Milan Práger; Irena Sýkorová
Jak počítače počítají

Pokroky matematiky, fyziky a astronomie, Vol. 49 (2004), No. 1, 32–45

Persistent URL: http://dml.cz/dmlcz/141207

Jak počítače počítají

Milan Práger a Irena Sýkorová, Praha

Podívejme se na práci počítače při provádění numerických výpočtů pro vědecké a technické účely. Máme na mysli takové výpočty, které se opírají o spojité matematické struktury, zejména o matematickou analýzu. Aplikace diskrétní matematiky ponechme stranou. Numerická výpočetní práce počítačů dnes pravděpodobně netvoří ani 10 % jejich činnosti, ale je třeba si uvědomit, že mnoho numerických výpočtů probíhá „za scénou“ při zpracování obrazu a zvuku, při počítačové grafice apod. Uplatnění výpočtů je dnes velmi široké, zmiňme se jen např. o numerické předpovědi počasí a o kosmickém výzkumu.

V tomto článku si všimneme toho, jak je v moderních počítačích vyřešen problém reprezentace reálných čísel, jak se provádějí aritmetické operace a jaké z toho plynou důsledky. Přesněji řečeno, jde o popis některých funkcí procesoru počítače. Programátor ovládá procesor tak, že vytvoří pro svůj výpočet program pomocí překladače některého programovacího jazyka (např. C, Fortran, Pascal). Při tom využívá možnosti, které mu překladač poskytuje. Překladač sám je program, který běží v rámci některého operačního systému (různé verze MS Windows, Linux, Mac OS, ev. další). Operační systém je základní program, který na počítači běží stále a řídí celou jeho práci.

Přestože základní princip je elementární, ukazuje se, že jeho neznalost nebo podcenění může vést k velkým ztrátám finančním a v extrémních případech dokonce ke ztrátám na lidských životech. Tato problematika je pěkně vyložena v knize [8], ze které jsme řadu věcí čerpali. Dalším pramenem je [13]. V nezbytné míře se těmito problémy zabývá každá učebnice numerické matematiky, např. [2].

Snažili jsme se, aby výklad byl srozumitelný bez zvláštních předběžných znalostí a aby byl přístupný i těm, kterým dosud počítač nepřirostl k srdci.

1. Reprezentace čísel v počítači

Vzhledem ke konstrukci počítačů se ukázalo jako nejvhodnější vyjádření čísel ve dvojkové soustavě. Každé číslo je zobrazeno jako řetězec dvojkových číslic 0 a 1, které odpovídají tomu, zda je nebo není napětí na tranzistoru nebo zda na miniaturních kondenzátorech je nebo není náboj. Dvojková číslice se nazývá *bit* (z anglického *binary digit*). Skupina osmi bitů se nazývá *byte* (slabika) a několik bytů tvoří počítačové *slovo*.

RNDr. Milan Práger, CSc. (1930), Matematický ústav AV ČR, Žitná 25, 115 67 Praha 1, e-mail: prager@math.cas.cz

RNDr. Irena Sýkorová (1956), katedra matematiky, Vysoká škola ekonomická, Ekonomická 957, 148 00 Praha 4, e-mail: sykorova@vse.cz

Při provádění numerických výpočtů je vhodné zavést dvojí typ počítačových čísel. Jsou to čísla typu *integer*, která reprezentují celá čísla, a čísla typu *real*, která reprezentují reálná čísla.[1]) Na rozdíl od matematiky, kde celá čísla tvoří podmnožinu množiny reálných čísel, jsou počítačové typy integer a real různé objekty. Reprezentace reálných čísel se dnes řídí americkou normou, jejíž hlavní myšlenky zmíníme v odst. 2.

V textu budeme užívat obvyklá označení pro číselné množiny: $\mathbb{R}$ – množina reálných čísel, $\mathbb{Z}$ – množina celých čísel, $\mathbb{N}$ – množina přirozených čísel, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

1.1. Typ integer

Každé celé číslo x můžeme vyjádřit ve tvaru

$$x = \operatorname{sgn}(x) \cdot \sum_{i=0}^{k} a_i 2^i, \quad \text{kde} \;\; a_i \in \{0,1\}, \;\; k \in \mathbb{N}_0. \tag{1}$$

Pro zobrazení čísla typu integer je v počítači většinou vyhrazeno 32 bitů. V (1) tedy zvolíme $k = 31$. Zobrazení nezáporných čísel je zřejmé, v pravém krajním bitu je uložen koeficient u 2^0 a levý krajní bit je určen pro koeficient u 2^{31}. Znaménko není v počítači reprezentováno. Je vidět, že tímto způsobem mohou být zobrazena nezáporná čísla od 0 (řetězec obsahující 32 nul) až po $2^{32} - 1$ (řetězec z 32 jedniček).

Například číslo $x = 37$ je zobrazeno jako

00000000000000000000000000100101

.

Binární zápisy čísel pomocí jedniček a nul jsou dlouhé a nepřehledné. Jejich zápis se zkrátí pomocí šestnáctkového (hexadecimálního) zápisu, kde se čtyři dvojkové číslice vyjádří jednou číslicí hexadecimální.[2]) Práce s čísly vyjádřenými hexadecimálně ovšem také vyžaduje jistý cvik. Počítačové vyjádření čísla 37 je hexadecimálně vyjádřeno osmi číslicemi: 00000025.

