B35APO: Architektury počítačů

Lekce 02. Reprezentace čísel

Petr Štěpán
stepan@fel.cvut.cz

12. září, 2023

Obsah

Opakování

V minulé lekci jsme měli:

- Reprezentace bitu pomocí napětí
- Reprezentace bajtů jako více paralelních vodičů, každý s hodnotou jednoho bitu
- Sčítání dvou celých kladných čísel
- Bitový posun celých čísel (násobení, nebo dělení mocninou 2)

Dnes:

- Rozsahy celých čísel a ukládání do paměti
- Násobení a dělení celých kladných čísel
- Reprezentace záporných čísel a operace s nimi
- Přetečení sčítání a odčítání
- Reálná čísla

Kvíz

Jak rychle lze spočítat součet dvou n-bitových čísel a kolik k tomu potřebujeme tranzistorů?

A v konstantním čase s lineárním počtem tranzistorů – $O(n)$

B v konstantním čase s exponenciálním počtem tranzistorů – $O(2^n)$

C v logaritmickém čase s lineárním počtem tranzistorů – $O(n)$

D v logaritmickém čase s kubickým počtem tranzistorů – $O(n^3)$

E v logaritmickém čase s exponenciálním počtem tranzistorů – $O(2^n)$

Kladná čísla

Reprezentace celých kladných čísel

V jazyce C jsou typy (podle normy ISO/IEC 9899:TC3):

typ	min	max	počet bajtů
unsigned char	0	255	1
unsigned short	0	65 535	2
unsigned long	0	4 294 967 295	4
unsigned long long	0	18 446 744 073 709 551 615	8

- Norma definuje většinou minimální rozsahy, `unsigned int` nemusí mít jen 2 bajty, ale většinou má 4 bajty (GNU, MS C).
- Pokud chcete zjistit skutečnou velikost, použijte např. příkaz `sizeof(int)`
- Doporučujeme používat typy `uintX_t` a `intX_t` (kde `X` je 8, 16, 32, nebo 64), např. `uint8_t`, `int64_t`
- V některých případech je vhodné použít typy `[u]int_fastX_t` a `[u]int_leastX_t`, např. `uint_least8_t`, `int_fast64_t`

Kladná čísla

V zápis konstant v jazyce C v soustavě:

- desítkové – nesmí začínat 0 kromě 0
- osmičkové – začíná 0
- hexadecimální – začíná 0x
- binární – začíná 0b (pouze GNU překladač)

Příklad: 252 == 0xfc == 0374 == 0b11111100

Poznámka: Podle hexadecimálního zápisu zjistíte velikost čísla v bajtech, např. 0x123456 se vejde do tří bajtů.

Kladná čísla - uložení v paměti

- Paměť počítače pracuje s bajty
- Historicky vzniklo několik možností uložení čísel do paměti.

Jak lze tedy uložit číslo 0x12345678 do paměti:

adresa	Big-endian	Little-endian
400	0x12	0x78
401	0x34	0x56
402	0x56	0x34
403	0x78	0x12

- Procesory Intel zavedly little-endian, procesory Motorola zavedly big-endian.
- Je to důležité, když získáte například přes internet data po bajtech, musíte definovat, jaká čísla reprezentují
- RISC V - little-endian, MIPS - big-endian
- Bitcoin - DER signatures big-endian, transaction hash little-endian

Kladná čísla - kvíz

```c
#include <stdio.h>
int main() {
  unsigned char p[] = {0,0,0,0};
  *(int*)p=10;
  printf("%02x,%02x,%02x,%02x\n", p[0],p[1],p[2],p[3]);
}
```

Co bude výstupem tohoto programu na procesorech Intel?

A nic, program nelze přeložit

B náhodný výstup, p nelze přetypovat na *int

C 0a,00,00,00

D 00,00,00,0a

Násobení celých čísel

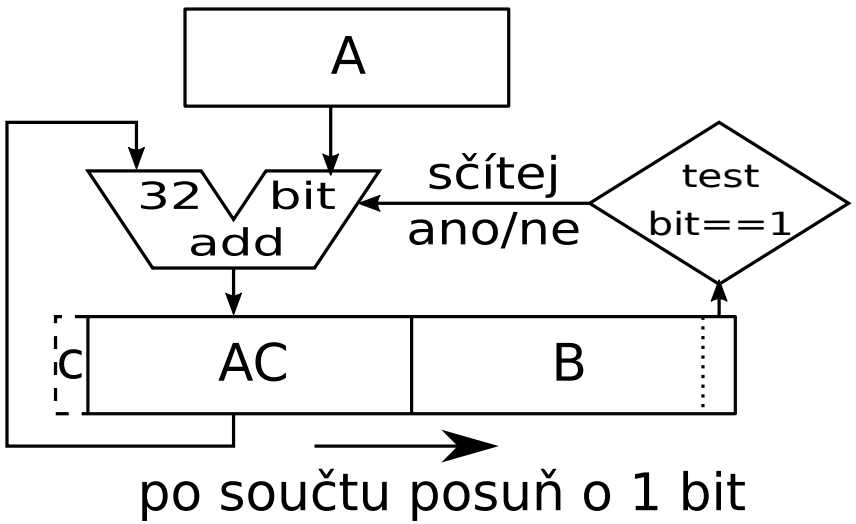
Stejný princip násobení, jak jste se ho naučili na základní škole pro desítkovou soustavu:

```
  153
  *45
------
  765
 612
------
 6885
```

```
  10011001
   *101101
-----------
  10011001
 00000000
10011001
10011001
00000000
10011001
-----------
1101011100101
```

