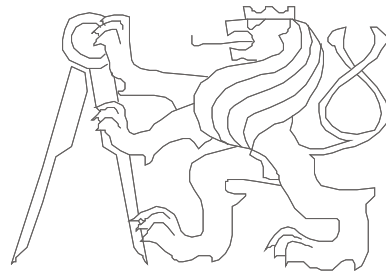


Architektury počítačů

Počítačová aritmetika

Miroslav Šnorek, Michal Štepanovský, Pavel Píša



České vysoké učení technické, Fakulta elektrotechnická

Důležitá poznámka úvodem

- Nechceme vás učit počítač navrhnout, ale
- Porozumět jeho struktuře, abyste mohli lépe využít jeho možností k dosažení jeho vyššího výkonu.
- Tento předmět je zejména o HW/SW interfejsu.
- Vychází ze světově uznávané knihy autorů
- Paterson, D., Henessy, V.: Computer Organization and Design, The HW/SW Interface. Elsevier, ISBN: 978-0-12-370606-5
- Stránky předmětu:
<https://cw.fel.cvut.cz/wiki/courses/a0b36apo/start>

Hodnocení a podmínky absolvování

Zápočet:

| Kategorie | Body | Nutné minimum |
|-----------------------------|-----------|---------------|
| 4 domácí úkoly | 24 | 10 |
| Vstupní test (1. přednáška) | 4 | 0 |
| Hlavní test (9. přednáška) | 20 | 10 |
| Týmový projekt | 20 | 5 |
| Celkem | 68 | |

Zkouška:

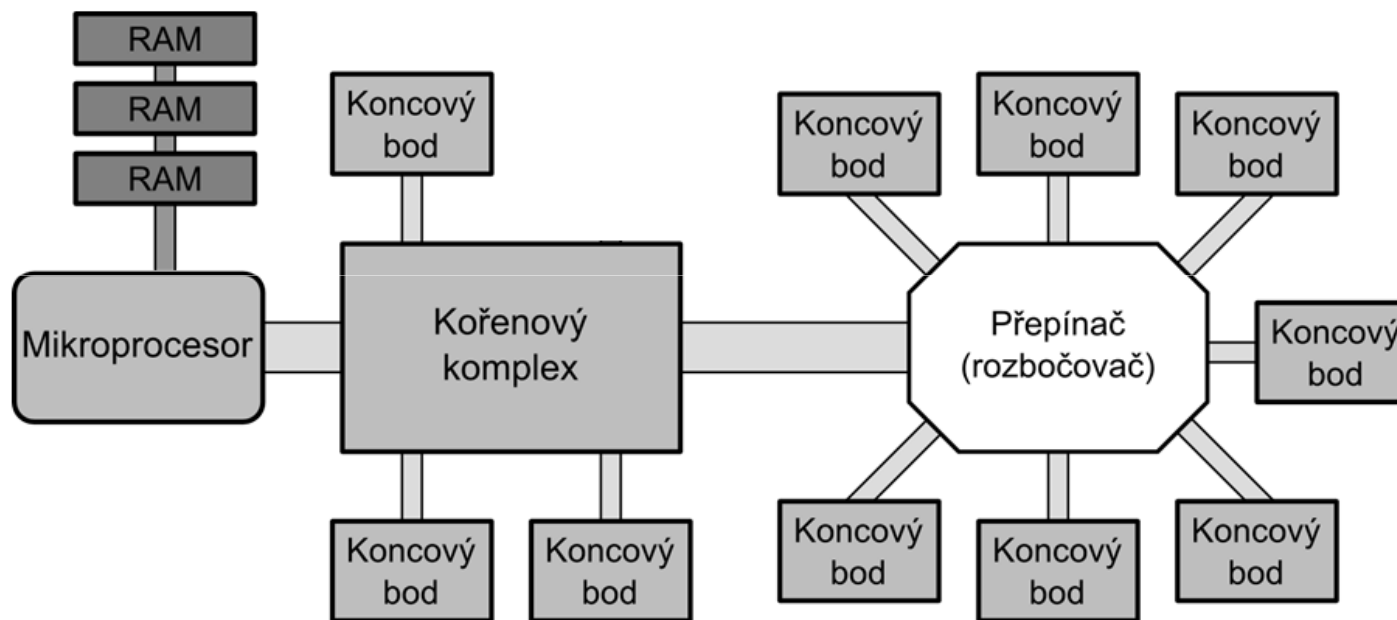
| Kategorie | Body | Nutné minimum |
|----------------------|--------|---------------|
| Písemná část zkoušky | 30 | 15 |
| Ústní část zkoušky | +/- 10 | 0 |

| Známka | Bodové rozmezí |
|--------|----------------|
| A | 90 a více |
| B | 80 - 89 |
| C | 70 - 79 |
| D | 60 - 69 |
| E | 50 - 59 |
| F | méně než 50 |

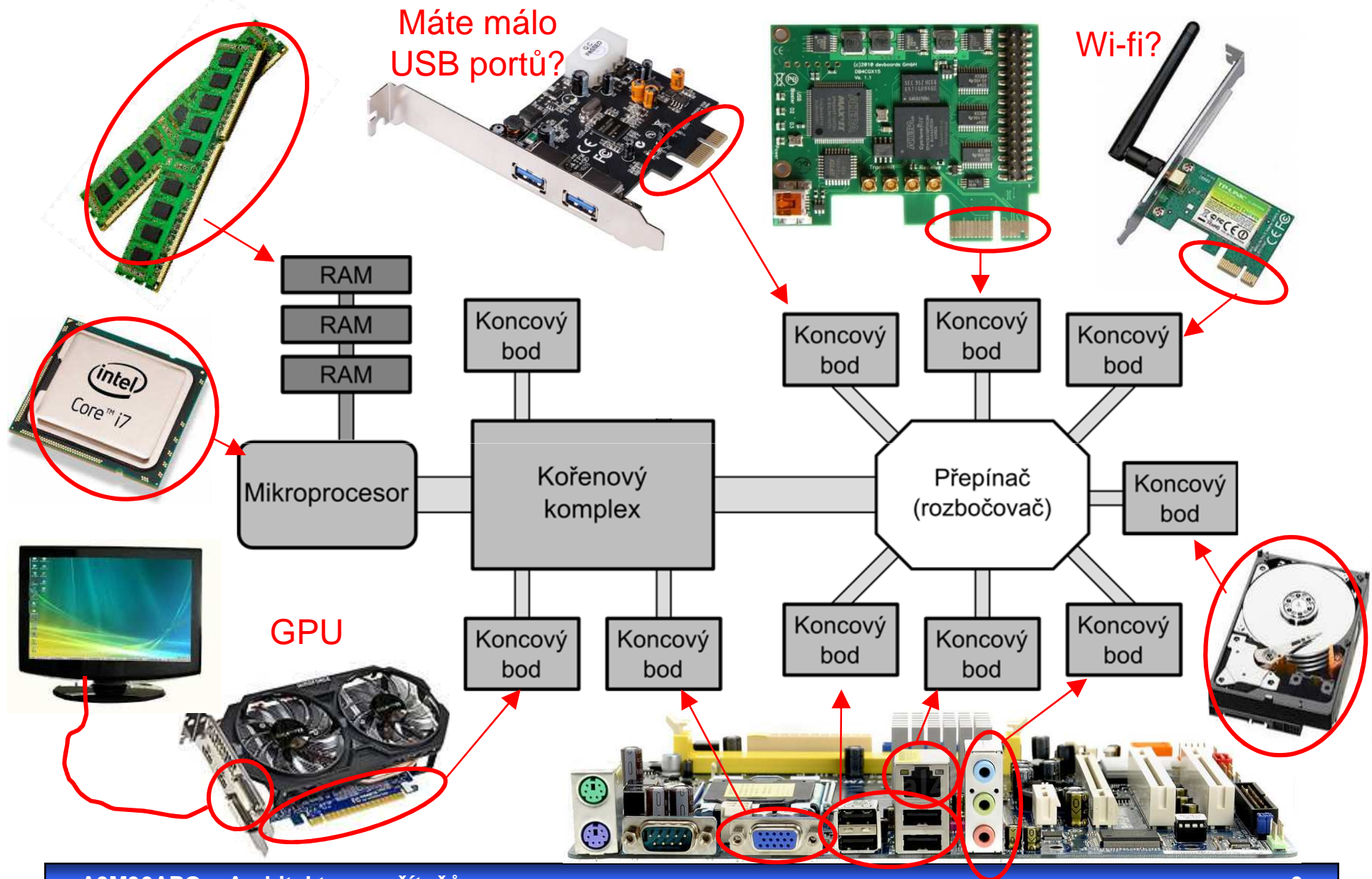
Architektura dnešního PC ???