Pro zobrazení celých čísel (nezáporných i záporných) se nabízí tento postup. Jeden bit použijeme pro uložení znaménka, 0 pro nezáporné číslo a 1 pro záporné, a ve zbylých 31 bitech uložíme absolutní hodnotu čísla.

Je ale obvyklé postupovat jinak. Nezáporná čísla zobrazíme bez znaménka (tak, jak bylo již uvedeno) a místo záporných čísel zobrazíme jejich binární doplňky (komplementy). *Binární doplněk* záporného čísla $y = -x$ je *kladné* číslo $2^{32} - x$. Pro nezáporná čísla budeme pak mít rozsah $0 \leqq x \leqq 2^{31} - 1$ a pro záporná $-2^{31} \leqq y \leqq -1$. Levý krajní bit bude pro nezáporná čísla roven nule, pro záporná čísla bude roven jedné.

Záporné číslo $y = -37$, bude tedy zobrazeno v počítači jako

11111111111111111111111111011011

,

hexadecimálně FFFFFFDB.

[1]) Kromě toho jsou ještě další typy, např. *complex*, ale to zde nebudeme probírat.

[2]) Jednotlivé cifry v hexadecimální soustavě se označují 0, 1, ..., 9, A, B, C, D, E, F.

Důvod, proč se k zobrazení záporných celých čísel užívá binární doplněk, je ten, že odčítání a operace se zápornými čísly jsou nahrazeny operacemi jen s kladnými čísly. Při sčítání čísel x ($x > 0$) a $-x$ v počítači sčítáme totiž čísla x a $y = 2^{32} - x$, jejichž součet je 2^{32}. Při tom se už přenos jedničky z levého krajního bitu neuskuteční, protože překročí rozsah, a výsledkem bude řetězec 32 nul, tj. číslo 0.

Pro zobrazení celých čísel se užívá v programovacích jazycích kromě 32 bitů i 16 nebo 8 bitů. Zobrazení je zcela analogické. Například v jazyce Pascal se pro čísla typu integer používá 16 bitů, a chceme-li mít čísla délky 32 bitů, musíme je deklarovat jako longint, což znamená long integer.

1.2. Typ real

Celá čísla ovšem pro numerické výpočty nestačí. Je zapotřebí pracovat s obecnými reálnými čísly. Proto jsou v počítačích zavedena čísla typu real, ale na rozdíl od reprezentace celých čísel počítačovým typem integer, kde je možno (v jistém rozsahu) reprezentovat všechna celá čísla, je zřejmé, že při reprezentaci reálných čísel budou další omezení.

Jsou dva užívané způsoby, jak zobrazit reálné číslo v počítači. Nazývají se zobrazení *v pevné řádové čárce* (fixed point) a zobrazení *v pohyblivé řádové čárce* (floating point).

1.2.1. Zobrazení v pevné řádové čárce

Při zobrazení v pevné řádové čárce je počítačové slovo rozděleno na tři části. První část — 1 bit — je vyhrazena pro znaménko (0 pro plus, 1 pro minus), další část je určena pro binární reprezentaci celé části čísla (před řádovou čárkou) a poslední část je pro binární reprezentaci zlomkové části čísla (za řádovou čárkou).

Jestliže budeme uvažovat například 32bitové počítačové slovo rozdělené na 1, 15 a 16 bitů, číslo $x = \frac{19}{8} = 1 \cdot 2^1 + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}$ bude uloženo jako

0	000000000000010	0110000000000000

.

Nevýhodou tohoto způsobu zobrazování reálných čísel je malý rozsah čísel a malá přesnost čísel menších než jedna. Čísla x zobrazená v pevné řádové čárce jsou totiž v rozmezí $2^{-16} \leqq |x| < 2^{15}$ (a $x = 0$).

Dnes můžeme tento způsob považovat jen za historickou reminiscenci.

1.2.2. Zobrazení čísel v pohyblivé čárce

V současné době se používají, snad až na výjimky, čísla v pohyblivé řádové čárce. Toto zobrazení vychází z následující skutečnosti. Mějme kladné reálné číslo a a jeho logaritmus $\log_2 a$ (vzhledem ke konstrukci počítačů je vhodné vzít za základ logaritmů číslo 2). Tento logaritmus vyjádříme, jak je zvykem, ve tvaru $\log_2 a = c + m$, kde

$c \in \mathbb{Z}$ se nazývá *charakteristika* a $m \in \mathbb{R}$ se nazývá *mantisa* a ta splňuje $0 \leqq m < 1$. Napíšeme-li a ve tvaru

$$a = 2^{\log_2 a} = 2^c \cdot M,$$

kde $M = 2^m$, platí $1 \leqq M < 2$. Tento tvar kladného čísla se nazývá *normalizovaný tvar*. Název mantisa se přenese i na číslo M. Číslo $M - 1$ je nezáporné a nazveme je *zlomková část*. Číslo c nazveme *exponent*.[3])

Podle uvedeného vzoru se počítačové číslo v pohyblivé čárce konstruuje ze tří částí: znaménka (jeden bit, 0 pro plus, 1 pro minus), druhé části délky s bitů pro zobrazení exponentu a třetí části délky k bitů pro zlomkovou část (celá část mantisy je vždy 1, a proto ji nezobrazujeme). Počet bitů, které zobrazují exponent, je konečný, množina zobrazitelných exponentů je tedy omezená. Podobně máme i pro zlomkovou část konečný počet bitů. Nezobrazená číslice 1 se nazývá *skrytý bit*.

