice číslo hodně blízké, ale to už není to nekonečné. Z toho vyplývá nutnost zkoumat, hodnoty kterých číselných kategorií uložíme a kterých ne. Znovu zdůrazněme, že jde o *přímé uložení hodnoty některým ze shora popsaných způsobů*, nikoliv o obcházení přes nějaké datové struktury apod.

Jak je zřejmé z popisu uložení *celočíselných* hodnot, zde problémy s přesností nenastávají. Každá celočíselná hodnota z intervalu přípustného pro daný typ a daný paměťový prostor se v něm uloží bez ztráty přesnosti. Chápeme-li čísla přirozená jako podmnožinu čísel celých, vyplývá z toho závěr: každé přirozené a celé číslo - až do jisté maximální velikosti - v počítačovém prostředí uložíme.

Jiná je však situace u *ne-celočíselných* hodnot.

Těmi jsou především čísla *racionální*. Některá jistě uložíme - viz shora v příkladu hodnotu 1,25. Některá ovšem ne - viz zmíněné hodnoty s nekonečným rozvojem. Jak je z popisu způsobů uložení zřejmé, množina uložitelných racionálních čísel je konečná - sice relativně velká, ale proti nekonečnému počtu racionálních čísel je trapně malá. Proto je důležitá informace o zaokrouhlovací chybě vzniklé při pokusu uložit nekonečné racionální číslo. Tuto chybu nelze obecně vyjádřit v absolutní hodnotě, protože uložená čísla se liší řádově. Lze však jednoznačně udat, na kolik platných cifer se lze spolehnout - kolik platných cifer určitě tvoří počátek zápisu jakékoliv racionální hodnoty.

U shora popsaných datových typů Single, Double a Extended je to zřejmé: je to počet bitů části pro mantisu + 1 (myšlená 1.) - 1 (poslední cifra už může být zaokrouhlená). Tedy např. u typu Single to je 23 cifer - ovšem dvojkových. Pro běžný život je to nic neříkající hodnota. Uvažme proto, že maximální číslo mající ve dvojkovém zápisu 23 cifer je $111,1111,1111,1111,1111,1111_2 = 2^{23} - 1 = 8\,388\,607_{10}$ a to má 7 cifer desítkových. Ale ne všechna 7-ciferná desítková čísla uložíme - už ne např. číslo o 1 větší. Proto závěr: datový model Single ukládá hodnoty s přesností max. 7 platných (desítkových) cifer. Obdobným způsobem se dojde i k maximálnímu počtu platných cifer u dalších modelů - viz tabulka v odstavci *Formáty FPU* shora.

Protože čísla *reálná* mají obecně nekonečný rozvoj, platí o nich - zvláště o přesnosti uložení - to samé jako pro čísla racionální.

## Hodnoty $10^{-n}$

Jednou z velmi nepříjemných skutečností binárního počítačového světa je pro nás, kteří uvažujeme zásadně v desítkové soustavě, toto: *žádná z hodnot tvaru $10^{-n}$ (n=1, 2, 3, ...) nemá konečný dvojkový rozvoj*. Uvědomíme-li si, že jde o tak běžně používané hodnoty (desítkově) 0,1, 0,01, 0,001 atd. pak skutečnost, že žádnou z těchto hodnot nelze popsanými modely uložit, je pro mnohé velkým překvapením.

Např. jedna desetina má dvojkový rozvoj 0.0001 1001 1001 1001 1001 1001 1001 1 ... Při ukládání této hodnoty se musí vzít jen takový počet platných cifer (počínaje první jedničkou), kolik bitů + 1 má část pro mantisu. Jestliže první dvojková cifra, která se už neuloží, je nula, pak je vše zřejmé, uložená hodnota je o trochu menší než jedna desetina. Jestliže je to však jednička, pak to zřejmé není: vznikne uložený zbytek zaokrouhlením nebo uříznutím? V prvním případě je uložená hodnota o trochu větší než desetina, ve druhém případě trochu menší než desetina.