Architektura dnešního PC



Architektura dnešního PC

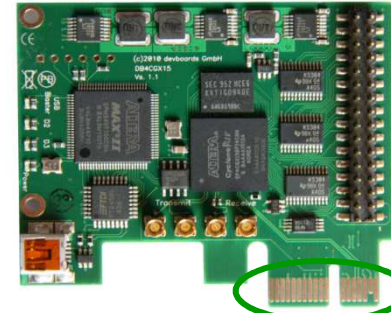


Přehled témat přednášek

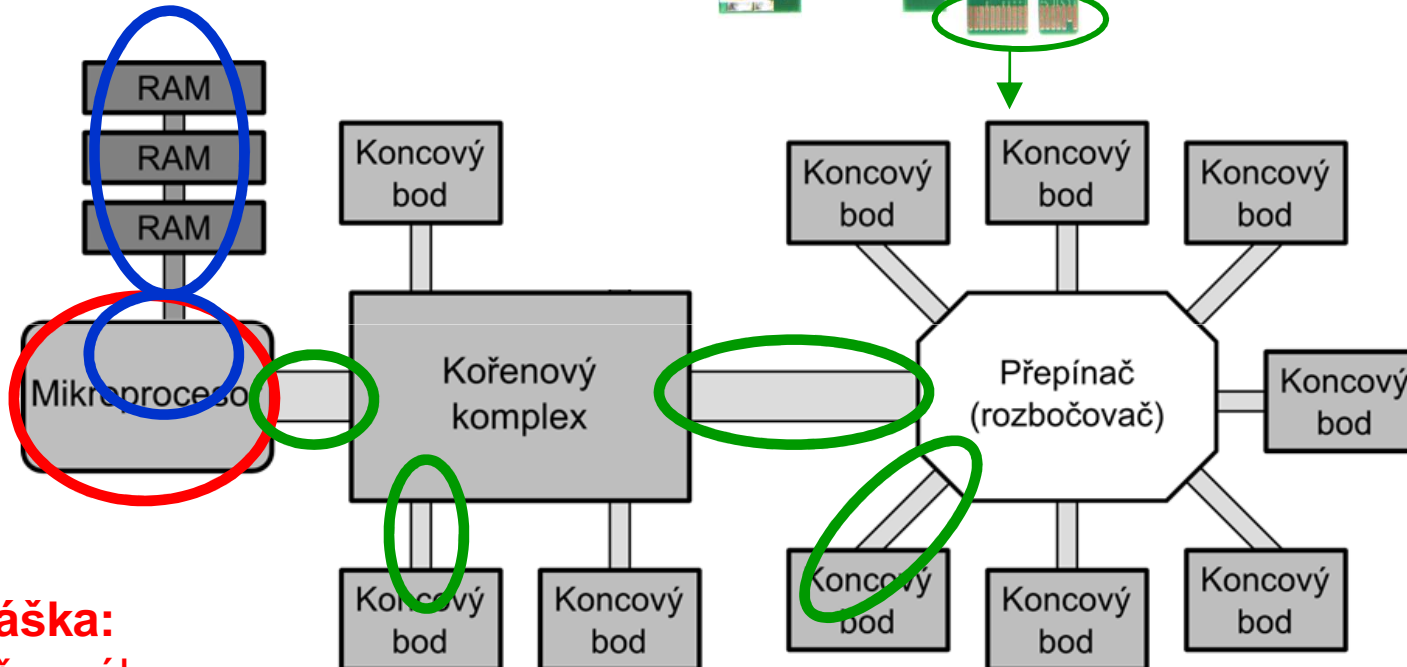
3. přednáška:
Paměťový subsystém –
hierarchie pamětí. Statická
a dynamická paměť

4. přednáška:
Paměťový
subsystém –
virtuální paměť

7. přednáška:
Vstupně/výstupní
podsytem. SW
pohled.



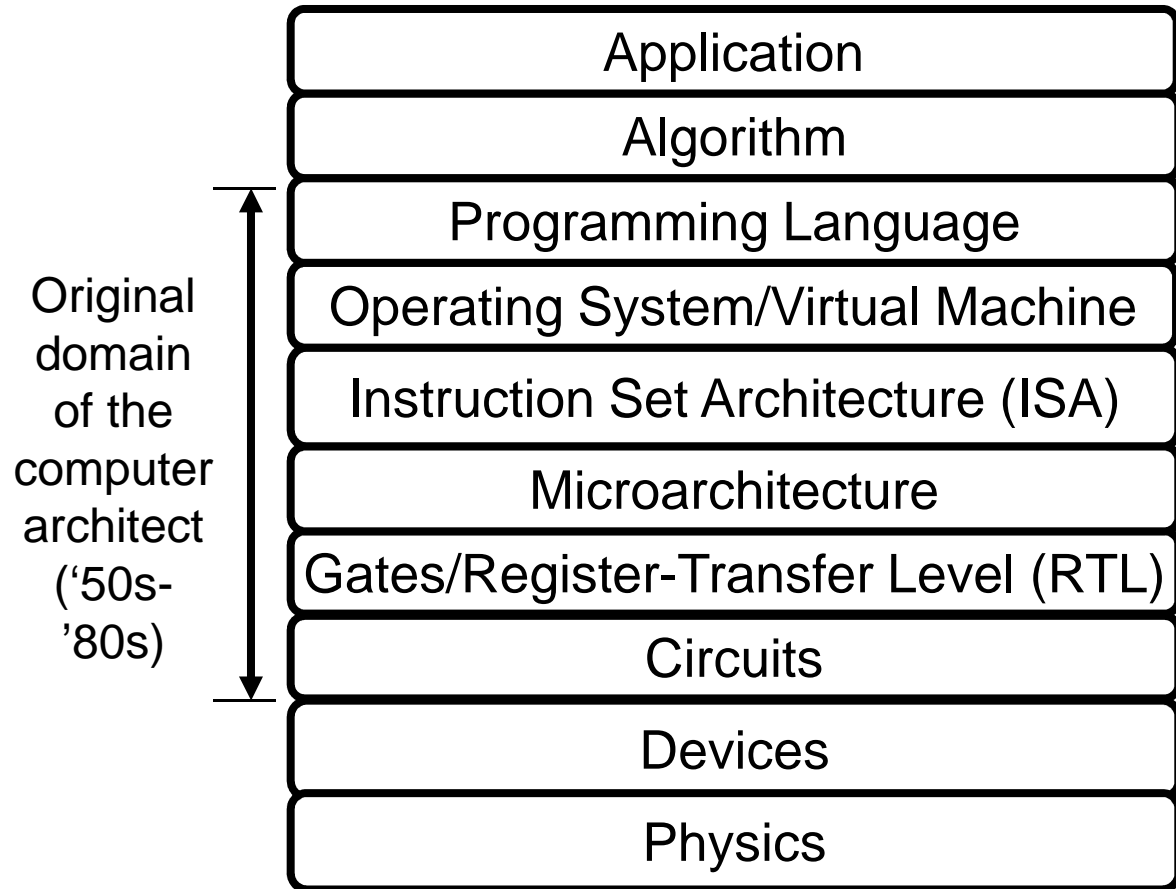
2. přednáška:
Návrh
jednoduchého
CPU,
Vykonávání
instrukcí,
Funkce řadiče



5. přednáška:
Princip zřetězeného
zpracování instrukcí,
Řešení hazardů uvnitř
CPU

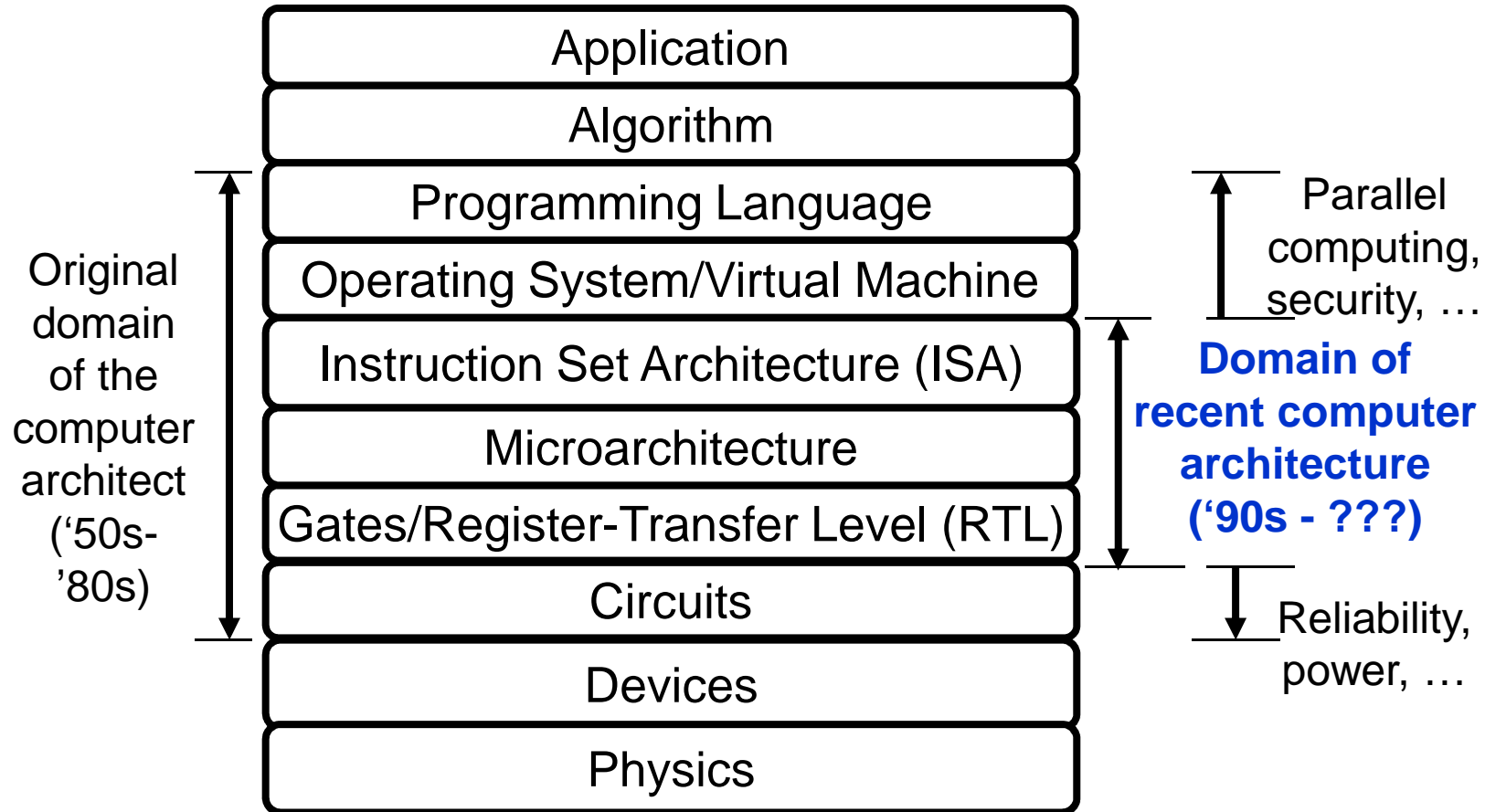
6. přednáška:
Vstupně/výstupní
podsytem. HW pohled.
Sběrnice PCI, propojení PCIe, USB, SerialATA,
HyperTransport, QuickPath interconnect

Co je to architektura počítače?



