Úvod do organizace počítače

Pohyblivá řádová čárka
(Floating Point)

Konvence zápisu, MIPS

Literatura

Normy:

- IEEE 754
- IEEE 854

FP = Floating Point

Knihy:

Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefévre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, Serge Torres:

Handbook of Floating-Point Arithmetic

ISBN 978-0-8176-4704-9 e-ISBN 978-0-8176-4705-6

Přehled

- Čísla v pohyblivé řádové čárce
- Zápis čísel
  - Desítková notace
  - Binární notace
- Standard IEEE 754 FP
- Interní reprezentace FP čísel v počítači
  - Větší rozsah vs. přesnost zobrazení
- Konverze desítkového zápisu na FP
- Typ není asociován s daty
- FP instrukce MIPS, registry

FP = Floating Point

Čísla a počítače

- Co lze zobrazit na n bitech ?
  - Čísla integer bez znaménka: 0 až $2^n - 1$
  - Čísla integer se znaménkem: $-2^{(n-1)}$ až $2^{(n-1)} - 1$
- Ostatní čísla?
  - Velmi velká čísla? (počet částic hmoty)
    $3{,}155{,}760{,}000_{10}$ ($3.15576_{10} \times 10^9$)
  - Velmi malá čísla? (rozměry částic hmoty)
    0.00000001_{10} ($1.0_{10} \times 10^{-8}$)
  - Racionální čísla (popř. čísla s periodou)
    2/3 (0.666666666. . .)
  - Iracionální čísla: $2^{1/2}$ (1.414213562373. . .)
  - Transcendetní čísla: e (2.718...), π (3.141...)

Notace

mantisa ***exponent***

$$\mathbf{6.02 \times 10^{23}}$$

základ (báze)

desetinná tečka

- Normalizovaný tvar: nemá úvodní nuly (přesně první číslice vlevo od desetinné tečky)
- Alternativní reprezentace 1/1,000,000,000
  - –Normalizováno: 1.0×10^{-9}
  - –Nenormalizováno: 0.1×10^{-8}, 10.0×10^{-10}

Binární notace

mantisa ***exponent***

$$1.0_2 \text{ x } 2^{-1}$$

“binární tečka” ***základ (báze)***

- Aritmetika čísel FP
  - –Binární tečka nemá pevnou polohu (jako tomu bylo u čísel typu integer)
  - –V jazyce C se taková proměnná deklaruje jako float

Počítače a normalizovaná čísla

- Protože počítače pracují „pouze“ s binárními čísly, budeme je vyjadřovat pomocí normalizovaného vyjádření (scientific notation) s použitím binární tečky.
- Proč se používá právě tato forma?
  - Zjednodušuje výměnu dat, protože FP-čísla pak mají stejný tvar.
  - Zjednodušuje FP-algoritmy, protože čísla jsou vždy v této formě.
  - Zvyšuje se přesnost zobrazení, protože se nezobrazují nevýznamné úvodní nuly, naopak, vytváří se prostor pro cifry napravo od binární tečky.

Standard IEEE 754 FP

- Používán téměř ve všech počítačích (od r. 1980)
  - Přenositelnost FP programů
  - Kvalita počítačové aritmetiky FP čísel
- Znaménkový bit:
  - 1 znamená záporné číslo
  - 0 znamená kladné číslo
- Mantisa:
  - Prvá 1 je u normalizovaných čísel implicitní
  - (1 + 23) bitů jednoduchá, (1 + 52) bitů dvojitá
  - vždy platí: $0 < \text{Mantisa} < 1$
- 0.0 do pravidla nezapadá, má zvláštní vyjádření

$$\boxed{(-1)^{S} * (1 + \text{Mantisa}) * 2^{\text{Exp}}}$$

Exponent v normě IEEE 754

- Operovat s FP čísly lze i bez FP hardware
  - Setřídění FP čísel použitím komparace pro čísla typu integer!
- Rozdělit FP číslo na 3 složky: porovnat znaménka, pak exponenty a nakonec mantisy
- Rychlejší (v ideálním případě, jednoduchá komparace při vhodném rozložení ve slově)
  - Nejvyšší bit je znaménko (záporné < kladné)
  - Následuje exponent, větší exponent => větší #
  - Nakonec mantisa: stejný exponent => větší # má větší mantisu

Exponent – kód s posunutou nulou

- Dvojkový komplement je pro exponent nefunkční
- Nejmenší exponent: 00000001_2
- Největší exponent: 11111110_2
- Posun: číslo, přičtené k reálnému exponentu
  - 127 pro jednoduchou přesnost
  - 1023 pro dvojitou přesnost
- $1.0 * 2^{-1}$

0	0111 1110	0000 0000 0000 0000 0000 000

$$\boxed{(-1)^S * (1 + \text{Mantisa}) * 2^{(\text{Exponent - Posun})}}$$

Převod z binárního do desítkového tvaru FP

0	0110 1000	101 0101 0100 0011 0100 0010

- Znaménko: 0 => kladné
- Exponent:
  - $0110\ 1000_2 = 104_{10}$
  - Výpočet posunu: 104 - 127 = -23
- Mantisa:
  - ❖ $1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5} + \ldots$
    $= 1+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}$
    $= 1.0 + 0.666115$
- Vyjadřuje: $1.666115*2^{-23} \sim 1.986*10^{-7}$

Implicitní část mantisy

Převod z desítkového do bin. tvaru FP (1/2)

- Jednoduchý případ: Je-li jmenovatel mocninou 2 (2, 4, 8, 16, atd.), je to snadné.
- Binární FP reprezentace čísla -0.75
  - -0.75 = -3/4
  - $-11_2/100_2 = -0.11_2$
  - Normalizováno na $-1.1_2 \times 2^{-1}$
  - $(-1)^S \times (1 + \text{Mantisa}) \times 2^{(\text{Exponent}-127)}$
  - $(-1)^1 \times (1 + .100\ 0000\ ...\ 0000) \times 2^{(126-127)}$

1	**0111 1110**	**100 0000 0000 0000 0000 0000**

Převod z desítkového do bin. tvaru FP (2/2)

- Jmenovatel není mocninou 2
  - Číslo nelze reprezentovat přesně
  - Mantisa má obvykle dost bitů na dosažení požadované přesnosti
  - Obtížnější krok: výpočet mantisy
- Racionální čísla mají periodu
- Převod
  - Zapište binární číslo s opakující se periodou.
  - Bity přesahující mantisu vpravo ořízněte (různý počet pro jednoduchou vs. dvojitou přesnost).
  - Odvoďte znaménko a pole exponentu a mantisy.