Násobení celých čísel

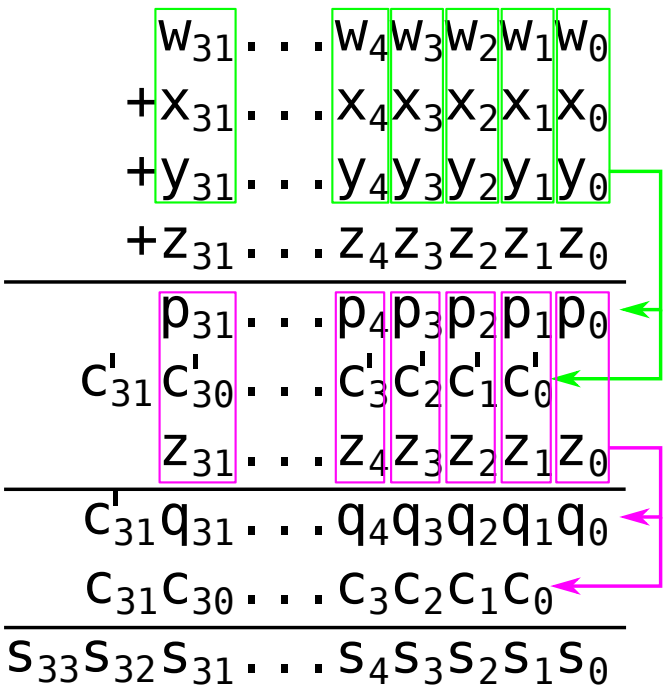
Podle uvedeného algoritmu můžeme vytvořit následující násobičku s posuvným registrem:
(A,B 32 bitů, výsledek 64 bitů)



- Výsledek je ve dvou registrech AC a B
- Pomalé, už sčítání je náročné, nyní 32 nebo 64 sčítání.

Rychlé násobení - Motivace Wallace tree

Jak to zrychlit - odložené carry (Carry Save Adder).
Jak nejrychleji spočítat součet čtyř 32-bitových čísel:



- Sečteme paralelně p=w+x a q=y+z a potom p+q – trvá dlouho, nejméně doby dvou plných součtů
- Odložíme carry (nebudeme carry propagovat, jen v posledním kroku):
  - 1. krok použijeme Full adder a sečteme vždy bity $w_i + x_i + y_i = c'_i p_i$
  - 2. krok použijeme Full adder a sečteme vždy bity $p_i + c'_{i-1} + z_i = c_i q_i$
  - 3. krok použijeme normální sčítačku 32-bitových čísel ($s_0 = q_0$, $c'_3 2$ připojíme ke q)

Rychlé násobení - Wallace tree

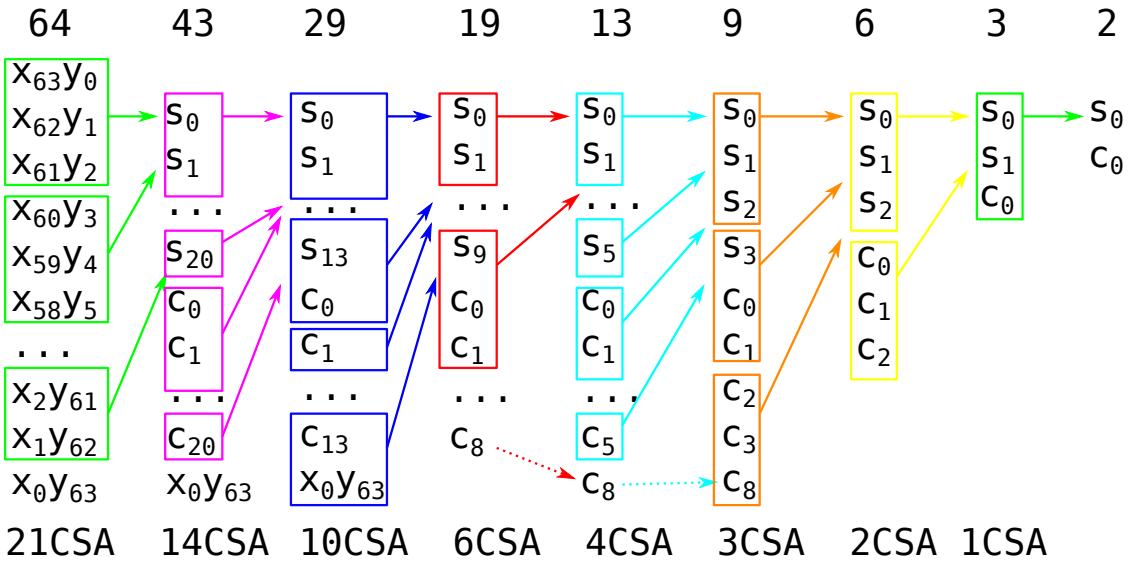
Použijeme předchozí princip na co nejrychlejší součet 32 nebo 64 různých čísel:

$$
\begin{array}{cccccccc}
 & x_{63}...x_4x_3x_2x_1x_0 * & y_{63}...y_4y_3y_2y_1y_0 & & & & & \\
\hline
 & 0 & 0 & 0 & \cdot\ x_{63}y_0 & \cdot\ x_2y_0 & x_1y_0 & x_0y_0 \\
 & 0 & 0 & 0 & \cdot\ x_{62}y_1 & \cdot\ x_1y_1 & x_0y_1 & 0 \\
 & 0 & 0 & 0 & \cdot\ x_{61}y_2 & \cdot\ x_0y_2 & 0 & 0 \\
 & & ... & & \cdot\ ... & \cdot & ... & \\
 & 0 & 0 & x_{63}y_{61} & \cdot\ x_2y_{61} & \cdot\ 0 & 0 & 0 \\
 & 0 & x_{63}y_{62} & x_{62}y_{62} & \cdot\ x_1y_{62} & \cdot\ 0 & 0 & 0 \\
 & x_{63}y_{63} & x_{62}y_{63} & x_{61}y_{63} & \cdot\ x_0y_{63} & \cdot\ 0 & 0 & 0 \\
\hline
q_{127} & q_{126} & q_{125} & q_{124} & q_{63} & q_2 & q_1 & q_0
\end{array}
$$

- Součiny jsou jednoduché: $x_i \cdot y_j = x_i \text{ and } y_j$
- Nejtěžší je sečíst prostřední sloupec 64 jednobitových čísel
- Pustíme na všechny bity co můžeme paralelně sčítačky a carry budeme přičítat v dalších krocích
- V první fázi to bude 1323 sčítaček

Rychlé násobení - Wallace tree

Když se podíváme jen na prostřední nejdelší sloupec:



- Vidíme, že za 8 kroků sčítaček, tedy 16 zpoždění hradel nám zbývá sečíst dva bity
- Vpravo od tohoto sloupce je již něco sečteno a již se i propagovali carry posledních 8 sčítanců
- Zbývá sečíst dvě 120-bitová čísla (součty a carry), což se dá stihnout asi za 30 zpoždění hradel
- Výsledek - vynásobíme dvě čísla za cenu doby odpovídající přibližně dvěma součtům 64-bitových čísel
- V praxi to trvá trochu déle, je použito méně sčítaček CSA.

Dělení celých čísel

Dělení lze zkonstruovat podobně, jako jste se učili na základní škole:

```
 240:11=21
-22
---
  20
 -11
---
   9
```

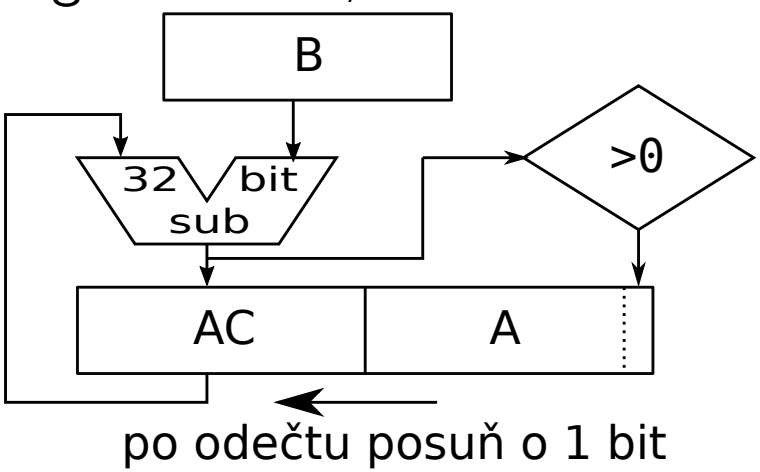
```
 11110000:1011=10101
-1011
-----
  1000
  10000
  -1011
-------
   1010
   10100
   -1011
---------
    1001
```

Oba výpočty počítají, že 240 děleno 11 je 21 a zbytek neboli 240%11=9.

Dělení celých čísel

Dělička počítající `A/B`, A má 64 bitů, B 32 bitů:

číslo A je uloženo ve dvou registrech AC,A



- Výsledek: v registru A je podíl, v registru AC je zbytek – modulo
- v posledním kroku je nutné posunout pouze registr A, AC se již neposouvá – promyslete proč
- Dělička – existuje rychlejší algoritmus High Radix Division (je velmi složitý, přesahuje rozsah tohoto předmětu)
  - Odhaduje několik bitů najednou a pak upřesňuje iteracemi
  - 1994 – Pentium FDIV bug – chyba při implementaci algoritmu Sweeney, Robertson, and Tocher (SRT) - odhaduje dva bity