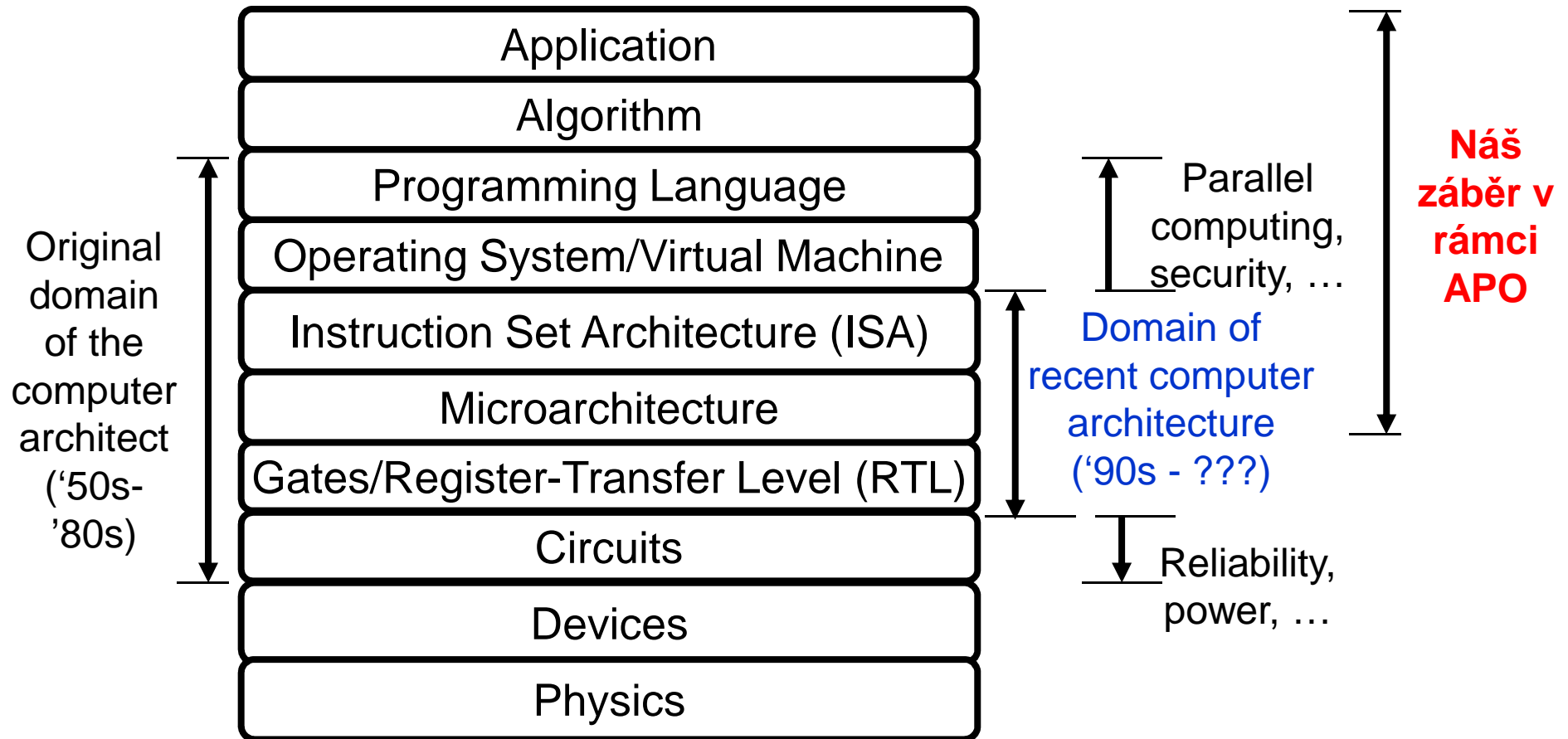
Reference: John Kubiatowicz: EECS 252 Graduate Computer Architecture, Lecture 1. University of California, Berkeley

Co je to architektura počítače?



Reference: John Kubiatowicz: EECS 252 Graduate Computer Architecture, Lecture 1. University of California, Berkeley

Co je to architektura počítače?



Reference: John Kubiatowicz: EECS 252 Graduate Computer Architecture, Lecture 1. University of California, Berkeley

Přehled témat přednášek

8. Technické a organizační prostředky (vnější události, výjimky, reálný čas)
9. Sítě procesorů a počítačů
10. Předávání parametrů funkcím a virtuálním instrukcím operačního systému
11. Klasická registrově orientovaná architektura CISC (M68000)
12. Procesorová rodina INTEL x86, Od 8086 k EMT64
13. Přehled vývoje architektury a koncepcí CPU (RISC/CISC)
14. Víceúrovňový model počítače, virtualizace

Proč je potřeba studovat i nízkoúrovňovou konstrukci počítače

- Pro návrh nové počítačové/procesorové architektury
- Pro implementaci vybrané architektury v integrovaném obvodu/FPGA
- Pro obvodový návrh hardware/systemu (velké nebo vestavné systémy)
- Pro porozumění obecným otázkám a problémům ohledně počítačů, jejich architektur a výkonnosti
- **Pro to jak efektivně využívat existující hardware** (to znamená, jak psát kvalitní software)
 - Bez přehledu a pochopení chování, možností, omezení a limitace zdrojů není možné efektivně využít žádný hardware (pro moderní HW s více jádry a výpočetními subsystémy to platí dvojnásob)
 - Určitě lze vytvořit dobře placené programy i bez těchto znalostí, ale budou vyžadovat mnohonásobně silnější hardware a budou plýtvat zdroji. Není však takto možné vytvořit žádné náročné aplikace ať již na výkon nebo na ušetření energie. Přitom to je oblast kde probíhá skutečný vývoj a mají z dlouhodobějšího technologického a i vědeckého pohledu smysl.

Další motivace a příklady

- Předkládané znalosti jsou nutné pro každého programátora, jehož aplikace pracují i jen s větším ovšem dnes běžným množstvím dat nebo vyžadují netriviální výpočetní výkon
- Žádná práce s multimédií nemůže být vykonána dobře bez takovýchto znalostí
- 1/3 našeho kurzu je zaměřená i na přístup k periferiím
- Příklady
 - Facebook – HipHop for PHP -> C++/GCC -> machine code
 - BlackBerry (RIM) – our consultations for time source
 - RedHat – JAVA JIT for ARM for future servers generation
 - Multimedia and CUDA computations
 - Photoshop, GIMP (organizace dat v paměti)
 - Knot-DNS (RCU, Copy on write, Cuckoo hashing,)

Obsah 1. přednášky

- Jak se v počítači ukládají
 - Čísla typu INTEGER, bez i se znaménkem,
 - Čísla typu REAL,
 - Hodnoty typu LOGICAL?
- Jak se realizují základní operace
 - Sčítání, odčítání,
 - Posuny,
 - Násobení, dělení,

MOTIVACE: Co program vytiskne?

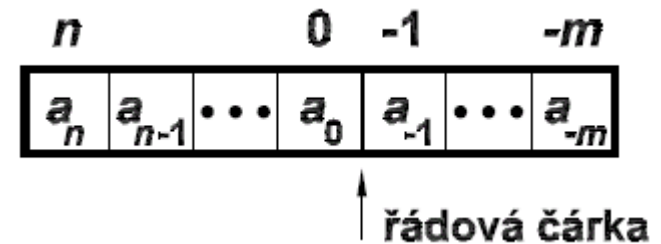
```
int main() {  
    int a = -200;  
    printf("hodnota: %u = %d = %f = %c \n", a,  
        a, *((float*)&a), a);  
  
    return 0;  
}
```

hodnota: 4294967096 = -200 = nan = 8

0x38 0xff 0xff 0xff

Základní terminologie

- **Poziční číselná soustava,**
- Řádová čárka, nebo lépe tečka,
 - z základ číselné soustavy,
- Nejmenší zobrazitelné číslo,
- **Modul** = \mathcal{Z} , číslo o jedničku větší, než je největší zobrazitelné číslo v dané řádové mřížce,
- Zobrazitelná čísla,
 - k je celé číslo.



$$\epsilon = z^{-m}$$

$$0 \leq A = k \cdot \epsilon < \mathcal{Z}$$

$$A \sim a_n a_{n-1} \dots a_0, a_{-1} \dots a_{-m}$$

$$A = a_n z^n + a_{n-1} z^{n-1} + \dots + a_0 + a_{-1} z^{-1} \dots a_{-m} z^{-m}$$

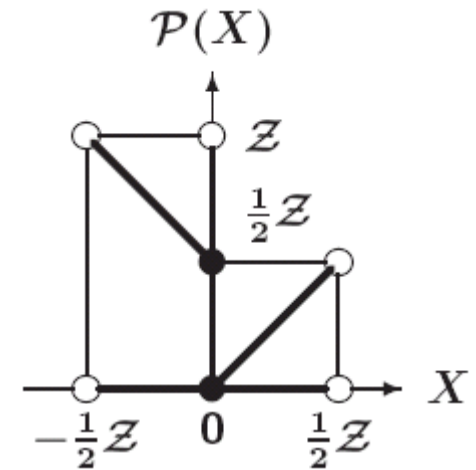
Uložení čísel typu INTEGER



- Kolik je různých stavů položky HODNOTA ?
 - $2^8 = 256D$ (desítkově). To není mnoho, že?
 - Řešení: proč nepoužít více bajtů?
 - $4B = 2^{32} = 4\,294\,967\,296D$,
 - $8B = 2^{64}$ = (spočítejte si sami).
- Čísla je ale třeba většinou ukládat se znaménkem.
- Při vhodném kódování můžeme ušetřit HW.
- Například odčítání lze provést obvody na sčítání (sčítačkou).