Převod z desítkového do binárního tvaru

$\boxed{-\ 3\ .\ 3\ 3\ 3\ 3\ 3\ 3\ldots}$

0.33333333	0.66666666	0.33333332
x 2	x 2	x 2
0 .66666666	1 .33333332	0 .66666664

- 1 1 . 0 1 0 1 0 1 0 . . . **=> - 1.1010101.. x 2^1**

- Mantisa: 101 0101 0101 0101 0101 0101
- Znaménko: záporné => 1
- Exponent: 1+ 127 = 128_{10} = $1000\ 0000_2$

1	1000 0000	101 0101 0101 0101 0101 0101

Hlediska návrhu formátu

- Pro uložení FP-čísla musíme uložit následující tři složky informace …
  - Znaménko (sign) kladné/záporné
  - Exponent
  - Mantisa
- Je-li dán pevný počet bitů pro uložení čísla (např. slovo), jak zvolit velikost pole pro mantisu a pro exponent?
  - Zvětšováním mantisy roste přesnost zobrazení.
  - Zvětšováním exponentu narůstá rozsah zobrazovaných čísel.
  - **Jde o kompromis** (ostatně jako u mnoha dalších podobných rozhodnutí).

Standardy IEEE 754 (Floating Point)

- IEEE respektoval volby návrhu a doporučil velikost exponentu 8 bitů a 23 bitů pro mantisu (za předpokladu, že délka slova je 32 bitů).
- Tento formát je použit u MIPS a u většiny počítačů po roce 1980 – jedná se dobré kompromisní řešení.

s	exponent	mantisa
1 bit	8 bitů	23 bitů

Reprezentované číslo = $(-1)^S \times F \times 2^E$

kde S, F a E jsou pole znaménka, exponentu a mantisy

(1 v poli s znamená zápornou hodnotu čísla)

FP výjimky

- Standard IEEE 754 pokrývá velmi velký rozsah reálných čísel, která mohou být vyjádřena, od nejmenších 2.0×10^{-38} k největším 2.0×10^{38}.
- ! Je třeba podotknout, že rozsah je velký, ale nikoliv nekonečný…
  - Přetečení v pohyblivé řádové čárce nastává, jestliže vypočtený exponent výsledku je příliš velký a nelze ho vyjádřit v poli exponentu (příliš velké číslo). ($> 2.0 \times 10^{38}$)
  - Podtečení v pohyblivé řádové čárce nastává, jestliže vypočtený exponent výsledku je příliš malý a nelze ho vyjádřit v poli exponentu (příliš malé číslo – co do abs. hodnoty (>0, $< 2.0 \times 10^{-38}$)

Dvojitá přesnost

- Aby se bylo možno s těmito případy lépe vyrovnat IEEE 754 standard zahrnuje specifikaci formátu *double precision*, ve které jsou použita dvě slova k zobrazení čísla.
- Exponent je rozšířen na 11 bitů a mantisa na 52 bitů…

s	exponent	mantisa
1 bit	11 bitů	20 bitů

mantisa (pokračování)
32 bitů

- V jazyce C proměnná deklarována jako **double**
- Reprezentuje čísla v rozsahu od nejmenšího 2.0 x 10^{-308} až po největší 2.0 x 10^{308}
- Primární výhodou je větší přesnost (52 bitů) (přesnost určuje mantisa !)

Výhody dvojité přesnosti

- Tento formát dovoluje vyjádřit čísla ve větším rozsahu a to od 2.0×10^{-308} do 2.0×10^{308}.
- Přestože primárním důvodem rozšíření je podstatné zvýšení přesnosti zobrazení, bylo zvětšeno i pole pro zobrazení exponentu.

Optimalizace

- Protože bit nalevo od binární tečky je trvale „1“ a nenese proto žádnou informaci, rozhodli se návrháři normy IEEE 754 tento bit nezahrnout do standardního formátu.
- Čísla IEEE 754 mají mantisu o délce 24 bitů (1 implicitní a 23 ukládaných) pro jednoduchou přesnost a 53 bitů mantisy (1 implicitní a 52 ukládaných) ve dvojité přesnosti.
- Nula je zobrazena speciálním způsobem a to s nulou v exponentu, v mantise i ve znaménku.

Další optimalizace…

- Mantisa využívá „skrytou“ jedničku, hodnota čísla je pak rovna:

  $$(-1)^S \times (1 + \text{mantisa}) \times 2^E$$

  kde bity mantisy představují zlomek mezi nulou a jedničkou.
- Pro zjednodušení a zrychlení komparace čísel bylo vhodně zvoleno i pořadí jednotlivých polí v zobrazení čísla.
  – To je hlavní důvod proč znaménkový bit leží v MSB.

Optimalizace porovnání

- Exponent leží vlevo od mantisy – to ulehčuje porovnání, které lze provést pomocí integer operace.
  - Není to tak snadné jako v případě čísel v doplňkovém kódu, protože je třeba vyšetřit znaménkový bit a amplitudu exponentu.
  - Tento postup je korektní, pokud jsou oba exponenty kladné. Jak tomu bude v případě záporných exponentů? Jak mají být kódovány? Uvědomte si, že je třeba jednoduše porovnat hodnoty dvou exponentů abychom určili jejich vzájemný vztah.

Kódování exponentu

- Kdybychom zakódovali exponenty v doplňkovém kódu, záporný exponent by se jevil jako velké kladné číslo (jednička v MSB).
- Například, zakódujeme-li 1.0 x 2^{-1} a 1.0 x 2^{1} s použitím doplňkového kódu pro exponent, dostaneme…

Toto číslo je menší !!!

0	1111 1111	0000 0000 0000 0000 0000 000
S	exponent	mantisa

0	0000 0001	0000 0000 0000 0000 0000 000
S	exponent	mantisa

Kódování exponentu

- Jako vhodná forma pro exponent se jeví takové zobrazení, u kterého je nejmenší (záporný) exponent zobrazen jako 00..00 a největší kladný exponet jako 11..11.
- Tato konvence se nazývá *kód s posunutou nulou (biased encoding)* – tento posun je přičten bez znaménka k exponentu. Tak je získán obsah pole exponentu.
- Standard IEEE 754 používá posun *(bias)* 127.
- Proto skutečný exponent –1 je kódován jako (-1)+127=126 (0111 1110) a exponent 1 je kódován jako 1+127=128 (1000 0000).

Kód s posunutou nulou v IEEE 754

- Z toho vyplývá, že hodnotu čísla kódovaného podle normy IEEE 754 určíme podle výrazu…

$$(-1)^S \times (1 + \text{mantisa}) \times 2^{(\text{exponent} - \text{bias})}$$

- Stejný výraz platí i pro dvojnásobnou přesnost, pouze s tím rozdílem, že posun je pak roven1023. (00..00) je opět nejmenší exponent, a (11..11) představuje největší možný exponent.