Pro určení typu počítačového čísla v pohyblivé řádové čárce je třeba zvolit počet s bitů pro exponent, s tím související konstanty $E_{\min}$ a $E_{\max}$ a počet bitů k pro zlomkovou část.

Nenulové *počítačové číslo* je pak racionální číslo tvaru

$$x = \operatorname{sgn}(x) \cdot \left(1 + \sum_{i=1}^{k} a_i 2^{-i}\right) \cdot 2^E, \tag{2}$$

kde $E \in \mathbb{Z}$, $E_{\min} \leqq E \leqq E_{\max}$ a $a_i \in \{0, 1\}$ jsou binární číslice.

Při jiné volbě hodnot s, $E_{\min}$, $E_{\max}$, k, dostaneme jiný typ zobrazení čísel v pohyblivé řádové čárce. O tom, jak se tato volba provádí, viz dále odst. 2.

Přesnost p zobrazení v pohyblivé řádové čárce je počet bitů mantisy včetně skrytého bitu, tedy $p = k + 1$.

Nejmenší počítačové číslo, které je větší než 1, je pak $x_1 = 1 + 2^{-(p-1)}$. Číslo

$$\varepsilon = x_1 - 1 = 2^{-(p-1)} \tag{3}$$

nazýváme *strojové epsilon* (*machine epsilon*). Toto číslo charakterizuje přesnost, s níž mohou být čísla v počítači zobrazena.

2. Norma ANSI/IEEE

Ještě v padesátých letech nebylo jasné, který způsob zobrazení reálných čísel v počítači je nejvýhodnější. Některé počítače užívaly pevnou řádovou čárku, jiné pohyblivou řádovou čárku. Jedním ze zastánců pevné řádové čárky byl např. John von Neumann, autor architektury prvních počítačů (srv. [8]). Chceme-li při použití pevné řádové čárky využít daný počet míst co nejlépe, musíme pro jednotlivé veličiny volit měřítka, tj. násobit je vhodnými koeficienty. A právě von Neumann soudil, že tato volba měřítek

[3]) Řádová čárka se nazývá pohyblivá proto, že se pohybuje tak, aby byla za první nenulovou číslicí.

by měla být svěřena programátorovi. Systematický výzkum zejména Jamese H. Wilkinsona ukázal, že výpočet v režimu pohyblivé řádové čárky, kde se vlastně provádí automatická volba měřítek, je lepší, i když část rozsahu čísla v počítači ztratíme na zobrazení exponentu.

Od začátku šedesátých let minulého století tento způsob naprosto převládl, snad i v důsledku dalšího rozvoje softwaru (programových prostředků) i hardwaru (technických prostředků). Většina počítačů ale měla svůj vlastní binární systém pohyblivé řádové čárky. Počítače řady IBM 360/370, které byly velmi rozšířené během 60. a 70. let minulého století, používaly hexadecimální systém. Existovalo mnoho odlišností mezi jednotlivými systémy a bylo těžké vytvořit přenositelný software. Standard pro binární reprezentaci čísel v pohyblivé čárce byl vyvinut na přelomu 70. a 80. let skupinou vědců pod vedením Williama Kahana. Tato skupina pracovala pod záštitou Institute for Electrical and Electronics Engineers, zkratka IEEE (v angličtině čteno „I triple E“).

Výsledky pracovní skupiny IEEE p754 byly převzaty do dvou norem USA, označovaných ANSI/IEEE p754 z r. 1985 [5] a ANSI/IEEE p854 z r. 1987 [6]. První je věnována pohyblivé řádové čárce ve dvojkové soustavě, druhá v obecné číselné soustavě. Budeme čerpat z první z těchto norem, zprostředkovaně přes [4], [8], a označovat ji jako normu ANSI/IEEE. Tato norma definuje dva základní typy zobrazování čísel typu real: v jednoduché přesnosti (single precision) a ve dvojnásobné přesnosti (double precision). Zobrazením čísel typu integer se norma nezabývá.

2.1. Jednoduchá přesnost

Pro zobrazení reálných čísel v jednoduché přesnosti se užívá 32 bitů. První bit je pro znaménko, dalších 8 bitů pro exponent a zbývajících 23 bitů pro zlomkovou část mantisy.

znaménko	exponent	$a_1, a_2, \cdots, a_{23}$

V prvním bitu je 0 pro kladné číslo, 1 pro záporné. Protože počítačový obraz exponentu E neobsahuje znaménko, je exponent zobrazen jako kladné binární číslo $E + 127$. Vztah mezi počítačovou reprezentací exponentu a jeho skutečnou hodnotou je popsán v tabulce 1. Skrytý bit, rovný 1, není v tomto tvaru zobrazen.

To znamená, že v počítači mohou být zobrazena čísla s exponenty od $E_{\min} = -126$ do $E_{\max} = 127$.