Číslo INTEGER se znaménkem

- Znaménko a hodnota. Jde o tzv. **přímý kód**.
 - +1234, -5678.
- Takhle vyjadřujeme čísla v matematice běžně.
- Nevýhoda: v počítači máme pro zobrazení k dispozici jen binární číslice 0, 1.
- Běžně dodržovaná dohoda:
 - $0 \approx +$, $1 \approx -$.
 - Nevýhoda: jinak se musí při aritmetických operacích pracovat se znaménkovým bitem, jinak s bity hodnoty.
- Jiná nevýhoda: máme 2 různá vyjádření nuly.

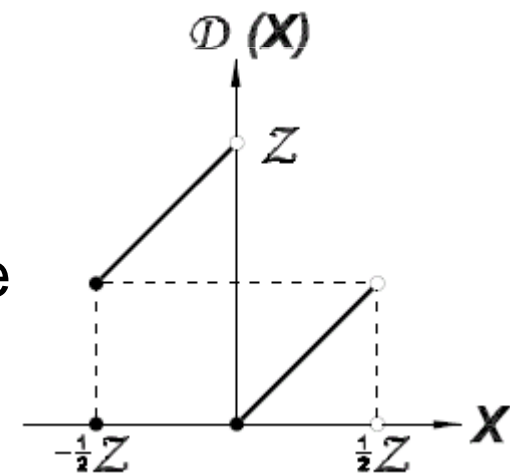


HW/SW interfejs

- V programovacím jazyce C se celým číslem bez znaménka říká *unsigned integers* a v programu se deklarují jako `unsigned int`.
- Těm se znaménkem se říká *integers* a v programu se deklarují jako `signed int`.

Čísla INTEGER se znaménkem II.

- Jiná možnost: část (typicky polovinu) stavů položky HODNOTA použijeme pro reprezentaci čísel kladných, druhou polovinu pro reprezentaci čísel záporných.
- Méně výhodná varianta: **inverzní kód**,
- Výhodnější varianta: **dvojkový doplněk**.
- Proč výhodnější? Úspora v HW!
- Poznámka: oba kódy vůbec nejsou omezeny na dvojkovou soustavu, stejně je můžeme použít např. v soustavě desítkové!
- Na dalším slajdu jsou příklady:



Doplňkový kód - příklady

- Rozumí se **dvojkový doplněk**. Příklady reprezentací:
 - $0_D = 00000000_H$,
 - $1_D = 00000001_H$,
 - $2_D = 00000002_H$,
 - $3_D = 00000003_H$,
 - $-1_D = FFFFFFFF_H$,
 - $-2_D = FFFFFFFE_H$,
 - $-3_D = FFFFFFFD_H$,
- Analogii dvojkového doplněku se v soustavě s jiným základem říká doplněk do modulu. Stejně se postupuje třeba v soustavě desítkové (doplněk do „10“).
- Všimněte si: součet dvou opačných čísel se stejnou absolutní hodnotou je 00000000_H .
- Přenos do vyššího řádu (32.) ignorujeme. Sčítáme vlastně $\text{mod } 2^{32}$.
- Jak je to vlastně v tomto kódování s **přeplněním** řádové mřížky (přetečením)? Budeme diskutovat později...

Existuje i jedničkový doplněk

- Jedničkový se od dvojkového doplňku se liší málo, jen o jedničku, o tzv. **horkou jedničku** (Hot-One), kterou přičítáme k nejnižšímu řádu.
- Jedničkový doplněk čísla se (ve dvojkové soustavě) vytvoří bitovou negací.
- Jde o tzv. **inverzní kód**.

Příklad sčítání a odčítání v doplňkovém kódu

- **Sčítání**

- $0000000\ 0000\ 0111_B \approx 7_D$ Vysvětl.: $0=0_H, 0=0_B$

- $+ 0000000\ 0000\ 0110_B \approx 6_D$

- $0000000\ 0000\ 1101_B \approx 13_D$

- **Odčítání** je v tomto kódu přičítáním čísla opačného

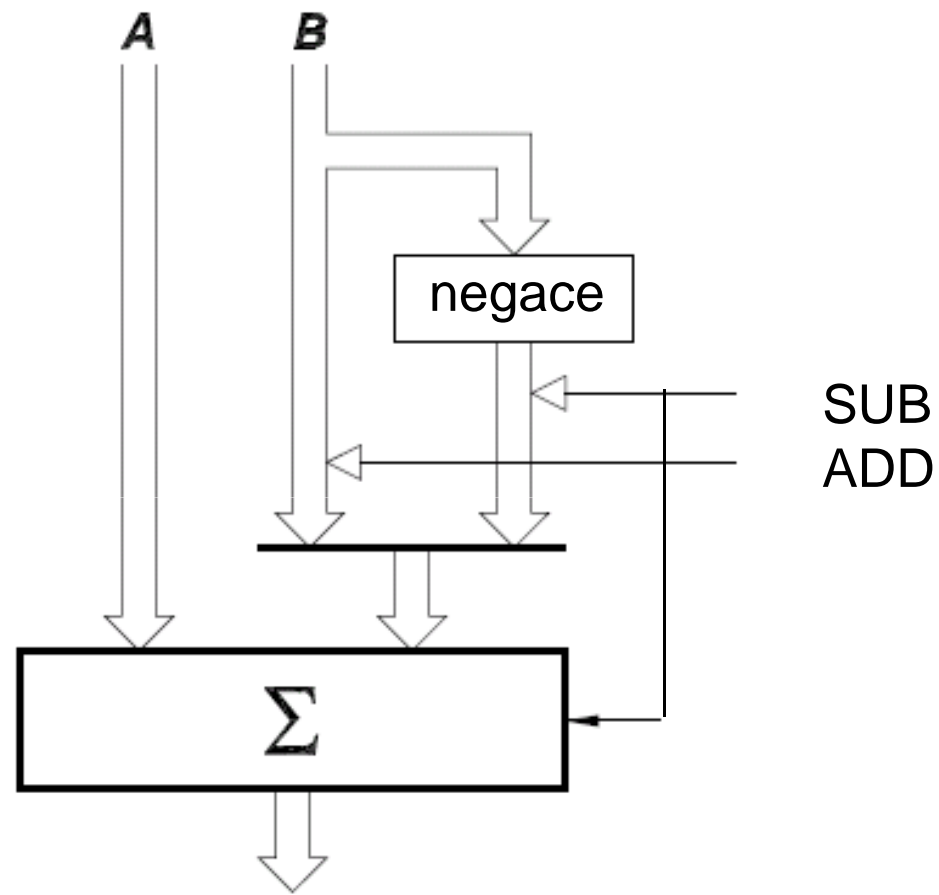
- $0000000\ 0000\ 0111_B \approx 7_D$

- $+ FFFFFFFF1111\ 1010_B \approx -6_D$

- $0000000\ 0000\ 0001_B \approx 1_D$

- **Otázka k opakování:** jak jsme vytvořili k číslu ve dvojkové soustavě jeho číslo opačné?

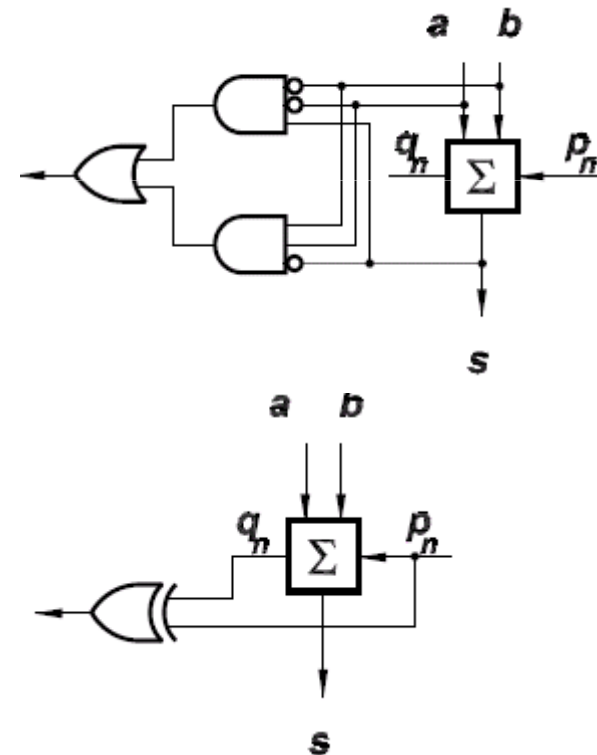
Sčítací/odčítací HW



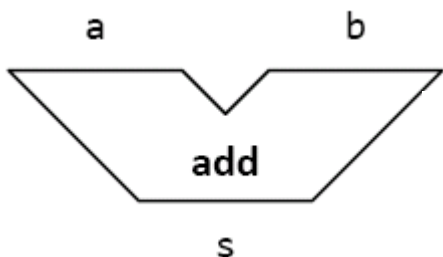
Inspirace: X36JPO, A. Pluháček

Rozumíte pojmu přeplnění?