Příklad: dekódování IEEE 754

- Zakódujeme $(–0.75)_{10}$ podle IEEE 754.
- Zápis ve dvojkové soustavě…$(–0.11)_2$.
- Normalizovaná forma…$(-1.1)_2 \times 2^{-1}$.
- Požadovaný tvar…

$$(-1)^S \times (1 + \text{mantisa}) \times 2^{(\text{exponent} - \text{bias})}$$

- Převod do požadovaného tvaru…

$$(-1)^1 \times (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 0000) \times 2^{(126 - 127)}$$

- Pro jednoduchou přesnost podle IEEE 754…

1	0111 1110	1000 0000 0000 0000 0000 000
S	exponent	mantisa

- Pro dvojnásobnou přesnost podle IEEE 754…

0	0111 1111 110	1000 0000 0000 0000 0000 000
S	exponent	mantisa

0000 0000 0000 0000 0000 0000 0000 0000

Příklad: dekódování IEEE 754

- Budeme dekódovat číslo podle IEEE 754…

1	1000 0001	0100 0000 0000 0000 0000 000
S	exponent	mantisa

- Pro reprezentaci čísla platí výraz…

$$(-1)^S \times (1 + \text{mantisa}) \times 2^{(\text{exponent} - \text{bias})}$$

- Dosazením hodnot polí…

$$(-1)^1 \times (1 + 0.25) \times 2^{(129 - 127)}$$

- Vyčíslením výrazu…

$$-1 \times 1.25 \times 2^2 = -1.25 \times 4 = -5.0$$

- Uvedený obsah polí tedy vyjadřuje $-(5.0)_{10}$.

Sčítání FP čísel

- Nyní když víme, jak se čísla v pohyblivé řádové čárce zobrazují, můžeme s nimi provádět operace – např. sčítání.
- Nejlépe lze porozumět této operaci tak, že ji sami krok po kroku vyzkoušíme.
- Pak se můžeme pokusit přidat hardware k ALU, který tyto kroky bude provádět podobně, jako jsme je dělali v předchozím případě „ručně“.
- Sečteme krok po kroku čísla $9.999 \times 10^{1} + 1.610 \times 10^{-1}$ (pro přehlednost použijeme desítkovou soustavu, ve dvojkové by operace probíhaly stejně).

Příklad - použité zjednodušení

- Zobrazení FP čísel v počítačích má pevnou délku.
- Pro jednoduchost budeme uvažovat formát, který používá 4 dekadické cifry pro mantisu a 2 dekadické cifry pro exponent.
- Stejný princip lze použít i na čísla podle standardu IEEE 754, uvedené zjednodušení je použito kvůli ilustraci procesu sčítání a ilustraci kompromisů s ohledem na omezenou délku zobrazení.

Sčítání FP čísel

Krok 1: Vyrovnání exponentů

- Abychom správně sečetli čísla, je nutné upravit polohu desetinné tečky jednoho z operandů (abychom sčítali cifry se stejnou vahou).
- V našem případě upravujeme exponent čísla 1.610×10^{-1} tak, aby odpovídal exponentu čísla 9.999×10^{1}.
- $1.610 \times 10^{-1} = 0.1610 \times 10^{0} = 0.01610 \times 10^{1}$
- Nezapomeňte, že můžeme ukládat pouze 4 cifry mantisy, takže dostaneme hodnotu 0.016×10^{1} (ztratili jsme na přesnosti vlivem omezení HW prostředků – délka zobrazení).

Sčítání FP čísel

Krok 2: Sečtení mantis

- Potom, co byly srovnány exponenty, můžeme provést operaci součtu mantis…

```
     9.999
+    0.016
   -------
    10.015
```

- Součtem dostáváme výsledek 10.015×10^1.

Sčítání FP čísel

Krok 3: Normalizace součtu

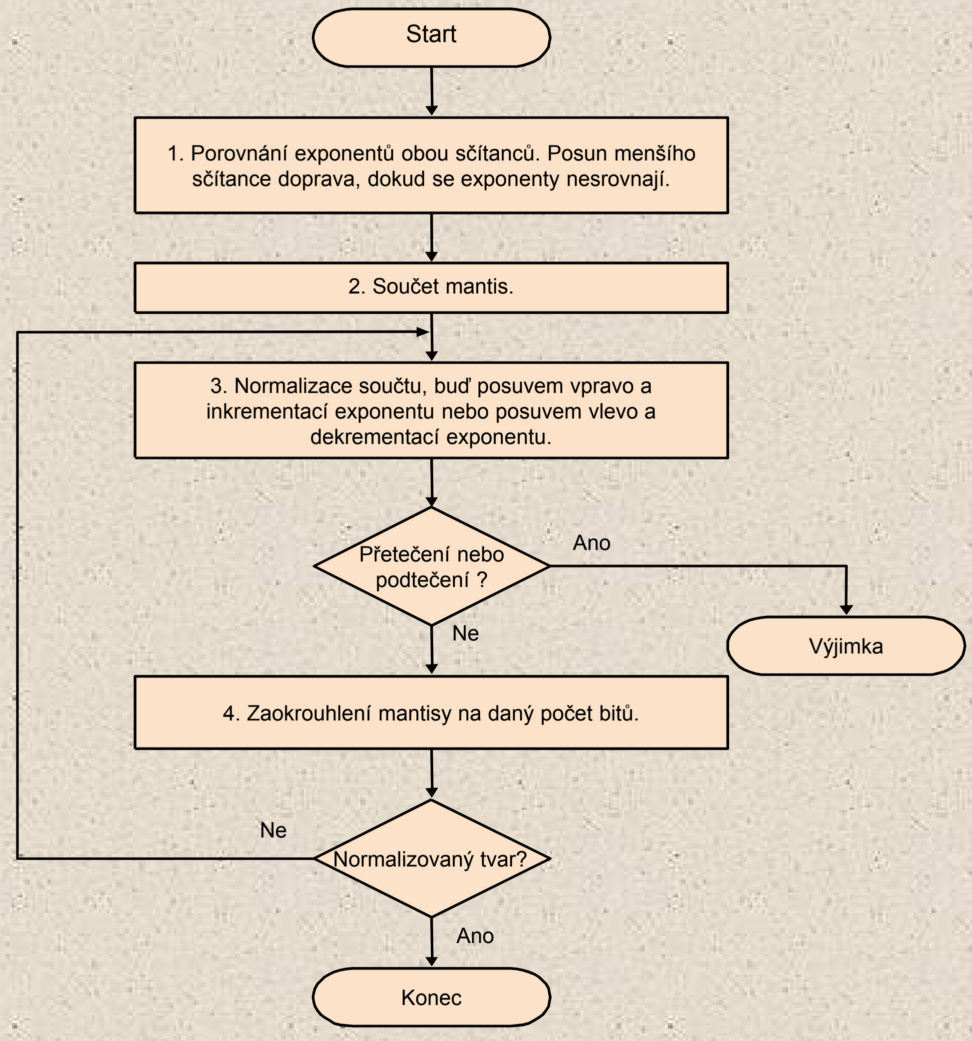
- Nakonec provedeme normalizaci součtu – převedení do standardního tvaru, který byl operací součtu porušen.
- $10.015 \times 10^1 = 1.0015 \times 10^2$
- Nezapomeňte, že i zde je nutné provést kontrolu, zda nenastalo přetečení nebo podtečení. V tomto případě k chybám nedošlo, exponent výsledku je roven 2 a lze ho zobrazit.