Rozdíl, o který se uložená hodnota liší od jedné desetiny, je např. v modelu Double okolo $6.6 \times 10^{-16}$. Zdálo by se, že je to tak málo, že hořejší úvahy jsou čistě akademické. Zdaleka tomu tak není. Jednak z toho plyne, že aplikační programy si nemohou dovolit testovat dvě ne-celočíselné hodnoty na rovnost. Pokud by se lišily byť jen na posledním bitu, procesor je vyhodnotí jako různé. Konstrukce typu **If P=S** ... jsou v programech vlastně nepřípustné. Vždy by se měla testovat absolutní hodnota jejich rozdílu; teprve když je menší než nějaká elementární hodnota určená programátorem podle povahy aplikace, teprve pak by se měly dvě hodnoty považovat za stejné.

Další nepříjemnou praktickou situací jsou cyklické výpočty s ne-celočíselnými hodnotami. Jestliže se totiž tisíckrát nakumuluje do původně nulové proměnné tisícina, výsledek určitě nebude roven přesně jedné. Buď bude o trochu menší nebo o troch větší podle toho, zda je ona tisícina zaokrouhlena nebo uříznuta. Proto nejde takový cyklus řídit podmínkou na dosažení hodnoty 1. I podmínky typu "... dokud je menší než 1" nebo "... dokud není větší než 1" jsou zavádějící, protože pak by se mohl cyklus provést třeba jen 999x.

Stačí si např. v Excelu spustit toto makro:

```
Sub TestTisiciny()
   Dim S As Double
   Dim i As Long
   S = 0
   For i = 1 To 1000
      S = S + 0.001
   Next
   MsgBox Format(S - 1, "0.00000000E+00")
End Sub
```

Výsledkem bude o kolik se S liší od jedné - a nebude to nula.

# Zaokrouhlování

Při přípravě dat jednorozměrného datového souboru je začasté kladena podmínka na nějakou formalizaci dat, např. na počet desetinných míst, na rovnost násobku nějakého čísla, na náhradu zlomku desetinným zápisem apod. I shora uvedené odstavce pracují s pojmem „zaokrouhlení“. Přitom osoby data připravující používají zcela automaticky jejich úpravu - de facto aproximují číslem „velmi podobným“. Tento odstavec velmi stručně popisuje věc zdánlivě zcela jasnou, ale při pečlivějším studiu lze ukázat, že je většinou populace nazírána velmi zjednodušeně. Nejčastější a stěžejní úlohou je zaokrouhlení na celé číslo (tj. aproximace celým číslem). Není to však jediná úloha - viz dále.

Pokud nebude výslovně řečeno jinak, bude v této kapitole X značit zaokrouhlované číslo, A pak zaokrouhlení čísla X.

# Zaokrouhlení na celé číslo

## Zaokrouhlení na nejbližší celé číslo

Jde zřejmě o nejznámější a nejčastější úlohu zaokrouhlování. Právě tak nám byla problematika zaokrouhlování prezentována počínaje základní školou. Zní: Nahraďte hodnotu reálného čísla X celočíselnou hodnotou. Obvykle je řešena následovně:

1. Pro každé reálné číslo X existuje jediné celé číslo A, pro které platí: $X \in \langle A-0.5; A+0.5)$.
2. Toto číslo A nazvěme **zaokrouhlením** čísla X.

Pro potřeby kapitoly o zaokrouhlování budeme značit A = [X].

Postup je často označován jako „zaokrouhlení na nejbližší“ (round to nearest). Většina programovacích jazyků a statistických programů obsahují funkci typu **Round(X)**, jejímž parametrem je obecně reálné číslo a výsledkem celočíselná hodnota (nebo reálná hodnota s nulovou desetinnou částí) rovna zaokrouhlené hodnotě podle popisu výše. Např. [2.4] = Round(2.4) = 2, [3.9] = Round(3.9) = 4, [7.5] = Round(7.5) = 8.