Obsah

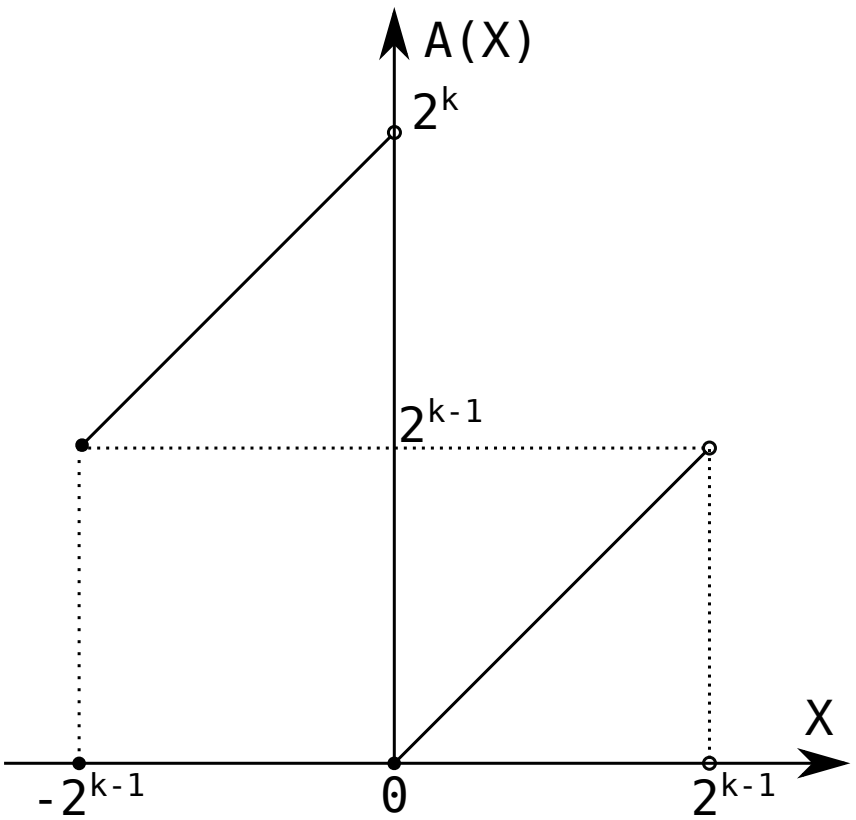
Záporná čísla

Potřebujeme zakódovat znaménko do reprezentace čísla:

- naivně - vrchní bit bude znaménko
  - Máme 0 a -0, přitom je to stejné číslo
  - Potřebujeme jiný algoritmus na sčítání
- dvojkový doplněk – two complement
  - reprezentace X k-bitovým číslem je vlastně $X \bmod 2^k$
    - pro $X \geq 0$ je reprezentace X
    - pro $X < 0$ je reprezentace $2^k - |X|$
  - výhoda: sčítání funguje tak, jak jsme si ho navrhli pro všechna čísla, kladná i záporná.
  - 8-bitová -1 je `11111111`
  - 5+(-1) je `101+11111111=100000100`, devátý bit se do reprezentace čísla nevejde, tedy výsledek je `101+11111111=100`

Dvojkový doplněk

- rozsah čísel reprezentovaných k-bity je $< -2^{k-1}, 2^{k-1} - 1 >$
- pro číslo X budeme značit A(X) jeho reprezentaci v dvojkovém doplňku:



Binární hodnota	Dvojkový doplněk
00000000	$0_{(10)}$
00000001	$1_{(10)}$
...	...
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-128_{(10)}$
10000001	$-127_{(10)}$
10000010	$-126_{(10)}$
...	...
11111101	$-3_{(10)}$
11111110	$-2_{(10)}$
11111111	$-1_{(10)}$

Opačné číslo

- Protože sčítání funguje s dvojkovým doplňkem, tak nemusíme navrhovat obvod pro odčítání, jednoduše `A-B` vypočteme jako `A+(-B)`
- potřebujeme vymyslet jak získat z `B` hodnotu `-B`
  - víme, že záporné $X = 2^k - |X|$
  - pokud znegujeme každý bit, tak je to vlastně $(2^{k-1} - 1) - X$, protože $2^{k-1} - 1$ je číslo složené z k jedniček
- Výsledný postup je tedy:
  1. znegujte všechny bity čísla X
  2. k výslednému číslu přičti 1

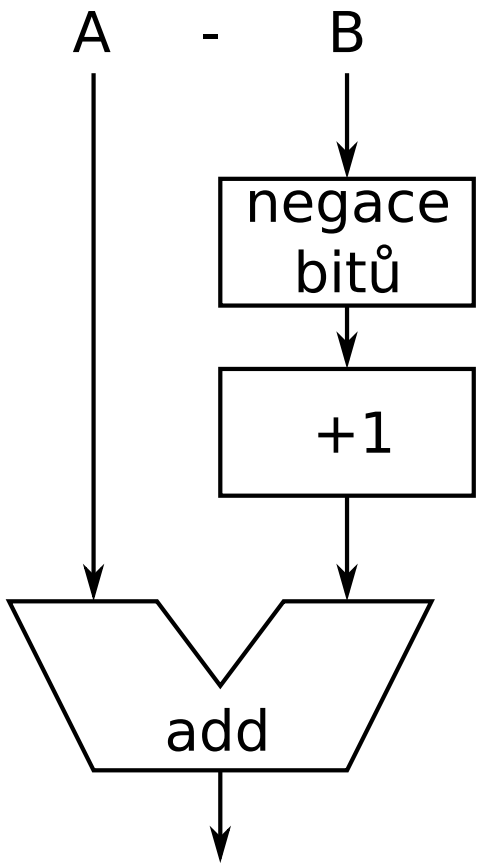
Příklad:

53=0b00110101 znegováním cifer dostaneme -54=0b11001010 po přičtení 1 pak -53=0b11001011