Sčítání FP čísel

Krok 4: Zaokrouhlení

- Při sčítání vznikl výsledek, který překračuje nároky na délku zobrazení => musíme výsledek zaokrouhlit.
- Použijeme staré zaokrouhlovací pravidlo ze základní školy, 1.0015×10^2 zaokrouhlíme na 1.002×10^2.
- Nutno poznamenat, že i zaokrouhlením lze opět dostat nenormalizované číslo a je nutno se pak vrátit ke kroku 3.

Algoritmus součtu FP-čísel



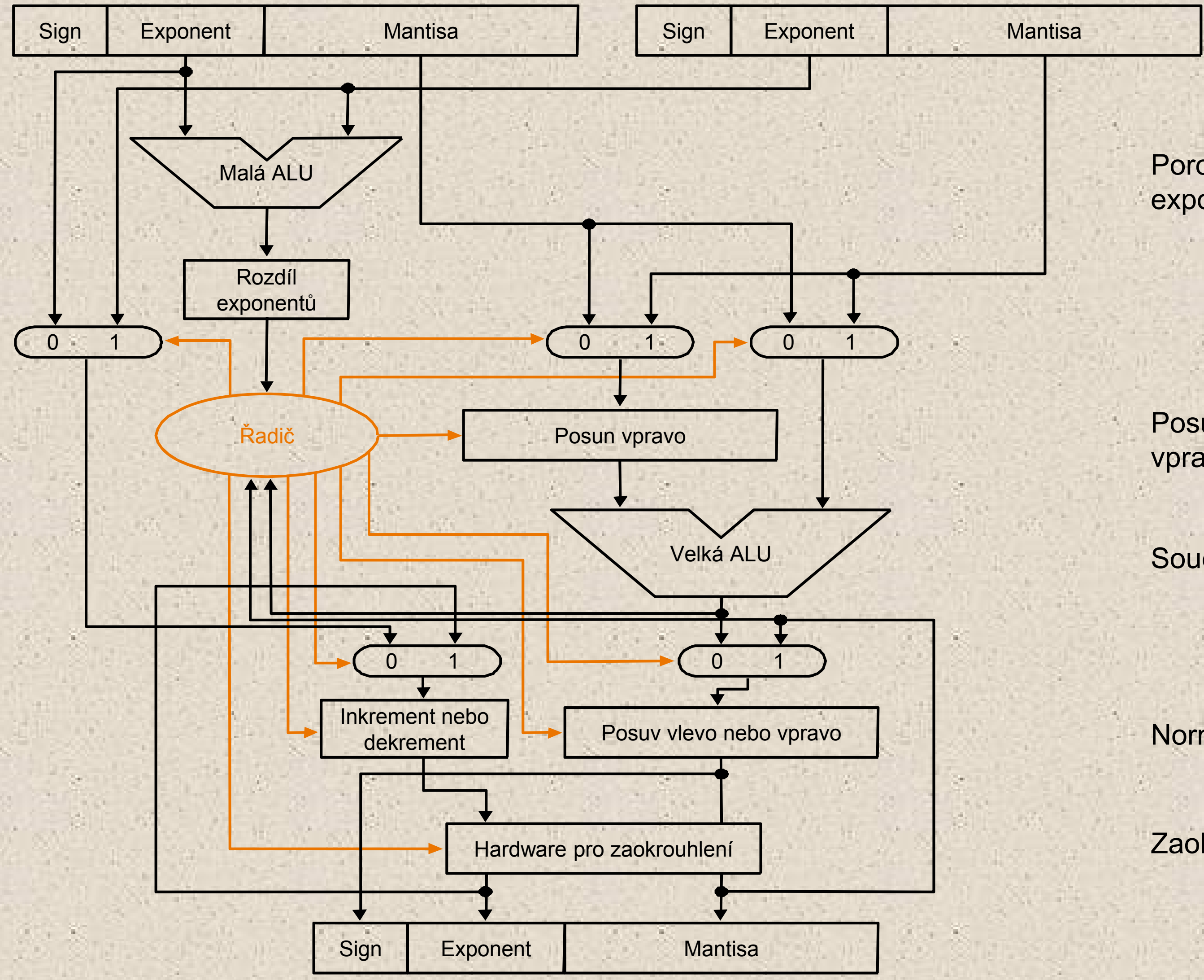
Není optimalizováno!

Jistě dokážete najít „rezervy“ tohoto algoritmu.

Hardware pro součet FP-čísel

- Moderní procesory mají často implementováno technické vybavení (hardware) pro rychlé provádění FP-operací, jako např. sčítání.
- Generický návrh takové implementace obsahuje dvě ALU, řídící jednotku, posuvný registr, mini-ALU (pro inkrement/dekrement) a obvody pro provedení zaokrouhlovacího procesu.

Hardware pro součet FP-čísel



Porovnání exponentů

Posuv menšího čísla vpravo

Součet mantis

Normalizace

Zaokrouhlení

Násobení FP čísel

- Když jsme zvládli jednoduchou operaci sčítání FP čísel, můžeme přikročit ke složitějšímu problému – násobení FP čísel.
- Podobně jako v předchozím případě budeme postupovat krok po kroku.
- Opět použijeme k zobrazení desítkovou soustavu. Mantisa bude zobrazena na 4 dekadických cifrách, exponent na 2 dekadických cifrách.
- Uvědomte si, že stejnou proceduru můžete aplikovat na binárně zobrazená čísla podle normy IEEE 754, jedná se jen o zjednodušený příklad.
- Budeme násobit čísla 1.110×10^{10} a 9.200×10^{-5}.

FP násobení

Krok 1: Sečtení exponentů

- Výpočet exponentu součinu je jednoduchý. Sečteme exponenty násobence a násobitele.
- Sečteme 10 a (-5), dostaneme 5 – exponent součinu je roven 5.
- Nyní totéž provedeme s posunutými exponenty (protože v této formě se exponenty ukládají), posun je 127.
  - (10+137) + (-5+137) = 137+122 = 259
  - To není správný výsledek: 259 - 127 = 132 a nikoliv 5.
  - Posun jsme započetli dvakrát! Proto je nutno posun odečíst: 132 – 127 = 5 (správný výsledek!).

FP násobení

- Krok 2: Násobení mantis

- Nyní vynásobíme mantisy…

```
     1.110
x    9.200
---------
     0000
    0000
   2220
  9990
---------
10212000
```

- Desetinná tečka je umístěna po šesté cifře zprava, protože násobitel i násobenec mají tři desetinná místa – součin je roven 10.212000.
- Předpokládejme, že můžeme uložit pouze tři cifry vpravo od desetinné tečky, bude součin roven 10.212 x 10^5.

FP násobení

Krok 3: Normalizace součinu

- Součin je třeba normalizovat, protože zatím nemá požadovaný normalizovaný tvar, ve kterém ho lze uložit do paměti.
- $10.212 \times 10^5 = 1.0212 \times 10^6$
- Připomeňte si, že je třeba zkontrolovat, zda nedošlo k přetečení nebo k podtečení. V tomto případě žádná z uvedených chyb nenastala.