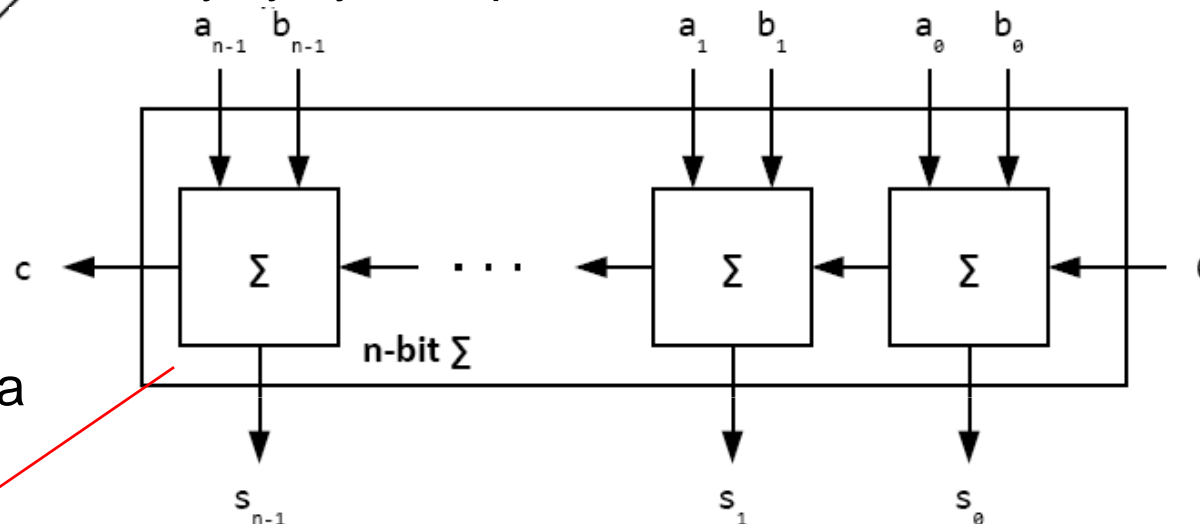
- Říká se tomu také **přetečení**.
- Situace, kdy výsledek operace není správný, protože se nevešel do dané řádové mřížky.
- Ale pozor! V této reprezentaci nenastává, generuje-li se přenos z nejvyššího řádu!
- **Nastává v situaci, kdy znaménko výsledku je jiné, než znaménka operandů, byla-li stejná, nebo**
- Nonekvivalencí přenosu do a z nejvyššího řádu.



Sčítací HW blokově

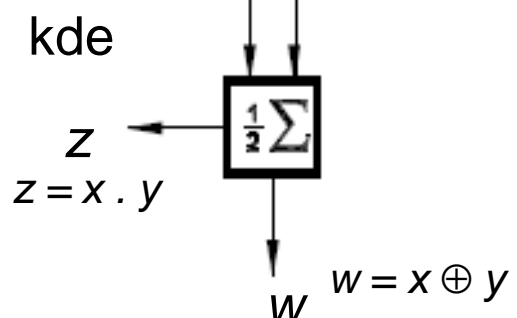
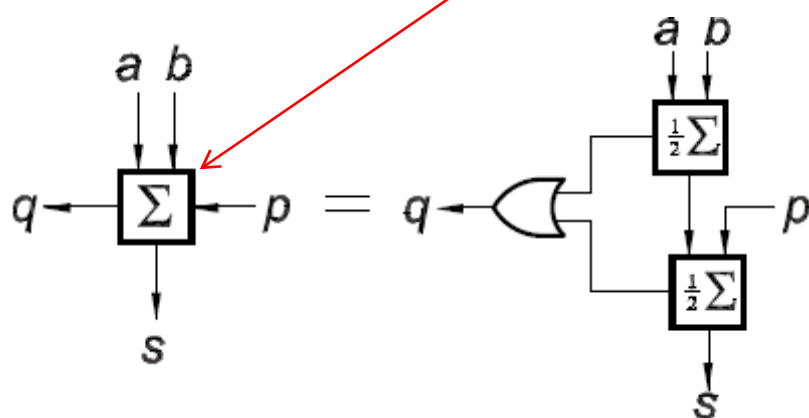


Obvyklý symbol pro funkční blok



Vnitřní struktura

Realizace

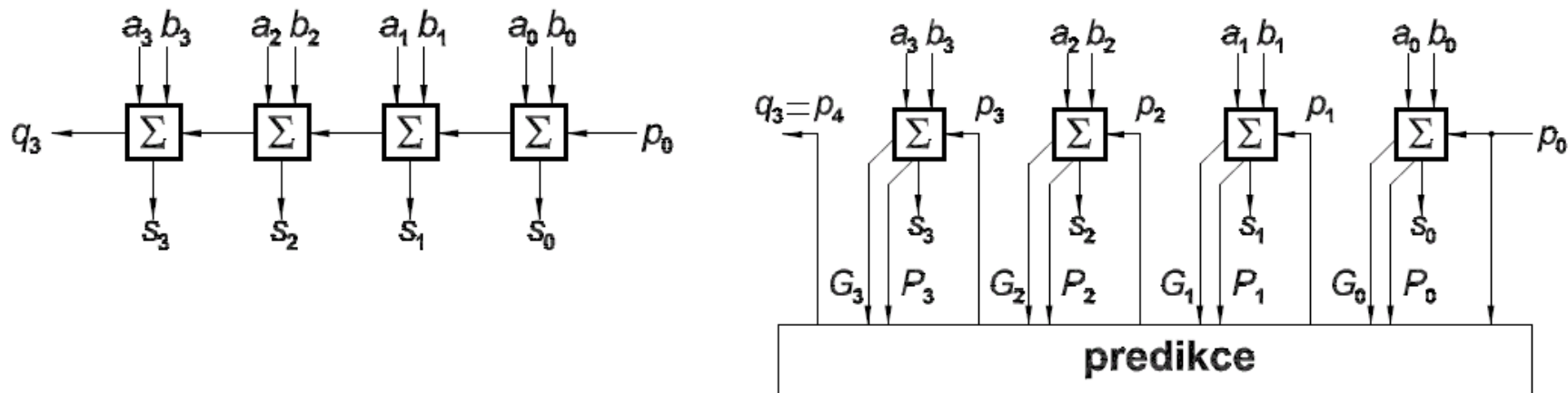


Poznámka k realizaci rychlé paralelní sčítačky

- Paralelní sčítačka s minimálním zpožděním (kombinační obvod) je prakticky nerealizovatelná.
- Pro 64-bitovou verzi bychom potřebovali 10^{20} hradel.
- Proto se staví tak, jak jsme viděli na předchozím slajdu, s postupným přenosem a
- ze dvou pulsčítaček, nebo...

Všimněte si:

- Paralelní sčítačka je **kombinační obvod**. Má tedy smysl mluvit o její rychlosti (ve smyslu: je dost rychlá?)
- Právě naopak, je pomalá! Vysvětlete!
- Zrychlení? Sčítačka s predikcí přenosů - CLA!



CLA?

- CLA – Carry Look-Ahead.
- Sčítačka CLA nabízí dostatečné zrychlení v porovnání se sčítačkou s postupným přenosem při přijatelném nárůstu ceny HW.
- 64-bitová verze zvýší cenu HW o necelých 50%, ale rychlost se zvýší 9:1.
- To představuje významné zvýšení poměru *rychlost/cena*).

Tyto rovnice jsou klíčové pro pochopení principu...

Označme:

- generování přenosu:

$$g_j = x_j y_j$$

- šíření přenosu (propagation):

$$p_j = x_j \oplus y_j = x_j \bar{y}_j \vee \bar{x}_j y_j$$

Pak:

- součet v j-tem řádu:

$$s_j = c_j (\overline{x_j \oplus y_j}) \vee \bar{c}_j (x_j \oplus y_j) = c_j \bar{p}_j \vee \bar{c}_j p_j = p_j \oplus c_j$$

- přenos do vyššího (j+1) řádu:

$$c_{j+1} = x_j y_j \vee (x_j \oplus y_j) c_j = g_j \vee p_j c_j$$

CLA

Takže platí: $c_1 = g_0 \vee p_0 c_0$

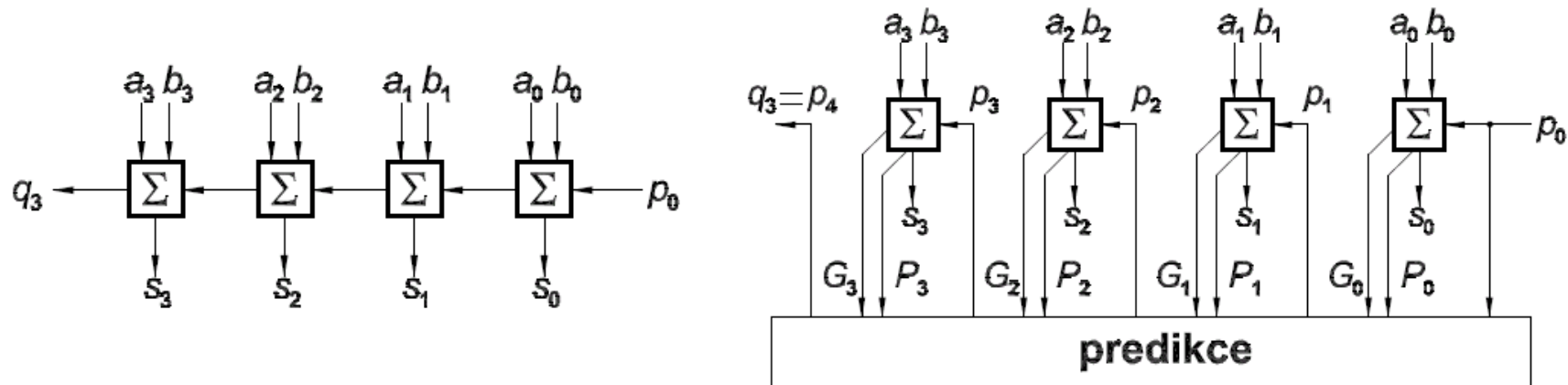
$c_2 = g_1 \vee p_1 c_1 = g_1 \vee p_1 (g_0 \vee p_0 c_0) = g_1 \vee p_1 g_0 \vee p_1 p_0 c_0$