Odčítání

Odčítání lze řešit:

- speciálním obvodem podobným sčítačce se všemi možnostmi zrychlení jako byly u sčítání
- nebo z druhého čísla vytvoříme číslo opačné a to pak jednoduše sečteme s tím prvním



Násobení a dělení

- pro násobení lze odvodit, že pokud bychom ignorovali znaménko a spočítali $M \cdot N$, pak pro M,N obě k-bitová čísla musíme výsledek upravit takto:

$$A(M \cdot N) = A(M) \cdot A(N) \begin{cases} -A(M) \cdot 2^k & \text{pokud } M < 0 \\ -A(N) \cdot 2^k & \text{pokud } N < 0 \end{cases}$$

- protože reprezentace záporného čísla dvojkovým doplňkem je vlastně $A(M) = 2^k + M$, pak součin dvou záporných čísel je $(2^k + M) \cdot (2^k + N) = 2^{2 \cdot k} + 2^k \cdot M + 2^k \cdot N + M \cdot N$
- moderní násobení a dělení počítá s absolutními hodnotami a nakonec určíme znaménko výsledku podle znaménka operandů.
  - nejvyšší bit značí znaménko
  - výpočet opačného čísla je rychlý a levný

Čísla se znaménkem

Reprezentace celých čísel

V jazyce C jsou typy:

typ	min	max	počet bajtů
char	-128	127	1
short	-32 768	32 767	2
long	-2 147 483 648	2 147 483 647	4
long long	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	8

- Norma C má ve skutečnosti min o 1 větší , kvůli procesorům s jedničkovým doplňkem – ten se dnes již prakticky nepoužívá
- Stejně tak `int` musíte otestovat, zda je 2 bajtový nebo 4 bajtový

Kvíz

Uvažujte tento program:

```
#include <stdio.h>
int main() {
  unsigned char a=150u, b=120u, c;
  char sa=-100, sb=-80, sc;

  c=a+b;
  sc=sa+sb;
  printf("c=%u sc=%d\n", c, sc);
}
```

Co program vytiskne:

A c=270 sc=-180
B c=14 sc=-76
C c=14 sc=76
D c=-14 sc=-76
E Numeric error

Přetečení při sčítání čísel bez znaménka

Pokud se podíváme co se stalo, unsigned char je 8-bitová reprezentace čísel:

```
 150 =   1001 0110
+120 =   0111 1000
-----------------
  14 =   0000 1110
 270 =1  0000 1110
```

Protože se výsledek nevejde do 8-bitů, nejvyšší jednička se ztratí a výsledek je pouze 14.

Pokud přetečení chcete detekovat, máte následující možnosti:

- C23 bool ckd_add(type1 *result, type2 a, type3 b) – součet dvou čísel s detekcí přetečení
- GNU GCC 5+, Clang 3.8+ __builtin_add_overflow(a, b, result) – obě verze i pro sub a mul

Přetečení při sčítání čísel se znaménkem

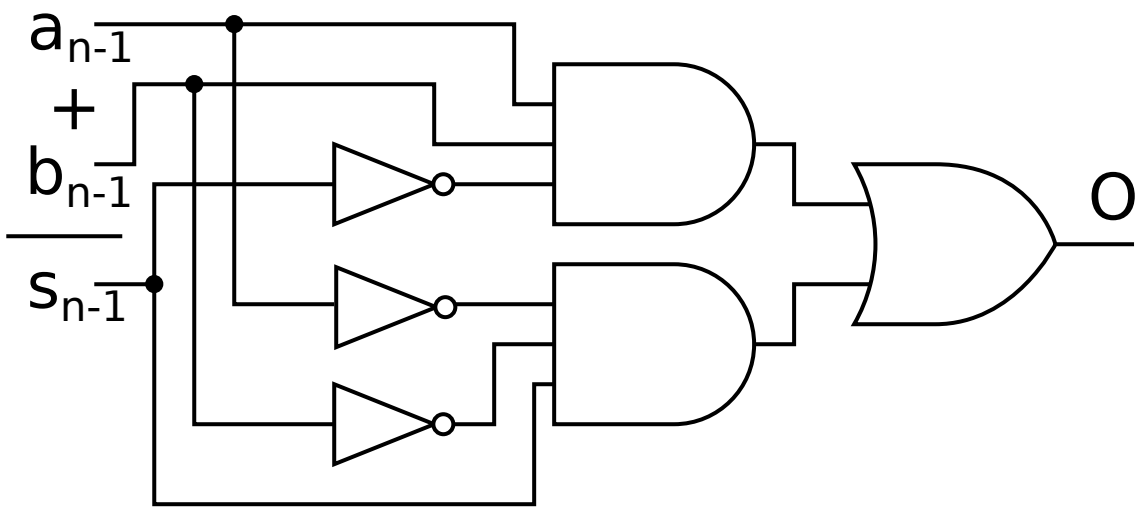
Přetečení při sčítání čísel se znaménkem je složitější. Co je dobře a co je špatně:

```
-112 = 10010000        -12 = 11110100        -90 = 10100110
+ 45 = 00101101    + -20 = 11101100    + -42 = 11010110
-----------------------------------------------------------
 -67 = 10111101        -32 =111100000        124 =101111100
```