Při nezaujatém pohledu (a zvláště při pohledu zaujatém naší peněženkou po zrušení desetníků, dvacetníků a padesátníků) se však musíme ptát: hodnoty rovny celočíselné hodnotě plus 0.5 jsou evidentně ve výjimečném postavení - vždy se popisovanou definicí (k naší škodě) zaokrouhlují „nahoru“. Touto problematikou se zabývá samostatný odstavec níže.

## Zaokrouhlení nahoru

V anglosaské literatuře se používá také termín „zaokrouhlení ke kladnému nekonečnu“. Zaokrouhlením A čísla X je pak nejmenší celé číslo, které není menší než X. Je-li číslo A rovno takovému zaokrouhlení čísla X, značí se A = ⌈ X ⌉

V programovacích jazycích a statistickém software je pro toto zaokrouhlení používána funkce pojmenovaná Ceiling: A = Ceiling (X) nebo také RoundUp: A = RoundUp(X). Platí:

$$\lceil X \rceil = - \lfloor -X \rfloor$$

## Zaokrouhlení dolů

V anglosaské literatuře se používá také termín "zaokrouhlení k zápornému nekonečnu". Zaokrouhlení A čísla X je největší celé číslo, které hodnotou nepřesáhne X. Je-li číslo A rovno takovému zaokrouhlení čísla X, značí se A = ⌊ X ⌋.

V programovacích jazycích a statistickém software je pro toto zaokrouhlení používána funkce pojmenovaná Floor: A = Floor (X) nebo také RoundDown: A = RoundDown(X). Platí:

$$\lfloor X \rfloor = - \lceil -X \rceil$$

## Zaokrouhlení k nule

V anglosaské literatuře se používá také termín "zaokrouhlení od nekonečna". Zaokrouhlení A čísla X je to celé číslo, které vznikne odstraněním zlomková části X.

V programovacích jazycích a statistickém software je pro toto zaokrouhlení používána funkce pojmenovaná Truncate: A = Truncate (X), nebo také Fix: A = Fix (X). Platí:

$$\text{Truncate}(X) = \text{Sgn}(X) \cdot \lfloor |X| \rfloor = -\text{Sgn}(X) \cdot \lceil -|X| \rceil$$

kde funkce Sgn je funkce "znaménka" (signum): vrací +1 pro kladná, -1 pro záporná čísla, a 0 pro nulu.

## Zaokrouhlení od nuly

V anglosaské literatuře se používá také termín "zaokrouhlení k nekonečnu". Zaokrouhlení A čísla X je číslo X, je-li X celé. Není-li X celé, pak je to největší celé číslo A, pro něž je A < X (pro kladná X) nebo nejmenší celé číslo A, pro něž je X < A (pro záporná X).

Platí:

$$A = \text{Sgn}(X) \cdot \lceil |X| \rceil = -\text{Sgn}(X) \cdot \lfloor -|X| \rfloor$$

kde funkce Sgn je funkce "znaménka" (signum): vrací +1 pro kladná, -1 pro záporná čísla, a 0 pro nulu.

# Zaokrouhlení poloviny

Odstavec popisuje možné postupy zaokrouhlení reálného čísla X na celé číslo A, je-li zlomková část X rovna přesně 1/2.

## Zaokrouhlení poloviny nahoru

(také "ke kladnému nekonečnu"). Je-li zlomková část čísla X rovna přesně 1/2, je zaokrouhlení A = X + 0.5. Je tedy zaokrouhlení čísla 17.5 rovno 18 a zaokrouhlení -17.5 rovno -17. Platí:

$$A = \lfloor X + 0.5 \rfloor = - \lceil -X - 0.5 \rceil$$

Tento typ zaokrouhlení není symetrický, přesněji: způsobuje kladné zešikmení zaokrouhlovací chyby.

## Zaokrouhlení poloviny dolů

(také "k zápornému nekonečnu"). Je-li zlomková část čísla X rovna přesně 1/2, je zaokrouhlení A = X - 0.5. Je tedy zaokrouhlení čísla 17.5 rovno 17 a zaokrouhlení -17.5 rovno -18. Platí:

$$A = \lceil X - 0.5 \rceil = - \lfloor -X + 0.5 \rfloor$$

Tento typ zaokrouhlení podobně jako předchozí není symetrický, přesněji: způsobuje záporné zešikmení zaokrouhlovací chyby.