FP násobení

Krok 4: Zaokrouhlení

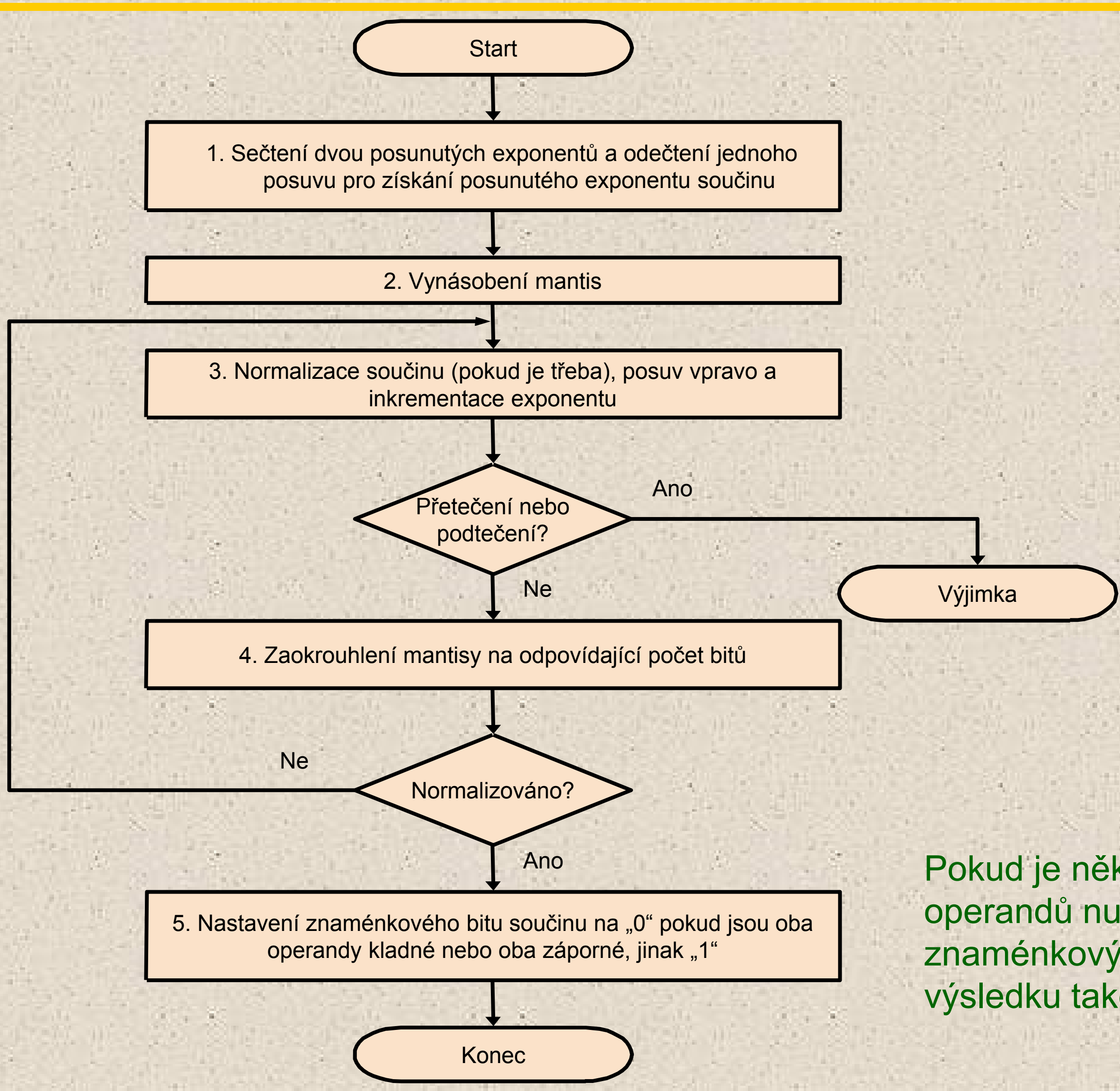
- Protože provedením operace se počet cifer zvýšil, je třeba provést zaokrouhlení výsledku.
- Použitím zaokrouhlovacích pravidel (ze základní školy) dostaneme: 1.0212×10^6 zaokrouhleno dává 1.021×10^6.
- Nakonec je opět třeba ověřit, zda zůstal zachován normalizovaný tvar stejně, jako tomu bylo v případě operace sčítání.

FP násobení

Krok 5: Určení znaménka

- Nakonec určíme znaménko součinu.
- Jsou-li znaménka obou operandů shodná, výsledek je kladný, v opačném případě záporný (násobení nulou neuvažujeme).
- V našem případě byly oba operandy kladné a proto i výsledek je kladný.
- Konečný výsledek: $+1.021 \times 10^6$.

Algoritmus násobení FP čísel

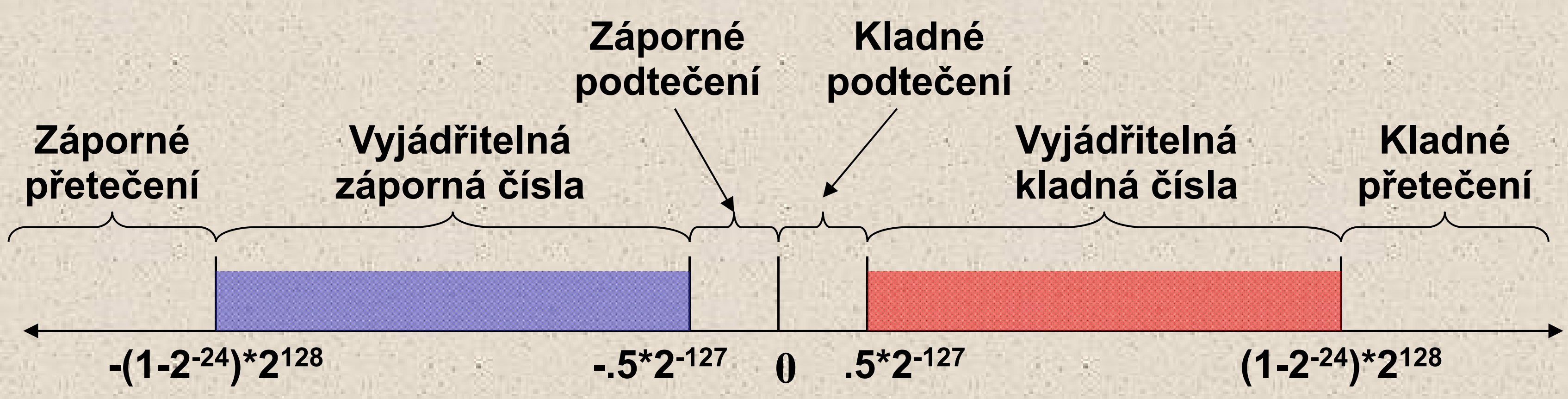


Pokud je některý z operandů nulový, je znaménkový bit výsledku také „0“

Dělení FP čísel

- Dělení FP čísel je složitá operace.
- K našemu postupnému budování hardware (metodou pokus-úspěch) zmiňme ještě některé urychlené komerční metody.
  - Newtonova iterační metoda se používá k nalezení převrácené hodnoty jednoho z operandů. Vynásobení optimalizovaným hardwarem s druhým operandem dostáváme podíl.
  - Sekvenční algoritmy (bit po bitu)
    - Binární dělení s obnovou zbytku
    - Binární dělení bez obnovy zbytku
    - SRT dělení – odhad většího počtu bitů podílu pomocí tabulek (Intel Pentium používá podobnou metodu).