Nenulové počítačové číslo v jednoduché přesnosti tedy je

$$x = \operatorname{sgn}(x) \cdot \left(1 + \sum_{i=1}^{23} a_i 2^{-i}\right) \cdot 2^E, \quad \text{kde} \ \ -126 \leqq E \leqq 127, \ \ a_i \in \{0, 1\}. \qquad (4)$$

Přesnost tohoto typu zobrazení je $p = 24$.

Číslo $x = \frac{19}{8} = 1 \cdot 2^1 + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = \left(1 + 1 \cdot 2^{-3} + 1 \cdot 2^{-4}\right) \cdot 2^1$ je zobrazeno jako

0	10000000	00110000000000000000000

.

Hexadecimální zápis čísla x je 40180000.

Tabulka 1.

Skutečná hodnota E	Počítačová reprezentace E
-126	00000001
-125	00000010
-124	00000011
...	...
0	01111111
1	10000000
...	...
125	11111100
126	11111101
127	11111110

Každé celé číslo může být v počítači zobrazeno jako typ integer i jako typ real v pohyblivé řádové čárce. Abychom rozlišili oba typy, budeme za celé číslo v pohyblivé řádové čárce psát tečku.

Například číslo $x = 5$ (typ integer) bude zobrazeno jako

00000000000000000000000000000101

,

zatímco číslo $x = 5.$ (typ real v pohyblivé řádové čárce) má počítačovou reprezentaci

0	10000001	01000000000000000000000

.

Řetězec bitů

01000000010000000000000000000000

představuje číslo $x = 3.$, zobrazené jako typ real v pohyblivé řádové čárce, nebo číslo $x = 1\,077\,936\,128$ uložené jako typ integer.

Označme největší počítačové číslo symbolem $N_{\max}$ a nejmenší kladné počítačové číslo symbolem $N_{\min}$.

Číslo $N_{\min} = 2^{-126}$ je v počítači zobrazeno jako

0	00000001	00000000000000000000000

,

počítačová reprezentace čísla $N_{\max} = (2 - 2^{-23}) \cdot 2^{127}$ je

0	11111110	11111111111111111111111

.

V tabulce 1 nejsou obsaženy exponenty, které se zobrazují jako řetězec samých 0 nebo řetězec samých 1. Tyto exponenty jsou vyhrazeny pro speciální čísla.

2.2. Speciální čísla

Nula je speciální číslo, protože nemůže být vyjádřena v normalizovaném tvaru. V počítači jsou exponent i mantisa čísla nula zobrazeny samými 0, nulový je dokonce i skrytý bit, počítačová reprezentace čísla nula je

0 nebo 1	00000000	00000000000000000000000

.

Poznamenejme, že 0 a -0 jsou v běžné praxi dvě různé reprezentace téhož čísla.

Jinými speciálními čísly jsou ∞ a $-\infty$, dvě různá čísla, zobrazená jako

0 resp. 1	11111111	00000000000000000000000

.

Speciálními čísly jsou dále veličiny *NaN* (Not a Number). Spíše než čísla jsou to speciální hodnoty a jsou výsledkem neplatné operace (např. 0/0). Zobrazení *NaN* v počítači je podobné jako pro ∞, exponent se skládá ze samých 1, ale zlomková část je jiná (neobsahuje samé 0)

0 resp. 1	11111111	nikoli samé 0

.

Nakonec se zmíníme ještě o speciálních číslech zvaných *subnormální*. Tato čísla jsou v absolutní hodnotě menší než $N_{\min}$, nemohou být v daném rozsahu exponentů normalizována, jejich skrytý bit je 0. Pro zobrazení subnormálních čísel se použije nulový exponent a nenulová zlomková část. Nejmenší kladné číslo, které může být takto zobrazeno, je 2^{-149}

0	00000000	00000000000000000000001

.

Speciální čísla se použijí jako standardní výsledek, jestliže při provádění aritmetických operací nastane výjimečná situace a její signalizace není povolena. Výjimečné situace jsou: dělení nulou, přeplnění (absolutní hodnota výsledku je větší než $N_{\max}$), podplnění (absolutní hodnota výsledku je menší než $N_{\min}$), neplatná operace, např. 0/0, nepřesná operace (jestliže je výsledek zaokrouhlen). V případě poslední výjimky se speciální čísla neuplatní, výjimka se nesignalizuje a výsledkem je zaokrouhlená hodnota (viz dále odst. 3). Ve všech těchto případech výpočet pokračuje. Je na uživateli, aby zajistil jeho smysluplnost.

Jestliže se výjimečná situace signalizuje, je třeba, aby uživatel zajistil ošetření této situace. Jestliže to uživatel neučiní explicitně, dodá překladač standardní odpověď, která ohlásí chybu a výpočet ukončí.

Tyto situace zde není možné podrobně popisovat. Upozorňujeme jen na možnosti, které norma ANSI/IEEE uživatelům poskytuje.