## Zaokrouhlení poloviny od nuly

V anglosaské literatuře se používá také termín "zaokrouhlení k nekonečnu". Je-li zlomková část čísla X rovna přesně 1/2, je zaokrouhlení A = X + 0.5 (pro kladná X) resp. A = X - 0.5 (pro záporná X). Je tedy zaokrouhlení čísla 17.5 rovno 18 a zaokrouhlení -17.5 rovno -18. Platí:

$$A = \mathrm{Sgn}(X) \cdot \lfloor |X| + 0.5 \rfloor = - \mathrm{Sgn}(X) \cdot \lceil -|X| - 0.5 \rceil$$

Tento typ zaokrouhlení zohledňuje kladné a záporné hodnoty symetricky a je bez celkového vychýlení, pokud jsou původní čísla kladná nebo záporná se stejnou pravděpodobností.

## Zaokrouhlení poloviny k nule

V anglosaské literatuře se používá také termín "zaokrouhlení od nekonečna". Je-li zlomková část čísla X rovna přesně 1/2, je zaokrouhlení A = X - 0.5 (pro kladná X) resp. A = X + 0.5 (pro záporná X). Je tedy zaokrouhlení čísla 17.5 rovno 17 a zaokrouhlení -17.5 rovno -17. Platí:

$$A = \mathrm{Sgn}(X) \cdot \lceil |X| - 0.5 \rceil = - \mathrm{Sgn}(X) \cdot \lfloor -|X| + 0.5 \rfloor$$

Tento typ zaokrouhlení stejně jako předchozí zohledňuje kladné a záporné hodnoty symetricky.

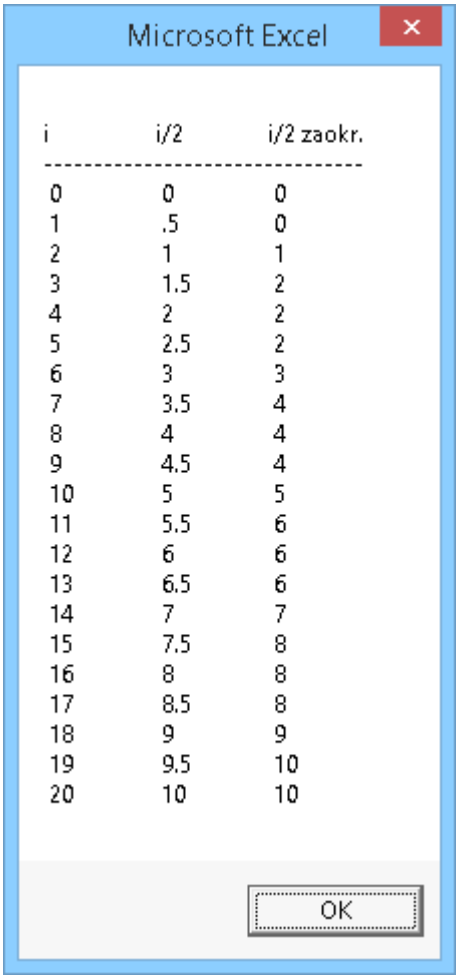
## Zaokrouhlení poloviny k sudé

Je-li zlomková část čísla X rovna přesně 1/2, je zaokrouhlení A rovno nejbližšímu sudému celému číslu. Je tedy zaokrouhlení čísla 17.5 rovno 18 (stejně jako zaokrouhlení 18.5), a zaokrouhlení -17.5 rovno -18 (stejně jako zaokrouhlení -18.5).

Tento typ zaokrouhlení stejně jako předchozí zohledňuje kladné a záporné hodnoty symetricky. Navíc pro rozumnou distribuci hodnot veličiny X je průměrná hodnota zaokrouhlených čísel stejná jako čísel původních. Tento typ zaokrouhlení je označován také jako nestranné, konvergentní, statistické, Holandské (Dutch), Gausovo, licho-sudé, bankéřské nebo přerušované (broken). Toto zaokrouhlení je také výchozím typem zaokrouhlení v "Normách pro aritmetiku pohyblivé řádové čárky" - IEEE-754.