DOBŘE | DOBŘE | ŠPATNĚ

- Přetečení při sčítání čísel se znaménkem nastane právě tehdy, když dojde k přenosu na posledních cifrách a zároveň nedojde k přenosu na předposledních cifrách:
  - overflow = c_n xor c_{n-1}; c_n je nejvyšší carry, c_n je druhé nejvyšší carry
- Druhá možnost je sledovat, zda při součtu dvou kladných čísel máme záporná výsledek, nebo při součtu dvou záporných čísel je kladný výsledek:
  - overflow = (a_n and b_n and (not s_n)) or ((not a_n) and (not b_n) and s_n); a_n, b_n jsou nejvyšší bity sčítanců; s_n je největší bit výsledku

Kvíz



Při sčítání dvou čísel s odlišnými znaménky:

A může dojít k přetečení jen v dvojkovém doplňku

B může dojít k přetečení pouze při jiné reprezentaci než je dvojkový doplněk

C nemůže dojít pouze při reprezentaci v dvojkovém doplňku

D nemůže dojít v žádné reprezentaci čísel se znaménkem

Jiné reprezentace záporných čísel

Čísla s posunutou nulou:

- pro k-bitovou reprezentaci zvolíme číslo S (většinou $S = 2^{k-1}$ nebo $S = 2^{k-1} - 1$)
- reprezentace – kód čísla X je $A(X) = X + S$
- rozkódování je funkcí $D(A) = A - S$
- rozsah čísel reprezentovaných je $< -S, 2^k - S - 1 >$

Pro k=8 a S=127

Binární hodnota	Hodnota
00000000	$-127_{(10)}$
00000001	$-126_{(10)}$
...	...
01111110	$-1_{(10)}$
01111111	$0_{(10)}$
10000000	$1_{(10)}$
10000001	$2_{(10)}$
...	...
11111110	$127_{(10)}$
11111111	$128_{(10)}$

Počítání s posunutou nulou

- sčítání a odčítání čísel s posunutou nulou je složitější:
- $A(X+Y) = (X+Y)+S = (X+S)+(Y+S)-S = A(X)+A(Y)-S$
- $A(X-Y) = (X-Y)+S = (X+S)-(Y+S)+S = A(X)-A(Y)+S$
- násobení je ještě složitější:
- $A(X\cdot Y) = (X\cdot Y)+S = (X+S)\cdot(Y+S)-(X+S+Y+S)\cdot S+S^2+S = A(X)\cdot A(Y)-(A(x)+A(y))\cdot S+S^2+S$

- Přetečení:
  - při sčítání jsou znaménka sčítanců stejná a výsledek má jiné znaménko
  - při odčítání čísel s různými znaménky má výsledek jiné znaménko než menšenec

Jiné reprezentace záporných čísel

Doplněk jedné:

- záporné číslo je negací bitů opačného čísla
  - pro $X \geq 0$ je reprezentace $A(X) = X$
  - pro $X < 0$ je reprezentace $A(X) = 2^k - 1 - |X|$
- nevýhody: dvě reprezentace 0, složitější sčítání čísel se znaménky

BCD formát

- jiná reprezentace celých čísel, každá cifra jeden nibble
  - číslo 1234 je uloženo v hexadecimálním tvaru jako `0x1234`
- výhody: snadný převod na desítkovou soustavu
- nevýhody: neefektivní reprezentace, složitější počítání

Obsah

Reálná čísla

- Celé číslo X v binární soustavě je součet k bitů b_i vynásobených mocninami 2, tedy $X = \sum_{i=0}^{k-1} b_i 2^i$
- Reálné číslo X v binární soustavě je obdobný součet $k + j$ bitů b_i vynásobených mocninami 2, ale začneme již v záporných mocninách: $X = \sum_{i=-j}^{k} b_i 2^i$
- jistě si všichni pamatujete ze střední školy, že $2^{-j} = \frac{1}{2^j}$

$$\begin{array}{cccccccccccccc} & & & & & & & \frac{1}{2} & \frac{1}{4} & & & & & \\ & & & & & 4 & 2 & 1 & \frac{1}{2} & \frac{1}{4} & & & & \\ & & & & & \| & \| & \| & \| & \| & & & & \\ 2^{i} & 2^{i-1} & 2^{i-2} & \ldots & 2^{2} & 2^{1} & 2^{0} & 2^{-1} & 2^{-2} & \ldots & 2^{-j+1} & 2^{-j} \\ b_i & b_{i-1} & b_{i-2} & \ldots & b_2 & b_1 & b_0 . & b_{-1} & b_{-2} & \ldots & b_{-j+1} & b_{-j} \end{array}$$

Reálná čísla s pevnou desetinnou čárkou

Čísla s pevnou desetinnou čárkou (fixed point numbers):

- obdoba čísel s posunutou 0
- reálné číslo je reprezentované k-bitovým celým číslem se znaménkem, dále zvolíme pevný počet desetinných míst s bitů, kdy $0 \leq s \leq k$
- reprezentace – kód čísla X je $A(X) = \lceil X \cdot 2^s \rceil$
- rozkódování je funkcí $D(A) = \frac{A}{2^s}$
- rozsah čísel reprezentovaných je $< -\frac{2^{k-1}}{2^s}, \frac{2^{k-1}-1}{2^s} >$
- přesnost reprezentace čísel je $\pm\frac{1}{2^s}$.