2.3. Dvojnásobná přesnost — double format

Pro mnoho aplikací není jednoduchá přesnost dostatečná. Norma ANSI/IEEE proto definuje dvojnásobnou přesnost, která používá 64 bitů. Způsob zobrazování je stejný jako pro jednoduchou přesnost, liší se jednotlivé rozsahy. První bit je určen pro znaménko stejně jako u jednoduché přesnosti. Pro exponent je dalších 11 bitů a pro mantisu zbylých 52 bitů. Rozsah exponentů ve dvojnásobné přesnosti je od $E_{\min} = -1022$ do $E_{\max} = 1023$. V počítači jsou exponenty reprezentovány jako $E + 1023$. Nejmenší a největší zobrazitelná kladná čísla jsou

$$N_{\min} = 2^{-1022} \quad \text{a} \quad N_{\max} = (2 - 2^{-52}) \cdot 2^{1023}.$$

Čísla, která nelze v jednoduché přesnosti zobrazit přesně (např. čísla s nekonečným rozvojem), jsou ve dvojnásobné přesnosti zobrazena přesněji.

Norma ANSI/IEEE doporučuje ještě další typ, rozšířenou dvojnásobnou přesnost (double-extended format). Mikroprocesory Intel mají aritmetiku s rozšířenou dvojnásobnou přesností, která pracuje s 80bitovým registrem, kde je 1 bit pro znaménko, 15 bitů pro exponent a 64 bitů pro mantisu. První bit mantisy není skrytý, ale je explicitně uložen. Jinak je tento typ podobný jako jednoduchá či dvojnásobná přesnost. Některé jiné počítače používají 128 bitů. V tabulce 2 je porovnání přesností různých typů zobrazení reálných čísel.

Tabulka 2.

Typ	Přesnost zobrazení
Single	$p = 24$
Double	$p = 53$
Double-extended (Intel)	$p = 64$

Norma ANSI/IEEE ale přesto nezaručuje přenositelnost programů mezi různými počítači, protože nedefinuje pořadí bytů. Každé 32bitové počítačové slovo je složeno ze 4 po sobě jdoucích bytů B_1, B_2, B_3, B_4. Počítače typu IBM a Sun užívají adresování typu Big Endian, tj. byty jsou v pořadí B_1, B_2, B_3, B_4, zatímco Intel pracuje s typem Little Endian, kde jsou byty v pořadí B_4, B_3, B_2, B_1.

3. Zaokrouhlování

Při provádění numerických výpočtů i při definici počítačových aritmetických operací (srv. dále odst. 4) potřebujeme reálným číslům přiřadit počítačová čísla typu real. Toto zobrazení se nazývá *zaokrouhlování*.

Norma ANSI/IEEE definuje čtyři způsoby zaokrouhlování: zaokrouhlování dolů, nahoru, k nule a k nejbližšímu počítačovému číslu.

Popíšeme jen poslední způsob, protože je nejrozšířenější. Tak pracují např. osobní počítače. Vyjádříme proto každé nenulové reálné číslo ve tvaru

$$x = \operatorname{sgn}(x) \cdot \left(1 + \sum_{i=1}^{\infty} a_i 2^{-i}\right) \cdot 2^E, \quad \text{kde} \;\; E \in \mathbb{Z}, \;\; a_i \in \{0, 1\}$$

a všechna a_i nejsou od některého indexu rovna jedné.

Takové vyjádření existuje vždy a je jediné.

Budeme říkat, že reálné číslo x je v *normalizovaném rozsahu* počítačových čísel, jestliže platí

$$N_{\min} \leqq |x| \leqq N_{\max},$$

kde $N_{\min}$ je nejmenší kladné počítačové číslo a $N_{\max}$ je největší počítačové číslo.

Není-li číslo x počítačové číslo, pak buď leží mimo normalizovaný rozsah (jeho zaokrouhlení se provádí výjimečným způsobem), nebo binární rozvoj jeho mantisy vyžaduje více než p bitů.

Zaokrouhlování k nejbližšímu číslu je podle normy ANSI/IEEE dáno následujícím předpisem:

Buď $E_{\min}$, E, $E_{\max} \in \mathbb{Z}$ a $p \in \mathbb{N}$ přesnost systému čísel v pohyblivé řádové čárce. Buď $k \in \mathbb{N}$, $k > p$ nebo $k = \infty$. Předpokládejme, že $x \neq 0$ je vyjádřeno jako

$$x = \operatorname{sgn}(x) \cdot \left(1 + \sum_{i=1}^{k} a_i 2^{-i}\right) \cdot 2^E, \quad \text{kde} \;\; E_{\min} \leqq E \leqq E_{\max}, \;\; a_i \in \{0, 1\}.$$

Pak definujeme

$$[x]_R = \begin{cases} \operatorname{sgn}(x) \cdot \left(1 + \sum_{i=1}^{p-1} a_i 2^{-i}\right) \cdot 2^E & \text{pro} \;\; a_p = 0, \\ \operatorname{sgn}(x) \cdot \left(1 + 2^{-(p-1)} + \sum_{i=1}^{p-1} a_i 2^{-i}\right) \cdot 2^E & \text{pro} \;\; a_p = 1. \end{cases}$$

Jsou-li ve druhém případě všechna a_i pro $i > p$ rovna nule, zaokrouhlíme tak, aby výsledek měl v posledním bitu mantisy nulu.

Položíme $[0]_R = 0$. Pak $[x]_R$ se nazývá *hodnota čísla x správně zaokrouhlená k nejbližšímu počítačovému číslu*.