$c_3 = g_2 \vee p_2 c_2 = g_2 \vee p_2 (g_1 \vee p_1 g_0 \vee p_1 p_0 c_0) = g_2 \vee p_2 g_1 \vee p_2 p_1 g_0 \vee p_2 p_1 p_0 c_0$

$c_4 = g_3 \vee p_3 c_3 = \dots = g_3 \vee p_3 g_2 \vee p_3 p_2 g_1 \vee p_3 p_2 p_1 g_0 \vee p_3 p_2 p_1 p_0 c_0$

$c_5 = \dots$

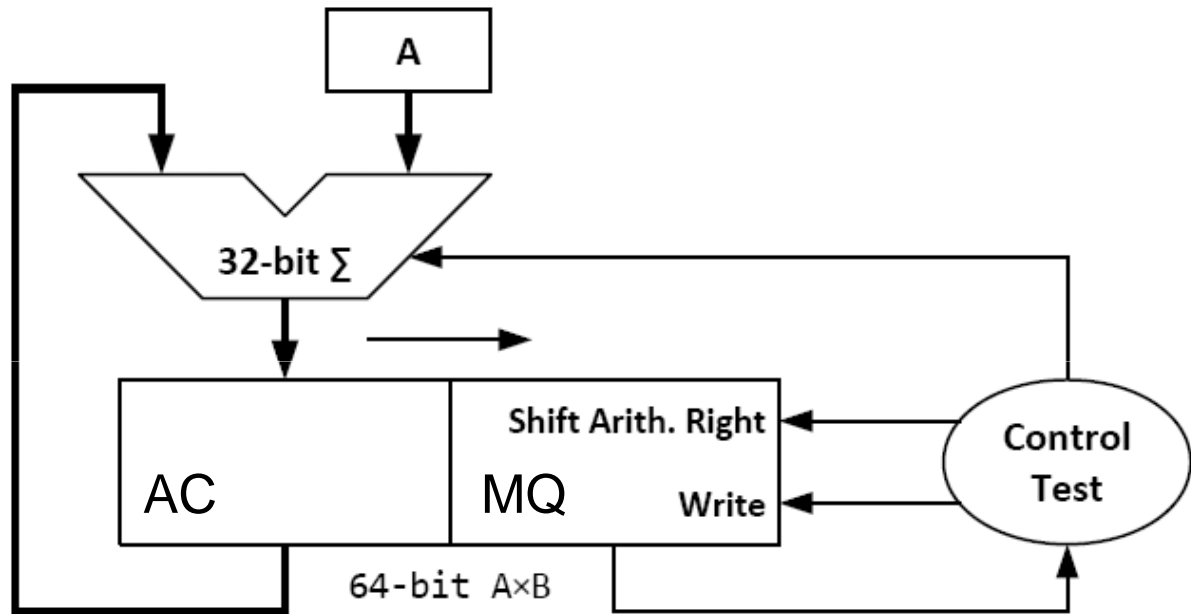
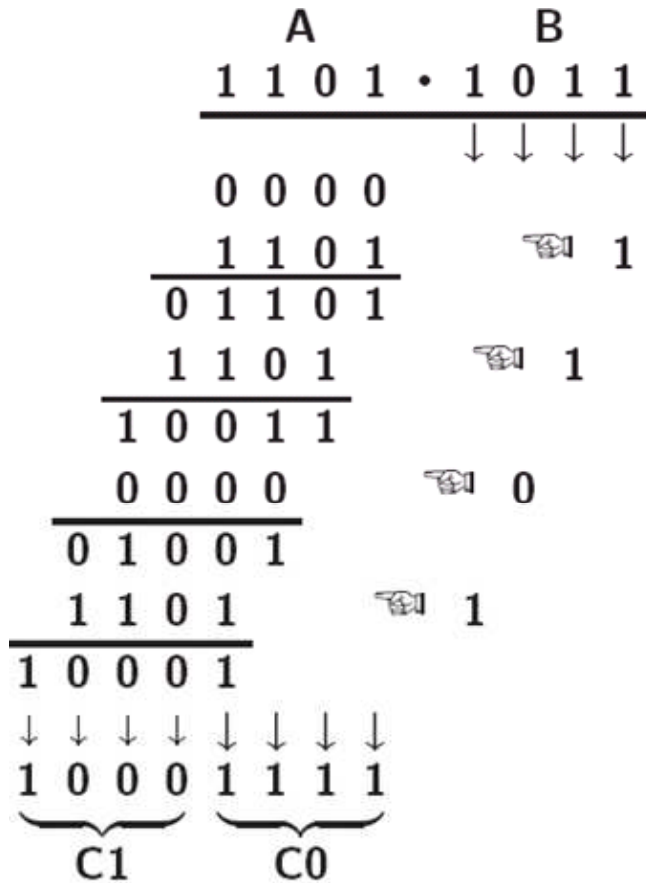
Například rovnici pro c_3 je možné číst následovně: Přenos do 3. řádu nastane, **pokud** přenos byl generován v 2. řádu, **nebo** se 2. řádem šíří a byl generován v 1. řádu, **nebo** se šíří 2. a 1. řádem a byl generován 0-tým řádem, **nebo** se šíří druhým, prvním a 0-tým řádem a byl v c_0 ($c_0=1$).



Násobení binárních čísel bez znaménka – pro připomenutí

$$\begin{array}{r}
 \begin{array}{cc}
 \text{A} & \text{B} \\
 1\ 1\ 0\ 1 & \cdot\ 1\ 0\ 1\ 1 \\
 \hline
 0\ 0\ 0\ 0 & \downarrow\ \downarrow\ \downarrow\ \downarrow \\
 1\ 1\ 0\ 1 & \leftarrow 1 \\
 \hline
 0\ 1\ 1\ 0\ 1 & \\
 1\ 1\ 0\ 1 & \leftarrow 1 \\
 \hline
 1\ 0\ 0\ 1\ 1 & \\
 0\ 0\ 0\ 0 & \leftarrow 0 \\
 \hline
 0\ 1\ 0\ 0\ 1 & \\
 1\ 1\ 0\ 1 & \leftarrow 1 \\
 \hline
 1\ 0\ 0\ 0\ 1 & \\
 \downarrow\ \downarrow\ \downarrow\ \downarrow & \downarrow\ \downarrow\ \downarrow\ \downarrow \\
 \underbrace{1\ 0\ 0\ 0}_{C1} & \underbrace{1\ 1\ 1\ 1}_{C0}
 \end{array}
 \end{array}$$

Sekvenční HW násobička (varianta 32b)



Diskuze o rychlosti: ta je ale pomalá, co?

Algoritmus

```
A = násobenec;  
MQ = násobitel;  
AC = 0;
```

```
for( int i=1; i <= n; i++) // n je počet bitů  
{  
    if(MQ0 == 1) AC = AC + A; // MQ0 = nejnižší bit MQ
```

```
    SR (posuň registr AC MQ o jedno místo doprava a doplň  
    případný přenos z nejvyššího řádu z předchozího kroku)  
}  
end.
```

Nyní je výsledek v AC MQ.

Příklad x.y

Násobenec $x=110$ a násobitel $y=101$.

| i | operace | AC | MQ | A | komentář |
|----------|----------------|-----------|-----------|----------|--------------------|
| | | 000 | 101 | 110 | prvotní nastavení |
| 1 | AC = AC+MB | 110 | 101 | | začátek cyklu |
| | SR | 011 | 010 | | |
| 2 | nic | 011 | 010 | | protože $MQ_0 = 0$ |
| | SR | 001 | 101 | | |
| 3 | AC = AC+MB | 111 | 101 | | |
| | SR | 011 | 110 | | konec cyklu |

Tedy: $x \times y = 110 \times 101 = 011110$, ($6 \times 5 = 30$)