Speciální případ:

- pokud chcete reprezentovat reálná čísla z intervalu $< 0, 1)$
- pak můžete nastavit $s = k$ a použít bezznaménkovou reprezentaci celých čísel
- dosáhnete lepší přesnosti než odpovídající `float`, nebo `double`

Některé SIMD instrukce používají pevnou desetinnou čárku.

Reálná čísla s pevnou desetinnou čárkou

Počítání s čísly s pevnou desetinnou čárkou:

- součet a rozdíl je součtem a rozdílem celočíselné reprezentace čísel
- násobení je složitější:
  - $A(X \cdot Y) = (X \cdot Y) \cdot 2^s = \frac{(X \cdot 2^s) \cdot (Y \cdot 2^s)}{2^s} = \frac{A(X) \cdot A(Y)}{2^s}$
- dělení je obdobné:
  - $A(\frac{X}{Y}) = (\frac{X}{Y}) \cdot 2^s = \frac{(X \cdot 2^s) \cdot 2^s}{(Y \cdot 2^s)} = \frac{A(X) \cdot 2^s}{A(Y)}$
- Nejedná se o výrazné zesložitění, protože násobení číslem 2^s je posun o s bitů doleva a dělení číslem 2^s je posun o s bitů doprava.

Plovoucí desetinná čárka

Plovoucí desetinná čárka (floating point numbers)

Obdoba vědeckého zápisu reálných čísel:
$-123000000000000.0 = -1.23 \cdot 10^{14} = -1.23\text{E}14$
$0.000000000000123 = -1.23 \cdot 10^{-13} = -1.23\text{E} - 13$

Binární reprezentace používá obdobně mocninu 2:
$110110000000000.0 = 1.1011 \cdot 2^{14} = 1.1011\text{E}14 = 29696_{10}$
$-0.0000000000000011101 = -1.1101 \cdot 2^{-16} = -1.1101\text{E} - 16 \approx$
$\approx 0.00002765_{10}$

Všimněte si, že každé reálné číslo (kromě 0) začíná v binárním vědeckém zápisu jedničkou.

IEEE-754

Standard IEEE-754 definuje, jakým způsobem zakódovat reálné číslo do 32, 64 bitů.
Reálné 32-bitové číslo obsahuje:

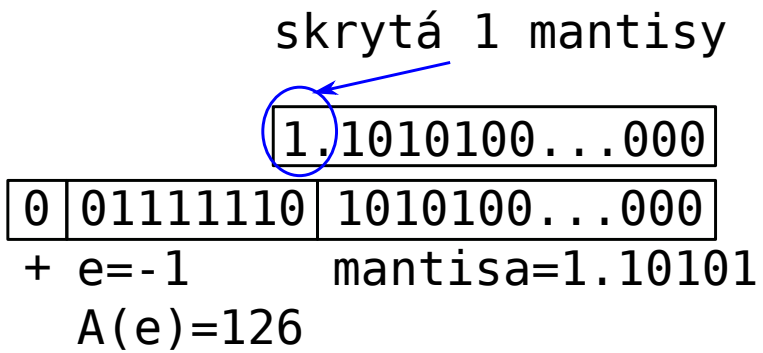
- 1 bit znaménko (pozn. existuje 0 i -0)
- 8 bit hodnota exponentu s posunutou nulou o 127
- 23 bitů hodnota mantisy, tedy cifer reprezentujících hodnotu

Reálné 64-bitové číslo obsahuje:

- 1 bit znaménko (pozn. existuje 0 i -0)
- 11 bitů hodnota exponentu s posunutou nulou o 1023
- 52 bitů hodnota mantisy, tedy cifer reprezentujících hodnotu

IEEE-754

Příklad: Reálné číslo $0.828125_{(10)} = 0.5 + 0.25 + 0.0625 + 0.015625 = 2^{-1} + 2^{-2} + 2^{-4} + 2^{-6} = 0.110101_{(2)}$.
Číslo převedeme do vědecké notace: $0.110101 = 1.10101\mathrm{E} - 1$.
Exponent $e = -1$, reprezentace s posunutou nulou o 127 je $A(-1) = -1 + 127 = 126$.
Abychom neplýtvali zbytečně bity, pro normalizovaná čísla se první 1 mantisy neukládá do binární reprezentace čísla:



Zkuste si
`https://www.h-schmidt.net/FloatConverter/IEEE754.html`

IEEE-754

Normalizované číslo, je každé číslo, které lze zapsat jako `1.XXXXX E exp`, kde exp je číslo od -126 do 127, tedy jehož reprezentace je od 1 do 254.

Denormalizovaná čísla jsou čísla, která lze zapsat jako `0.XXXXX E -126`, tedy například 0.0, nebo všechna čísla v intervalu $(-1.17549\text{E}-38, 1.17549\text{E}-38)$, tedy od $(-2^{-126}, 2^{-126})$.

Pokud je reprezentace exponentu nastavena na samé 1, tedy 255, tzn. hodnota exponentu je 128, pak se jedná o speciální čísla.