Je vhodné přesnost zaokrouhlování posuzovat podle relativní chyby. Mějme tedy $x \in \mathbb{R}$ v normalizovaném rozsahu, a $[x]_R$ jeho zaokrouhlenou hodnotu. Pak se číslo $[x]_R - x$ nazývá *absolutní chyba* čísla x. Jestliže $x \neq 0$, pak se zlomek $([x]_R - x)/x$ nazývá *relativní chyba čísla x*. Relativní chyba při uvedeném způsobu zaokrouhlování splňuje, jak se snadno ukáže, $|([x]_R - x)/x| \leqq 2^{-p}$. Je-li p přesnost počítačových čísel v pohyblivé řádové čárce, pak se číslo $\gamma = 2^{-p}$ nazývá *zaokrouhlovací jednotka*. Je vidět, že je $\gamma = \frac{1}{2}\varepsilon$, kde ε je strojové epsilon.

Platí pak tento základní odhad:

Věta. *Buď x reálné číslo v normalizovaném rozsahu počítačových čísel přesnosti p. Pak*

$$[x]_R = x(1+\delta), \tag{5}$$

kde

$$|\delta| \leqq \gamma.$$

Absolutní hodnota relativní chyby při zaokrouhlení je tedy nejvýše rovna zaokrouhlovací jednotce.

4. Aritmetické operace

Protože počítač pracuje jen s počítačovými čísly, musí být výsledek aritmetických operací provedených na počítači opět počítačové číslo. V převážné většině případů však výsledek aritmetické operace v tělese reálných čísel není počítačové číslo. Např. $1.$ a 2^{-24} jsou počítačová čísla v jednoduché přesnosti, ale jejich součet $1. + 2^{-24}$ není počítačové číslo.

Je užitečné rozlišovat mezi počítačovými operacemi a přesnými operacemi, tj. operacemi v tělese reálných čísel. Označíme čtyři základní aritmetické operace v tělese reálných čísel symboly $+$, $-$, $\times$, $:$ (znaménko násobení budeme někdy vynechávat) a symboly $\oplus$, $\ominus$, $\otimes$, $\oslash$ označíme příslušné počítačové operace.

Symbolem $*$ označíme libovolnou ze čtyř základních aritmetických operací a symbolem $\circledast$ příslušnou počítačovou operaci.

Podle normy ANSI/IEEE musí počítačové operace splňovat

$$x \circledast y = [x * y]_R. \tag{6}$$

Nebudeme probírat, jak procesor splní tento požadavek. Poznamenejme jen, že počítačová operace není definována nejen při dělení nulou, ale i při přeplnění a v některých dalších případech.

Podle předešlé věty platí:

Věta. *Je-li počítačová operace $x \circledast y$ definována, pak existuje číslo δ takové, že platí*

$$x \circledast y = (x * y)(1+\delta), \tag{7}$$

kde $|\delta| \leqq \gamma$, γ je zaokrouhlovací jednotka.

Číslo δ je relativní chyba zaokrouhlení. Rovnost (7) se zapisuje také ve tvaru $x \circledast y = (x * y)(1+\beta\gamma)$, kde $|\beta| \leqq 1$, aby se ukázala souvislost se zaokrouhlovací jednotkou. Výraz $1 + \delta = 1 + \beta\gamma$ se nazývá *opravný faktor*.

Počítačová aritmetika nezachovává vlastnosti přesných operací. Ani počítačové sčítání, ani počítačové násobení není asociativní. Pořadí operací je potřeba vyznačit závorkami. Ukážeme to na jednoduchých příkladech.

Příklad 1. Je $\left(1. \oplus 2^{-24}\right) \ominus 2^{-24} = 1. - \gamma \neq 1.$. Při uzávorkování druhého a třetího členu dostaneme ovšem jedničku.

Příklad 2. Existují čísla x, pro která platí $(1. \odot x) \otimes x = 1. - \gamma \neq 1.$. Z celých čísel jsou to např. $x = 41., 47., 55., 61.$.

Příklad 3. Existují čísla x, pro která platí $1. \odot (1. \odot x) \neq x$. Z celých čísel jsou to např. $x = 7., 13., 14., 15.$.

Rovnost (7) můžeme upravit při operaci násobení na tvar

$$x \otimes y = x(1+\delta) \times y \quad \text{nebo} \quad x \otimes y = x \times y(1+\delta) \tag{8}$$

a při operaci sčítání na tvar

$$x \oplus y = x(1+\delta) + y(1+\delta), \tag{9}$$

a pak tyto rovnosti interpretovat tak, že počítačová operace s danými počítačovými čísly dá stejný výsledek jako přesná operace s čísly, jejichž relativní odchylka nepřevyšuje zaokrouhlovací jednotku γ.

Vzniká zde otázka, zda uvedený odhad je dosažitelný. Je zřejmé, že zaokrouhlovací chyba bude maximální, jestliže mantisa výsledku bude jednička nebo číslo jí blízké a zaokrouhlení bude velikosti γ. Na exponentu nezáleží, neboť chyba je relativní. Uveďme jednoduché příklady pro jednoduchou aritmetiku. Zaokrouhlovací jednotka pro jednoduchou přesnost je $\gamma = 2^{-24} \doteq 5.960464477539063 \cdot 10^{-8}$.