Názorně to dokumentuje tento příklad: Zobrazte z čísel 0 až 20 jejich poloviny. Malý prográmek (např. tzv. makro třeba v Excelu) a jeho výsledek:

```
Sub ZaokrouhleniPoloviny()

   Dim i As Long
   Dim j As Long
   Dim r As Double
   Dim T As String

   T = "i" + vbTab + "i/2" + vbTab + "i/2
zaokr." + vbCrLf
   T = T + "--------------------------------
" + vbCrLf

   For i = 0 To 20
      j = i / 2
      r = i / 2
      T = T + Str(i) + vbTab + Str(r) +
vbTab + Str(j) + vbCrLf
   Next

   MsgBox T

End Sub
```




## Zaokrouhlení poloviny k liché

Je-li zlomková část čísla X rovna přesně 1/2, je zaokrouhlení A rovno nejbližšímu lichému celému číslu. Je tedy zaokrouhlení čísla 17.5 rovno 17 (stejně jako zaokrouhlení 16.5), a zaokrouhlení -17.5 rovno -17 (stejně jako zaokrouhlení -16.5).

Tento typ zaokrouhlení má stejné vlastnosti jako předchozí.

## Stochastické zaokrouhlení poloviny

Je-li zlomková část čísla X rovna přesně 1/2, je zaokrouhlení A rovno náhodně stanovené hodnotě X+0.5 a X-0.5, a to se stejnou pravděpodobností.

Tento typ zaokrouhlení je v podstatě rovněž bez celkového zkreslení, navíc je "spravedlivé" k lichým i sudým hodnotám A. Na druhé straně do výsledku vnáší náhodnou komponentu: Opakovaný výpočet na stejných datech může mít jiné výsledky.

## Alternativní zaokrouhlení poloviny

Jsou-li zlomkové části čísel X rovny přesně 1/2, použije se pro první takovou hodnotu zaokrouhlení nahoru, pro druhou zaokrouhlení dolů, pro třetí opět nahoru atd. Tato metoda sice odstraňuje náhodnou komponentu, při opakovaných výpočtech na datech sice stejných, ale v jiném pořadí může dávat odlišné výsledky.

## Zaokrouhlení na daný krok

Obecnější úlohou je zaokrouhlování na daný krok - např. na jednu setinu, na celé stovky, ale i na čtvrtminuty (= násobky 15 sec). Jedna z možností je využití výše popsané definice zaokrouhlení na celé číslo potažmo funkce Round:

Označme q daný krok (např. 100 pro zaokrouhlení na celé stovky, 0.001 pro zaokrouhlení na tisíciny). Zaokrouhlení reálného čísla X na krok q je pak hodnota

```
B = [X/q] . q
```

Zaokrouhlení hodnoty 123 456.789 na celé stovky je pak rovno součinu [1 234.56789]=1235 a 100, tj.ve výsledku 123 500.

Funkci Round z předchozího odstavce lze pak rozšířit přidáním druhého nepovinného parametru (není-li zadán, je roven 1):

1. Round (X, 1) = Round (X)
2. Round (X, q) = Round (X/q, 1) . q

Zaokrouhlení na definovaný počet m (desetinných) míst je pak rovno hodnotě funkce Round s druhým parametrem rovným $10^{-m}$ (tedy pro zaokrouhlení na tisíciny je m=-3, na celé stovky rovno 2).

# Literatura


[1] *Endianness*. [online]. Wikipedia: [cit.6.1.2013]. Dostupné z: http://en.wikipedia.org/wiki/Endianness.

[2] *Binární předpona*. [online]. Wikipedia: [cit.7.1.2013]. Dostupné z: http://cs.wikipedia.org/wiki/Bin%C3%A1rn%C3%AD_p%C5%99edpona.


Rev: 11 / 2021