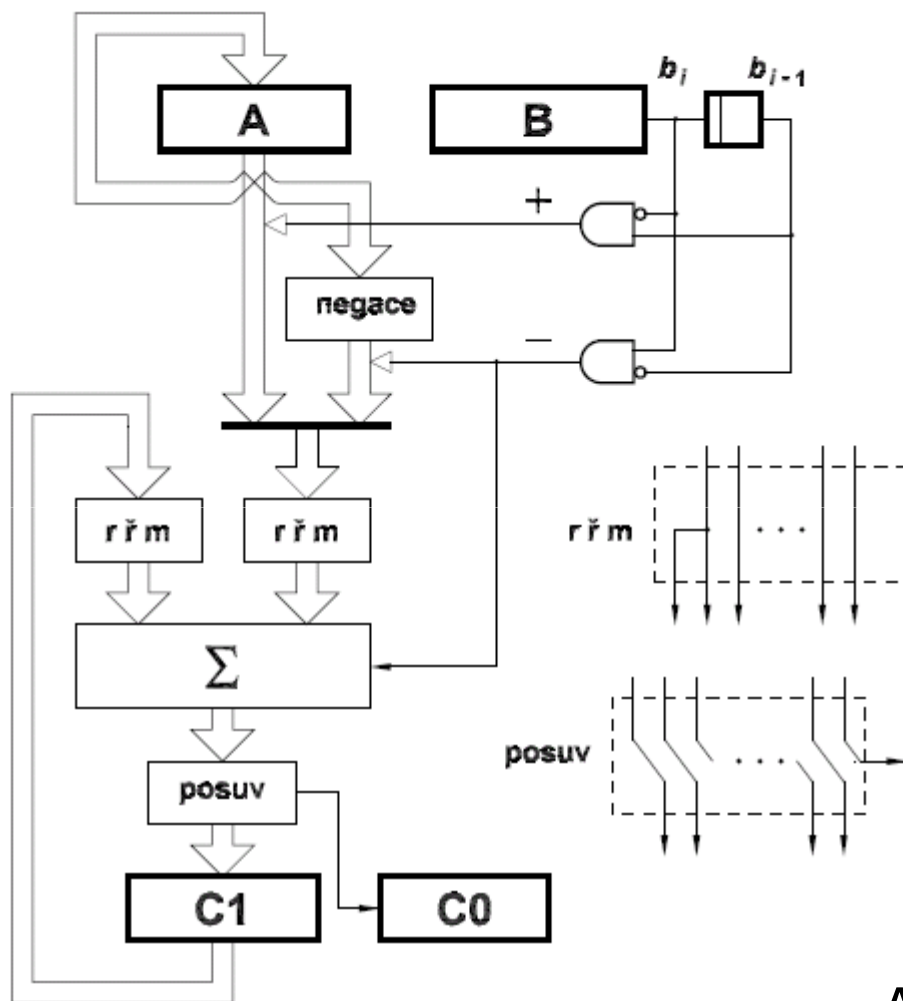
Násobení v doplňkovém kódu

- Lze realizovat, ale je tu problém...
 - **Obraz součinu obecně není roven součinu obrazů!**
- Řešení spočívá v modifikaci dvojkové soustavy na soustavu s relativními číslicemi $0, 1, \hat{1} = -1$
- Podrobnosti už jsou mimo zamýšlený rozsah APO.

$$\begin{aligned} \text{př.: } 1\hat{1}10 &\sim 1 \cdot 8 + (-1) \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 6 \\ \hat{1}1\hat{1}0 &\sim (-1) \cdot 8 + 1 \cdot 4 + (-1) \cdot 2 + 0 \cdot 1 = -6 \\ 110 &= 10\hat{1}0 = 1\hat{1}10 \sim 6 \end{aligned}$$

- Nebo v znaménkovém rozšíření na $2N$ bitů a násobení obvyklým způsobem. Z výsledku bereme pouze $2N$ bitů. -> „ruční“ násobení

Násobička v doplňkovém kódu Boothova metoda



APO – jen pro informaci

Rychlá násobička podle Wallaceova stromu

$Q = X \cdot Y$, X a Y necht' jsou 8b čísla

$$(x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0) \cdot (y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0) =$$

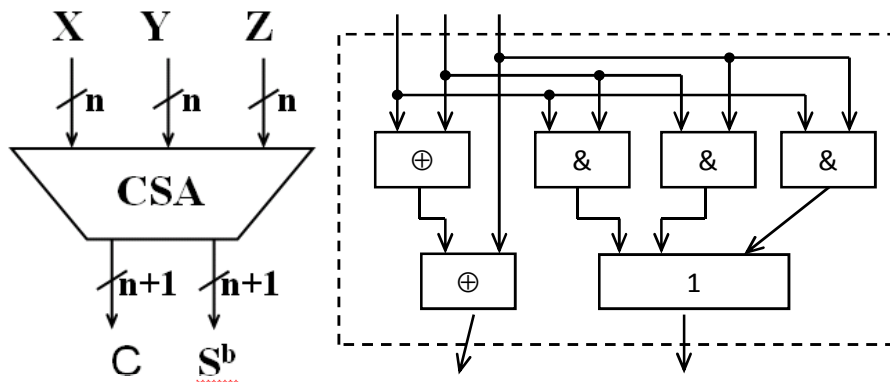
| | | | | | | | | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x_7y_0 | x_6y_0 | x_5y_0 | x_4y_0 | x_3y_0 | x_2y_0 | x_1y_0 | x_0y_0 | P0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | x_7y_1 | x_6y_1 | x_5y_1 | x_4y_1 | x_3y_1 | x_2y_1 | x_1y_1 | x_0y_1 | 0 | P1 |
| 0 | 0 | 0 | 0 | 0 | 0 | x_7y_2 | x_6y_2 | x_5y_2 | x_4y_2 | x_3y_2 | x_2y_2 | x_1y_2 | x_0y_2 | 0 | 0 | P2 |
| 0 | 0 | 0 | 0 | 0 | x_7y_3 | x_6y_3 | x_5y_3 | x_4y_3 | x_3y_3 | x_2y_3 | x_1y_3 | x_0y_3 | 0 | 0 | 0 | P3 |
| 0 | 0 | 0 | 0 | x_7y_4 | x_6y_4 | x_5y_4 | x_4y_4 | x_3y_4 | x_2y_4 | x_1y_4 | x_0y_4 | 0 | 0 | 0 | 0 | P4 |
| 0 | 0 | 0 | x_7y_5 | x_6y_5 | x_5y_5 | x_4y_5 | x_3y_5 | x_2y_5 | x_1y_5 | x_0y_5 | 0 | 0 | 0 | 0 | 0 | P5 |
| 0 | 0 | x_7y_6 | x_6y_6 | x_5y_6 | x_4y_6 | x_3y_6 | x_2y_6 | x_1y_6 | x_0y_6 | 0 | 0 | 0 | 0 | 0 | 0 | P6 |
| 0 | x_7y_7 | x_6y_7 | x_5y_7 | x_4y_7 | x_3y_7 | x_2y_7 | x_1y_7 | x_0y_7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | P7 |
| Q_{15} | Q_{14} | Q_{13} | Q_{12} | Q_{11} | Q_{10} | Q_9 | Q_8 | Q_7 | Q_6 | Q_5 | Q_4 | Q_3 | Q_2 | Q_1 | Q_0 | |

Součtem $P_0 + P_1 + \dots + P_7$ získáme výsledek součinu X a Y .

$$Q = X \cdot Y = P_0 + P_1 + \dots + P_7$$

Rychlá násobička podle Wallaceova stromu

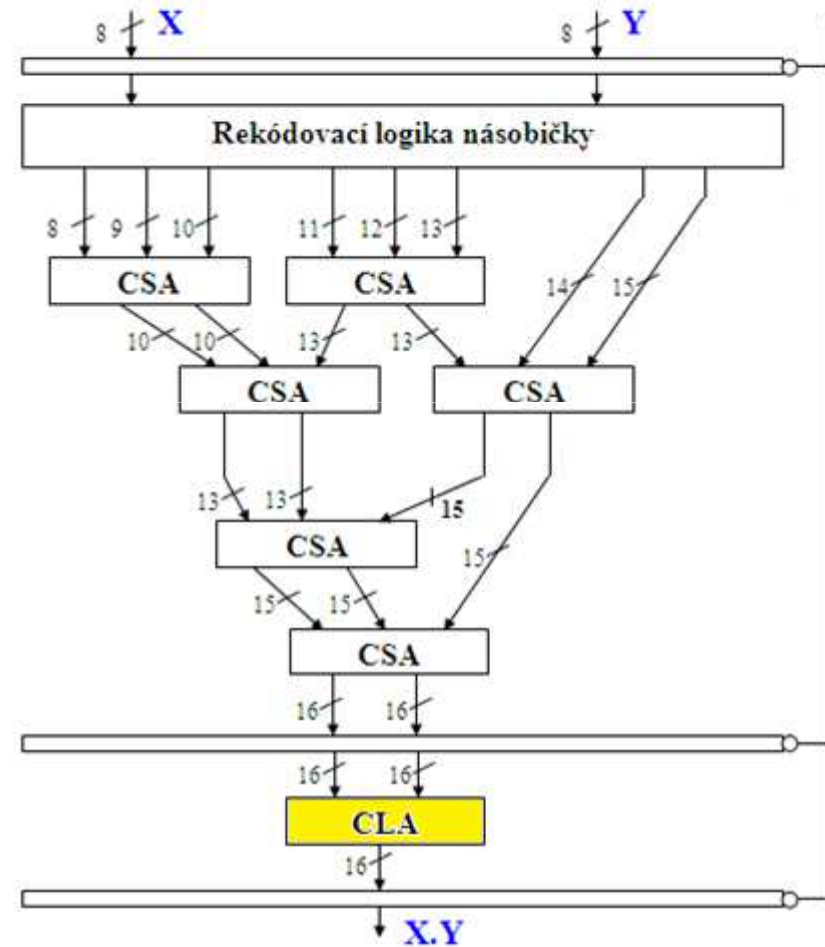
Jejím stavebním prvkem je sčítačka s uchováním přenosu CSA (Carry Save Adder)