Příklad 4. Máme dvě počítačová čísla

$$x = 1. + 2^{-22} \doteq 1.0000002384, \quad y = 1. - 2^{-24} \doteq 0.9999999404.$$

Výsledek výpočtu součinu $x \times y$ na počítači je $x \otimes y = 1. + 2^{-23}$ a platí $x \otimes y = (x \times y)(1 - \beta\gamma)$, kde $\beta \doteq 0.999996$.

Provedeme-li více operací násobení za sebou, dostaneme

$$x_0 \otimes \cdots \otimes x_{k-1} = (x_0 \times \cdots \times x_{k-1})(1+\delta_1)\cdots(1+\delta_{k-1}).$$

Protože počítačové násobení není asociativní, znamená použitý zápis, že operace provádíme zleva doprava.

Je-li $(1+\Delta) = (1+\delta_1)\cdots(1+\delta_{k-1})$, pak Δ je relativní chyba součinu k čísel a chceme zjistit, jak velká může být. Ilustrujme to na dalším příkladu.

Příklad 5. Zvolme k čísel

$$x_0 = 1. + k2^{-23}, \quad x_1 = \cdots = x_{k-1} = 1. - 2^{-24}.$$

Pro počítačové násobení platí $(1. + k2^{-23}) \otimes (1. - 2^{-24}) = 1. + (k-1)2^{-23}$. Zvolíme-li $k = 16$, je

$$x_0 \doteq 1.000001907$$

a postupně dostáváme

$$\begin{aligned} x_0 \otimes x_1 &= (x_0 \times x_1)(1 - 0.9999962\gamma), \\ x_0 \otimes x_1 \otimes x_2 &= (x_0 \times x_1 \times x_2)(1 - 1.9999927\gamma), \\ &\cdots \\ x_0 \otimes x_1 \cdots \otimes x_{15} &= (x_0 \times x_1 \cdots \times x_{15})(1 - 14.9999624\gamma). \end{aligned}$$

Uvedené hodnoty koeficientů β jsou zaokrouhlená čísla. Relativní chyby jednotlivých operací jsou zde blízké zaokrouhlovací jednotce a jsou téhož znaménka. Je patrné, že se při násobení (jak známo) sčítají. Celkové relativní chyby dosahují skoro maximální teoretické hodnoty. Podobný příklad lze snadno najít i pro dělení.

Tento příklad je poněkud vykonstruovaný. Při praktickém výpočtu bude s velkou pravděpodobností situace příznivější, protože mezivýsledky nebudou v těsné blízkosti mocnin dvojky a chyby budou mít různé znaménko.

Jestliže se budou operace násobení (a dělení) střídat s operacemi sčítání (a odčítání), bude chování relativní chyby i její analýza mnohem složitější.

5. Závěrečné poznámky

Souvislost počítačových a přesných operací (8) a (9) je základem zpětné analýzy zaokrouhlovacích chyb, kterou v padesátých a šedesátých létech systematicky vybudoval J. H. Wilkinson. Jeho analýza ukazuje, že výsledek počítačového výpočtu se rovná výsledku přesného výpočtu, který vyjde z jiných, fiktivních vstupních dat. Chceme, aby se tato fiktivní data málo lišila od daných dat a pro jejich rozdíl získáváme příslušné odhady (viz [16]).

Wilkinsonova zpětná analýza vlivu zaokrouhlovacích chyb také umožňuje oddělit ve výsledné chybě vliv zaokrouhlovacích chyb způsobených použitým numerickým algoritmem a vliv citlivosti vlastní matematické úlohy na zaokrouhlovací chyby.

Řada varovných příkladů ilustrujících nežádoucí vliv zaokrouhlovacích chyb je v [7]. Analýza citlivosti na zaokrouhlovací chyby je složitý problém a v literatuře se mu věnuje velká pozornost.

Vliv zaokrouhlovacích chyb lze zmírnit tím, že numerické výpočty budeme provádět s čísly ve dvojnásobné přesnosti. Spotřeba počítačového času je prakticky stejná jako při jednoduché přesnosti. Jsou ovšem situace, kdy ani to nestačí a je nutno použít aritmetiku s libovolnou volitelnou přesností, jak ji nabízejí systémy Mathematica nebo

Maple, alespoň pro část výpočtu. Ve Fortranu je taková aritmetika implementována v souboru FMLIB, viz [10], [12]. Čas výpočtu se při tom značně prodlouží.

Norma ANSI/IEEE pamatuje také na některé další operace prováděné při počítání, které mohou být zdrojem zaokrouhlovacích chyb. Jsou to operace vstupu a výstupu, to je vlastně převod čísel z desítkové soustavy do dvojkové a naopak. Dále jsou to převody z jednoho typu počítačových čísel na jiný. Tyto operace jsou nejen zdrojem zaokrouhlovacích chyb, ale může při nich dojít k překročení číselného rozsahu jednotlivých typů čísel a ke zhroucení výpočtu.

Používají se ovšem i operace nearitmetické, nezahrnuté do normy ANSI/IEEE, jako je např. rozhodování na základě nerovností. I tady je namístě velká opatrnost. Rozhodování podle znaménka výsledku, který má malou abolutní hodnotu a je ovlivněn zaokrouhlovací chybou, je totiž velmi problematické.