Speciální hodnoty



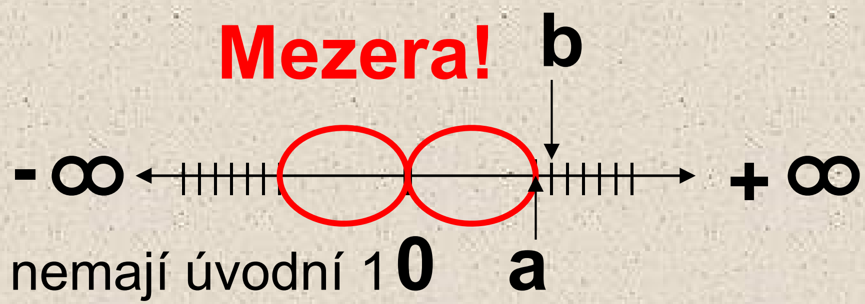
Speciální hodnoty	Exponent	Mantisa
+/- 0	0000 0000	0
denormalizované číslo	0000 0000	nenulová
NaN	1111 1111	nenulová
+/- nekonečno	1111 1111	0

Not a Number

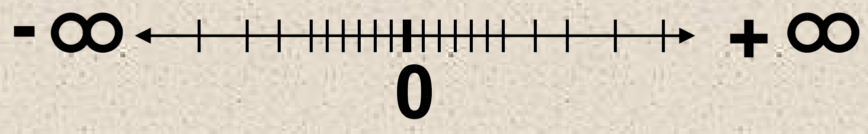
- Co je výsledkem operace: `sqrt(-4.0)` or `0/0`?
  - Jestliže nekonečno není chyba, tohle by také nemělo.
  - Nazývá se Not a Number (NaN)
  - Exponent = 255, mantisa je nenulová
- Aplikace
  - někdy lze „NaNy“ využít při ladění programu
  - šíření v návazných operacích: op(NaN, X) = NaN

Denormalizovaná čísla

- Problém: Kolem nuly se mezi reprezentovatelnými čísly vytváří mezera
  - Nejmenší kladné číslo: $a = 1.0\ldots_2 * 2^{-126} = 2^{-126}$
  - Druhé nejmenší kladné číslo: $b = 1.001_2 * 2^{-126} = 2^{-126} + 2^{-150}$
  - $a - 0 = 2^{-126}$
  - $b - a = 2^{-150}$



- Řešení:
  - Denormalizovaná čísla: nemají úvodní 1
  - Nejmenší kladné číslo: $a = 2^{-150}$
  - Druhé nejmenší kladné číslo: $b = 2^{-149}$



Častý omyl při práci s FP čísly

- FP operace Add, Sub jsou asociativní: CHYBA!

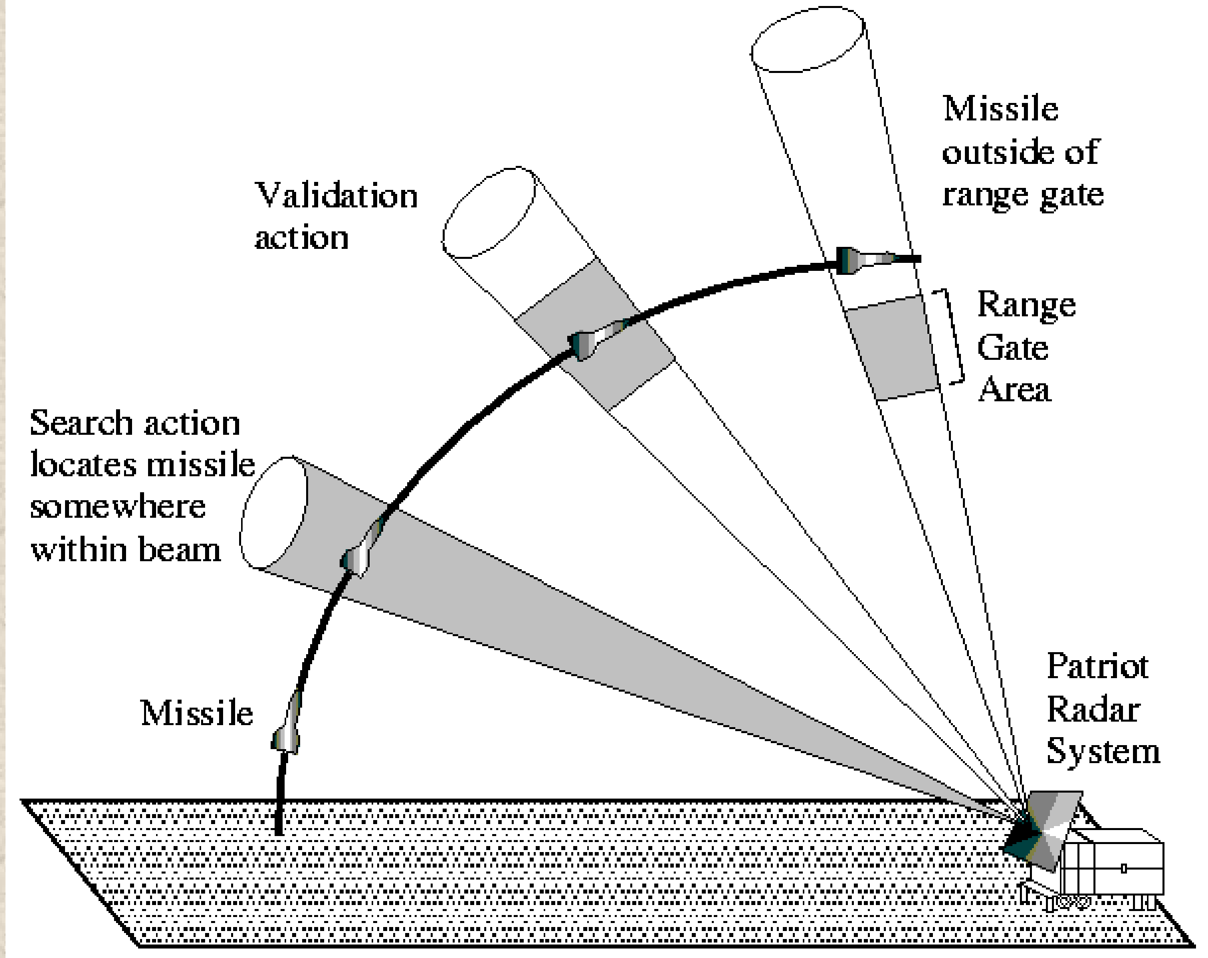
$x = -1.5 \times 10^{38}$ $y = 1.5 \times 10^{38}$ $z = 1.0$