- pokud je mantisa 0, pak se jedná o nekonečno, podle znaménka *-inf*, nebo *inf*.
- pokud je mantisa nenulová, pak se jedná o speciální výraz *NaN* – Not a Number, tedy chybná hodnota čísla, například po výpočtu odmocniny záporného čísla

IEEE-754

Přehled reálných čísel:

Exponent	Mantisa	Hodnota
00000000	0	0.0 – čistá nula
00000000	nenulová	Denormalizovaná čísla blízká 0
00000001	0	nejmenší normalizované číslo se skrytou 1 v mantise
1 až 254	cokoliv	normalizovaná čísla, skrytá 1 v mantise
11111111	0	nekonečno
11111111	nenulová	NaN chybná hodnota

Denormalizované číslo s nejmenší absolutní hodnotou různé od 0 je:

- exponent = 0 (-126), mantisa=000...0001, hodnota = $2^{-23+(-126)} \approx 1.4\text{E} - 45$

Normalizované číslo s nejmenší absolutní hodnotou:

- exponent = 1 (-126), mantisa=000...0000, hodnota = $2^{-126} \approx 1.17\text{E} - 38$

Normalizované číslo s největší absolutní hodnotou:

- exponent = 255 (127), mantisa=111...1111, hodnota = $(2 - 2^{-23})2^{127} \approx 3.4\text{E}38$

IEEE-754 revize 2008

Standard IEEE-754 navíc definuje reálné číslo s 16 bity (half precision) a s 128 bity (quad precision).
Reálné 16-bitové číslo obsahuje:

- 1 bit znaménko
- 5 bitů hodnota exponentu s posunutou nulou o 15
- 10 bitů hodnota mantisy, (plus 11tý bit skrytý neukládaný)

Reálné 128-bitové číslo obsahuje:

- 1 bit znaménko
- 15 bitů hodnota exponentu s posunutou nulou o 16383
- 112 bitů hodnota mantisy, (plus 113 bit skrytý)

IEEE-754

Porovnání reálných čísel na velikost:

- Kladná čísla jsou vždy větší než záporná
- Po odstranění znamének, lze absolutní hodnoty čísel porovnat tak, jak jsou uložené v paměti jako celé číslo bez znaménka
  - To je možné díky zvolené reprezentaci exponentu jako čísla s posunutou nulou
  - Větší exponent větší číslo, při rovnosti exponentů větší mantisa znamená větší číslo

IEEE-754

Sčítání nebo odčítání dvou reálných čísel ve vědecké notaci

1. Převést mantisy čísel na společný exponent, který má hodnotu největšího exponentu (porušíme notaci)
2. Provést součet, nebo rozdíl mantis i se skrytými jedničkami
3. Výsledné číslo normalizovat
   - při sčítání se může exponent zvětšit
   - při odčítání se exponent může zmenšit

IEEE-754

Příklad: Sčítání dvou reálných čísel 31.5+0.75

$31.5_{(10)} = 11111.1_{(2)} = 1.11111\text{E}4 \qquad 0.75_{(10)} = 0.11_{(2)} = 1.1\text{E} - 1$

Obě čísla převedeme na stejný exponent 4 a sečteme:

```
  1.11111
  0.000011
-----------
 10.000001
```

Výsledné číslo musíme znormalizovat zvýšením exponentu na 5 a tím posunutím desetinné čárky vlevo:

$10.000001\text{E}4 = 1.0000001\text{E}5$

Toto číslo je reprezentací čísla 32.25.

IEEE-754 – Násobení

Násobení dvou reálných čísel:

1. Exponent výsledku je součet exponentů čísel
2. Mantisa výsledku je součin mantis čísel (i se skrytými jedničkami)
3. Výsledné číslo normalizovat
   - podle výsledku násobení mantis je někdy nutné zvýšit exponent o 1 a vyrotovat mantisu doprava

IEEE-754 – Násobení

Příklad: Vynásobíme čísla $0.375 \cdot 1.5$

$0.375_{(10)} = 0.011_{(2)} = 1.1\text{E} - 2 \qquad 1.5_{(10)} = 1.1_{(2)} = 1.1\text{E}0$

Součin mantis je:

```
    11 odpovídá 1.1
   *11 odpovídá 1.1
---------------
    11
   11
---------------
  1001 odpovídá 10.01 protože výsledek má dvě desetinná čísla
```

Exponent vychází $-2 + 0 = -2$, ale výsledek součinu mantis potřebujeme normalizovat, neboli zvětšit exponent o 1 na -1.

Tím dostáváme výsledek $10.01\text{E} - 2 = 1.001\text{E} - 1$ což je správný výsledek $0.5625_{(10)}$

IEEE-754

Reálná čísla shrnutí:

- reálná čísla s plovoucí čárkou umí reprezentovat čísla ve velmi velkém rozsahu:
  - float – absolutní hodnota čísel od $1.175494351E-38$ do $3.402823466E+38$
  - double – absolutní hodnota čísel od $2.2250738585072014E-308$ do $1.7976931348623158E+308$
- přesnost čísla se udává na počet validních cifer:
  - float – v desítkové soustavě 6-7 platných cifer
  - double – v desítkové soustavě 15-16 platných cifer

POZOR: Následující `while` cyklus neskončí:

```
float a=1.0, step=5e-8;
while (a*a<1.01) {
  a+=step;
}
```