Závěrem uveďme příklady situací, kdy zanedbání vlivu zaokrouhlování (nebo jiných operací s počítačovými čísly) vedlo k vážným haváriím.

První je kolaps řízení obranného protiraketového systému Patriot ve válce v Zálivu v r. 1991. Systém fungoval efektivně, ale v jednom případě selhal.

Identifikace střel Scud probíhala tak, že radarový systém po zachycení podezřelého objektu předpověděl jeho budoucí polohu po uplynutí nějakého časového intervalu, a jestliže v tom místě v příslušnou dobu objekt zjistil, dal signál k jeho zničení. A právě v předpovědi nové polohy docházelo k nepřesnostem, a to tím větším, čím déle systém pracoval. Po 20 hodinách byla chyba v určení polohy zhruba 137 metrů, po 100 hodinách 687 metrů.

Prvotní příčinou problému bylo to, že vnitřní hodiny počítače systému uchovávaly čas v celých násobcích desetiny sekundy a program je převáděl na číslo v pohyblivé čárce v sekundách. Při tom se dekadické číslo 0,1 , které má ve dvojkové soustavě nekonečný rozvoj 0,0001100110011..., zaokrouhlovalo.

V pramenech [3], [11], [15], které se zabývají analýzou selhání, není výpočet přesně popsán. Zaokrouhlovací chyba se zřejmě násobila koeficientem úměrným době funkce systému. Restart systému totiž problém odstranil.

Mnoho rozruchu udělala chyba v původním mikroprocesoru Pentium. Dělení $\frac{4195835}{3145727}$ v pohyblivé čárce bylo správné jen na 4 cifry! Chybu objevili matematici experimentující v teorii čísel. Původně zdráhavý postoj firmy vedl nakonec až k poklesu hodnot jejích akcií. Poté byly všechny vadné mikroprocesory vyměněny (viz [1]).

Převod čísla v pohyblivé čárce na celé číslo způsobil destrukci rakety Ariane 5 v ceně zhruba jedné miliardy dolarů v červnu 1996 (srv. [14]). Třicet sedm vteřin po startu se program pokusil konvertovat horizontální rychlost rakety z dvojnásobné přesnosti na krátké (16bitové) celé číslo. Číslo bylo pro tento formát příliš velké. Počítačový program ohlásil neplatnou operaci a řídicí systém provedl samodestrukci rakety.

Podobná chyba v převodu čísla v pohyblivé čárce na celé číslo se objevila při chybě v procesorech Pentium Pro a Pentium II v dubnu 1997. Zde to byla ovšem chyba hardwaru. Nebyl nastaven příslušný příznak (angl. flag), ačkoli podle normy ANSI/IEEE měl být. Tentokrát společnost Intel reagovala profesionálně a chybu brzy napravila.

L i t e r a t u r a

[1] Edelman, A.: *The mathematics of the Pentium division bug.* SIAM Review *39* (1997), 54–67.

[2] Hämmerlin, G., Hoffmann, K.- H.: *Numerical Mathematics.* Springer-Verlag, Inc., New York 1991.

[3] *GAO Report: Patriot Missile Defense.* `http://www.fas.org/starwars/gao/im92026.htm`

[4] Goldberg, D.: *Computer Arithmetic.* Kaufmann, San Mateo, CA 1995, second edition. Appendix in [9].

[5] IEEE standard for binary floating-point arithmetic: ANSI/IEEE Std 754-1985, 1985. Reprinted in SIGPLAN Notices *22* (2) (1987), 9–25.

[6] IEEE standard for radix-independent floating-point arithmetic: ANSI/IEEE Std 854--1987, 1987.

[7] Křížek, M., Práger, M., Vitásek, E.: *Spolehlivost numerických výpočtů.* PMFA *42* (1997), 8–23.

[8] Overton, M. L.: *Numerical Computing with IEEE Floating Point Arithmetic.* SIAM, Philadelphia 2001.

[9] Patterson, D. L., Hennessy, J. L.: *Computer Architecture: A Quantitative Approach.* Kaufmann, San Mateo, CA 1998, second edition.

[10] Priest, D. M.: *Algorithms for arbitrary precision floating point arithmetic.* In: P. Kornerup and D. Matula, editors, *Proceedings of the 10th Symposium on Computer Arithmetic*, 132–143. IEEE Computer Society Press, Piscataway, NJ, 1991.

[11] Skeel, R.: *Roundoff error and the Patriot missile.* SIAM News *25* (4), July 1992, 11. `http:/www.siam.org/siamnews/general/patriot.htm`

[12] Smith, D. M.: *Multiple Precision Complex Arithmetic and Functions.* TOMS *24* (1998), 4, též `http://www.netlib.org/toms/814`

[13] Sterbenz, P.: *Floating Point Computation.* Prentice–Hall, Englewood Cliffs, NJ, 1974.

[14] *The Explosion of Ariane 5.* `http://www.ima.umn.edu/~arnold/disasters/ariane.html`

[15] *The Patriot Missile Failure.* `http://www.ima.umn.edu/~arnold/disasters/patriot.html`

[16] Wilkinson, J. H.: *Rounding Errors in Algebraic Processes.* Her Majesty's Stationery Office, London 1963.