$x + (y + z) = -1.5\text{x}10^{38} + (1.5\text{x}10^{38} + 1.0) = -1.5\text{x}10^{38} + (1.5\text{x}10^{38}) = \underline{0.0}$

$(x + y) + z = (-1.5\text{x}10^{38} + 1.5\text{x}10^{38}) + 1.0 = (0.0) + 1.0 = \underline{1.0}$

- Floating Point operace Add, Sub nejsou asociativní!
  - Proč? Výsledky operací s čísly FP **aproximují** výsledky operací s reálnými čísly
  - 1.5×10^{38} je mnohem větší než 1.0, takže $1.5 \times 10^{38} + 1.0$ v reprezentaci floating point dává stále 1.5×10^{38}

Chyby u čísel ve formátu FP



Perský záliv – 1990

Chyby v aritmetice protiraketového systému způsobily vážné ztráty

Zaokrouhlení a přesnost

- Při práci s čísly v pohyblivé řádové čárce přichází v potaz další fenomén.
- Číslo, uložené v FP formátu (běžně tedy v IEEE 754), představuje pouze aproximaci reálného čísla (protože počet míst za desetinnou/binární tečkou je omezený).
- I nejlepší počítače mohou pro zobrazení reálných čísel pouze vybírat aproximaci FP číslem, nejbližším zobrazovanému reálnému číslu.
- Pro dosažení co nejlepších výsledků používá norma IEEE 754 několik režimů pro zaokrouhlování FP čísel.

Zaokrouhlovací procesy

- Matematika reálných čísel => zaokrouhlování
- Zaokrouhlování také při konverzi typů
  - Double ⇔ single precision ⇔ integer
- Zaokrouhlení směrem k + nekonečno
  - VŽDY zaokrouhluje “nahoru”: 2.001 => 3; -2.001 => -2
- Zaokrouhlení směrem k - nekonečno
  - VŽDY zaokrouhluje “dolů”: 1.999 => 1; -1.999 => -2
- Oříznutí
  - Vypuštění nejméně významných bitů (zaokrouhlení k 0)
- Zaokrouhlení k (nejbližšímu) sudému (default)
  - 2.5 => 2; 3.5 => 4

Standardní podpora zaokrouhlení

- V dosud uvedených příkladech jsme poněkud „neopatrně“ zacházeli s délkou zobrazení mezivýsledků.
- Kdybychom „ořízli“ vše co reprezentujeme na délku zobrazení, nemohli bychom zaokrouhlovat, protože tak bychom ztratili potřebnou informaci.
- Z toho důvodu používá norma IEEE 754 vždy dva extra bity, které prodlužují mezivýsledky zprava během průběžných operací. Nazývají se *guard bit* a *round bit*.
- Tyto bity jsou vypočítávány během výpočtu podobně jako všechny ostatní. Na konci algoritmu jsou pak použity v procesu zaokrouhlení výsledku (po zaokrouhlení jsou uvolněny).

„Sticky“ bit

- Norma IEEE 754 dále zavádí třetí speciální bit, který se nazývá *sticky bit*.
- Tento bit leží úplně napravo a nastavuje se, jestliže jsou za *round bitem* nenulová data.
- Díky tomuto bitu dosahuje počítač stejných výsledků, jako kdyby byly mezivýsledky počítány s neomezenou přesností a pak zaokrouhleny.

Podpora pohyblivé řádové čárky u MIPS

- Architektura MIPS podporuje formáty IEEE 754 pro jednoduchou i dvojnásobnou přesnost…
  - **FP addition** – jednoduchá (*add.s*) a dvojitá (*add.d*)
  - **FP subtraction** – jednoduchá (*sub.s*) a dvojitá (*sub.d*)
  - **FP multiply** – jednoduchá (*mul.s*) a dvojitá (*mul.d*)
  - **FP divide** – jednoduchá (*div.s*) a dvojitá (*div.d*)
  - **FP comparison** – jednoduchá (*c.x.s*) a dvojitá (*c.x.d*), kde *x* je jedna z: equal (*eq*), not equal (*neq*), less than (*lt*), greater than (*gt*), less than or equal to (*le*) nebo greater than or equal (*ge*)
  - **FP branch** – pozitivní (*bc1t*) a negativní (*bc1f*)
  - (FP komparace nastavuje speciální bit na *true* nebo *false* a FP branch rozhoduje na základě této podmínky).

!? Řešení podmínek FP instrukcí klasickým způsobem ?!

Oddělené FP registry?

- Jedno z důležitých rozhodnutí při návrhu procesorů je, zda pro práci s FP čísly budou použity tytéž registry jako pro čísla integer a nebo vyhrazená sada registrů.
  - Operace integer a FP operace často pracují nad různými daty a proto nevzniká příliš mnoho konfliktů při sdílení registrů.
  - Hlavní důraz je kladen na oddělený soubor instrukcí přenosu dat pro FP.
  - Je-li použita oddělená sada registrů, dostáváte dvojnásobek registrů, aniž by bylo třeba více bitů v instrukčním formátu.
- Rozhodnutí návrháře – stojí to za to ?

Historická hlediska

- Jedním z důvodů, proč některé návrhy používají oddělené registrové banky pro integer a FP čísla je omezení, pocházející ze starších počítačů.
- V 80-tých letech nebylo ještě možné kvůli nízké hustotě integrace jednoduše zahrnout FP hardware na chip s hlavním procesorem. FP jednotka a její registrová banka byly implementovány na dalším chipu, který byl dodáván volitelně. Označoval se jako *akcelerátor* nebo jako *matematický koprocesor*.
- V 90-tých letech se začíná FP hardware integrovat rovnou do procesoru (spolu s mnoha dalšími funkcemi). Od té doby je použití koprocesorů méně časté.

Architektura MIPS - FP (1/2)

- Rozdílné FP instrukce pro:
  - jednoduchou přesnost: add.s, sub.s, mul.s, div.s
  - dvojitou přesnost: add.d, sub.d, mul.d, div.d
- Tyto instrukce jsou mnohem složitější, než odpovídající operace s typem integer

- Problémy:
  - Pro celý procesor je nepříznivé, jestliže se doba zpracování intrukcí zásadně liší.
  - Obecně platí, že během zpracování určitého programu většinou data nemění svůj charakter (FP < = > Int).
  - Některé programy vůbec neprovádí FP operace
  - Hardware pro rychlé provádění FP operací je značně rozsáhlý v porovnání s hardwarem pro operace integer

Architektura MIPS - FP (2/2)

- 1990 Řešení: vyhrazený čip, který provádí pouze FP.
- Koprocesor 1: FP čip
  - Obsahuje 32 32-bitových registrů: **`$f0, $f1`, …**
  - Většina registrů je specifikována v .s a .d instrukcích (`$f`)
  - Separátní load a store: **`lwc1`** a **`swc1`** (“load word coprocessor 1”, “store …”)
  - Dvojitá přesnost: **konvence**, sudý/lichý pár obsahuje jedno DP FP číslo: `$f0/$f1, … , $f30/$f31`
- 1990 Počítače často obsahují více vyhrazených obvodů:
  - Procesor: provádí běžné operace
  - Koprocesor 1: pouze FP operace;
- Přenos dat mezi hlavním procesorem a koprocesorem:
  - **`mfc0, mtc0, mfc1, mtc1`**, atd.

FP sada registrů MIPS

- Návrháři MIPS se rozhodli zařadit oddělenou sadu FP registrů `$f0`, `$f1`, atd.
- Pro plnění a ukládání FP registrů jsou také použity vyhrazené instrukce – *lwc1* a *swc1*. Jako bázové registry jsou použity normální registry z integer sady.
- Následující příklad ukazuje načtení dvou čísel v jednoduché přesnosti, jejich sečtení a uložení výsledku…

```
lwc1   $f2, x($sp)   # load 32-bit FP num into $f4
lwc1   $f6, y($sp)   # load 32-bit FP num into $f6
add.s  $f2,$f4,$f6   # $f2=$f4+$f6, single precision
swc1   $f2, z($sp)   # store 32-bit FP num from $f2
```

- Registr pro dvojitou přesnost je tvořen párem registrů jednoduché přesnosti (sudý/lichý), používající jméno sudého.

C => MIPS

```
Float f2c (float fahr) {
    return ((5.0 / 9.0) * (fahr – 32.0));
}
```

```
F2c:
    lwc1    $f16, const5($gp)      # $f16 = 5.0
    lwc1    $f18, const9($gp)      # $f18 = 9.0
    div.s   $f16, $f16, $f18       # $f16 = 5.0/9.0
    lwc1    $f20, const32($gp)     # $f20 = 32.0
    sub.s   $f20, $f12, $f20       # $f20 = fahr – 32.0
    mul.s   $f0, $f16, $f20        # $f0 = (5/9)*(fahr-32)
    jr      $ra                    # return
```

Podpora FP u architektury PowerPC

- Architektura PowerPC je z hlediska zobrazení čísel v pohyblivé řádové čárce podobná MIPS. Existuje několik rozdílů, které pramení hlavně z toho, že PowerPC je novější a pokročilejší architektura.
  - Neobsahuje žádné registry Hi a Lo – PowerPC instrukce operují přímo nad registry.
  - PowerPC má 32 registrů pro jednoduchou přesnost a 32 registrů pro dvojitou přesnost, tedy dvakrát tolik, než architektura MIPS.
  - Power PC zavádí také instrukci multiply-add (více na následujícím snímku).

Instrukce multiply-add

- Instrukce PowerPC multiply-add pro FP operandy čte tři operandy, dva z nich vynásobí, třetí připočítá k součinu a výsledek uloží do registru, kde ležel třetí operand.
  - dvě instrukce MIPS = jedna PowerPC instrukce
  - tato instrukce provádí na závěr jediné zaokrouhlení; dvě oddělené instrukce = dvě zaokrouhlení a tím i nižší přesnost výsledku
- Tato instrukce je také použita v PowerPC při provádění FP dělení (použitím Newtonovy iterační metody, jak bylo zmíněno) – přesnost dělení byla primárním důvodem pro redukci počtu zaokrouhlení (zaokrouhlení až na závěr této sdružené operace).

Podpora FP u architektury IA-32

- Podpora operací v pohyblivé řádové čárce u IA-32/x86 započala s koprocesorem 8087 v roce 1980 a velmi se lišila od architektur MIPS a PowerPC.
- Intel používá zásobníkově orientovanou architekturu s FP instrukcemi, jedná se o odlišný samostatný celek.
  - Operace Load ukládají FP čísla na vrcholek FP zásobníku a inkrementují FP stack pointer.
  - Operace Store odebírají FP čísla z vrcholku FP zásobníku, dekrementují FP stack pointer a ukládají FP čísla do paměti.

Zásobníková FP architektura

- FP operace se provádějí se dvěma operandy na vrcholku zásobníku, operandy jsou nahrazeny jedním výsledkem (dvakrát pop, jeden push).
- Existuje také možnost provádět FP operaci s jedním operandem v paměti a druhým ležícím na vrcholku zásobníku nebo v jednom ze sedmi speciálních FP registrů.
- FP instrukce v IA-32 patří do jedné ze čtyřech skupin …
  - Přesun dat – load, load immediate, store, atd.
  - Aritmetika – add, sub, mul, div, sqrt, abs, atd.
  - Komparace – může zasílat výsledek do integer procesoru, který pak případně vykoná instrukci větvení
  - Transcendentní – sinus, kosinus, log, exp, atd.

Zásobníkově orientované stroje

- Tato zásobníkově orientovaná architektura se velmi liší od registrově orientované, kterou jsme se doposud zabývali.
- Data se pro provedení operací přenáší do/ze zásobníku namísto registrů procesoru.
- Tento typ architektury není neobvyklý. Některé počítače (i velmi moderní) používají podobnou architekturu …
  - Java Virtual Machine (JVM)
  - Microsoft Common Language Runtime (CLR)
  - Většina graf. HP kalkulátorů (interface, nikoliv použitý procesor)

Dvojitá rozšířená přesnost

- Zásobník v systému pohyblivé řádové čárky Intel IA-32 je široký 80 bitů => označení *double extended precision*.
- Všechna FP čísla jednoduché i dvojité přesnosti jsou konvertována do tohoto formátu, když se přesouvají z paměti do zásobníku a naopak.
- FP registry mají šířku 80 bitů.
- Leží-li jeden operand FP operace v paměti, je během operace (on-the-fly) konvertován do 80-bitového formátu.
- Tato forma není využívána kompilátory moderních programovacích jazyků, ale je k dispozici pro přímé programování v assembleru (časově náročné algoritmy).

Závěr

- Čísla s pohyblivou řádovou čárkou (FP) pouze ***aproximují*** reálná čísla, která bychom chtěli používat, představují dokonce jen podmnožinu racionálních čísel.
- IEEE 754 Floating Point Standard je dnes široce akceptovaným standardem pro práci s FP aritmetikou.
- Nové prvky architektury MIPS
  - Registry **(`$f0-$f31`**)
  - Jednoduchá přesnost (32 bitů, $2x10^{-38}$… $2x10^{38}$)
    - **`add.s, sub.s, mul.s, div.s`**
  - Dvojitá přesnost (64 bitů, $2x10^{-308}$…$2x10^{308}$)
    - **`add.d, sub.d, mul.d, div.d`**
- Typ není asociován s daty, význam bitů závisí na kontextu (například *int* vs. *float*)

Standardní reprezentace FP čísel IEEE 754

Jednoduchá přesnost (32-bit)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	exponent								mantisa																						
1bit	8 bitů								23 bitů																						

$$(-1)^{sign} \times (1+ \text{mantisa}) \times 2^{exponent-127}$$

Dvojitá přesnost (64-bit)

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
S	exponent											significand																			
1bit	11 bitů											20 bitů																			

mantisa (pokračování)
32 bitů

$$(-1)^{sign} \times (1+ \text{mantisa}) \times 2^{exponent-1023}$$