$$S = S^b + C$$

$$S_i^b = x_i \oplus y_i \oplus z_i$$

$$C_{i+1} = x_i y_i + y_i z_i + z_i x_i$$



HW dělička – algoritmus dělení

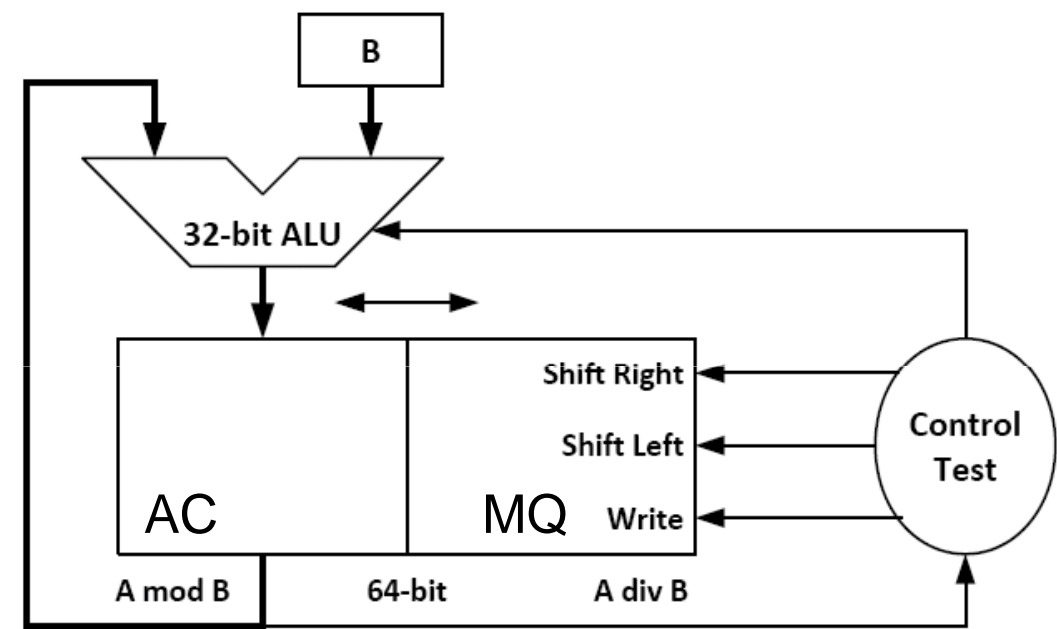
| | | |
|---|----------------------|------------------|
| | 1 1 1 : 0 1 1 | |
| | 0 0 0 1 1 1 | : 0 0 1 1 |
| ⊖ | 1 1 0 0 : : | negace |
| | 1 : : | horká 1 |
| | 0 <u>1 1 1 0</u> : : | - ⇒ 0 |
| | ↓ ↓ ↓ ↓ : | |
| | 1 1 0 1 : | |
| ⊕ | <u>0 0 1 1</u> : | |
| | 1 <u>0 0 0 0</u> 1 | + ⇒ 1 |
| | ↓ ↓ ↓ ↓ | |
| | 0 0 0 1 | |
| ⊖ | 1 1 0 0 | |
| | 1 | |
| | 0 <u>1 1 1 0</u> | - ⇒ 0 |
| ⊖ | <u>0 0 1 1</u> | návrat |
| | 1 <u>0 0 0 1</u> | |
| | 0 0 1 — zbytek | 0 1 0 — podíl |

Sekvenční HW dělička (varianta 32b)

1 1 1 : 0 1 1

dělenec = podíl × dělitel + zbytek

| | | | |
|---|-------------|---|---------|
| | 0 0 0 1 1 1 | : | 0 0 1 1 |
| ⊖ | 1 1 0 0 | : | negace |
| | 1 | : | horká 1 |
| | 0 1 1 1 0 | : | - ⇒ 0 |
| | ↓ ↓ ↓ ↓ | | |
| ⊕ | 1 1 0 1 | : | |
| | 0 0 1 1 | : | |
| | 1 0 0 0 0 1 | : | + ⇒ 1 |
| | ↓ ↓ ↓ ↓ | | |
| ⊖ | 0 0 0 1 | : | |
| | 1 1 0 0 | : | |
| | 1 | : | |
| | 0 1 1 1 0 | : | - ⇒ 0 |
| | 0 0 1 1 | : | návrat |
| ⊖ | 1 0 0 0 1 | : | |
| | 0 0 1 | : | zbytek |



Algoritmus dělení

MQ = dělenec;

B = dělitel; (Podmínka: dělitel různý od 0!)

AC = 0;

```
for( int i=1; i <= n; i++)      {  
    SL (posuň registr AC MQ o jednu pozici vlevo, přičemž vpravo se připíše nula)  
    if(AC >= B) {  
        AC = AC - B;  
        MQ0 = 1;      // nejnižší bit registru MQ se nastaví na 1  
    }  
}
```

→ Nyní registr MQ obsahuje podíl a zbytek je v AC

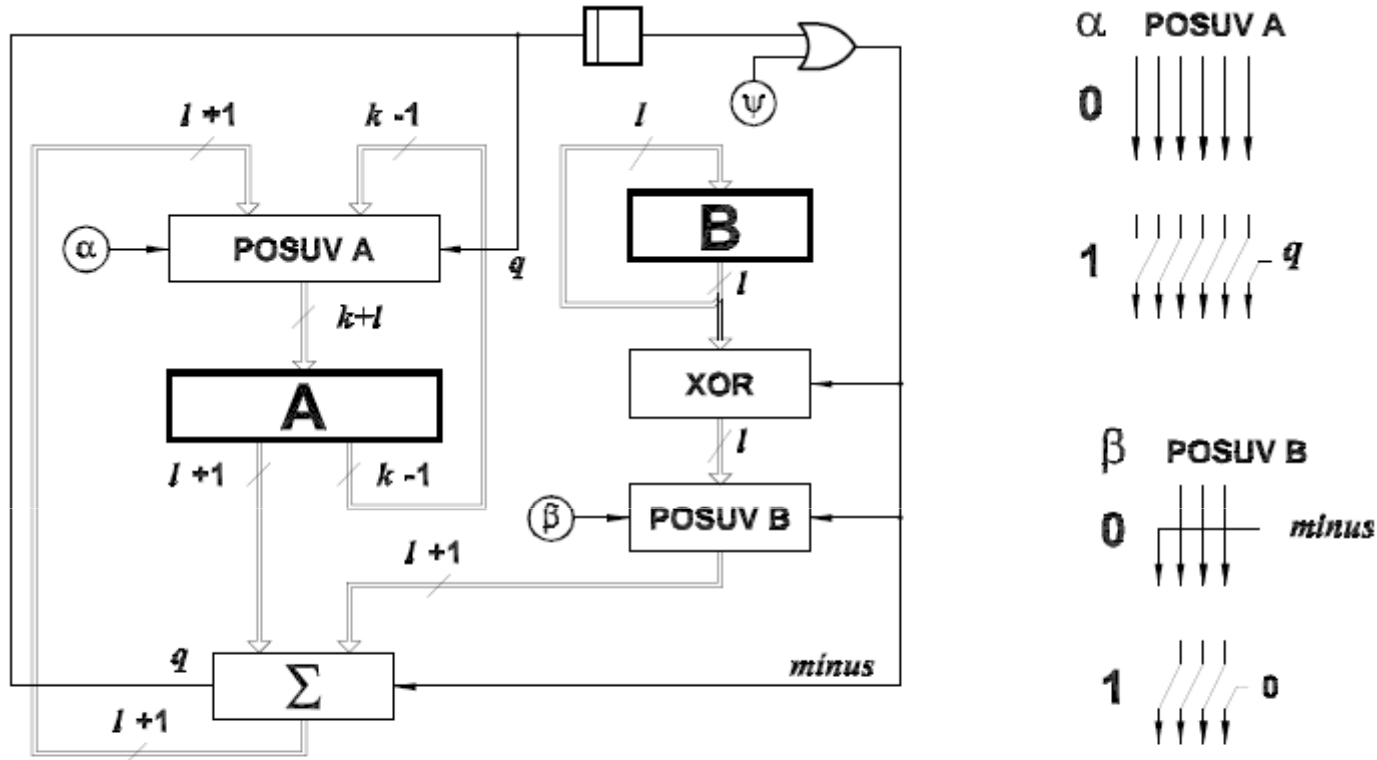
Příklad x/y

Dělenec $x=1010$ a dělitel $y=0011$

| i | operace | AC | MQ | B | komentář |
|----------|---|-----------|-----------|----------|--------------------------|
| | | 0000 | 1010 | 0011 | prvotní nastavení |
| 1 | SL | 0001 | 0100 | | |
| | nic | 0001 | 0100 | | podmínka if není splněna |
| 2 | SL | 0010 | 1000 | | |
| | | 0010 | 1000 | | podmínka if není splněna |
| 3 | SL | 0101 | 0000 | | $r \geq y$ |
| | AC = AC - B; MQ₀ = 1; | 0010 | 0001 | | |
| 4 | SL | 0100 | 0010 | | $r \geq y$ |
| | AC = AC - B; MQ₀ = 1; | 0001 | 0011 | | konec cyklu |

$x : y = 1010 : 0011 = 0011$ zbytek 0001 , ($10 : 3 = 3$ zbytek 1)

Dělička celých čísel - blokově

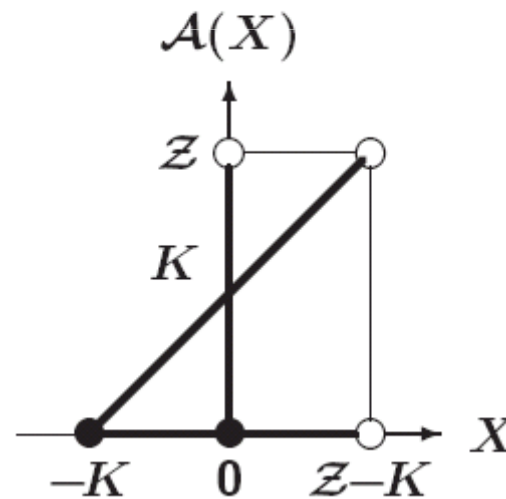


| | α | β | ψ |
|-------------|----------|---------|--------|
| 1. takt | 1 | 0 | 1 |
| další takty | 1 | 0 | 0 |
| návrat | 0 | 1 | 0 |

l (popř. $l+1$) taktů
 podíl ... nižší řády A
 zbytek ... vyšší řády A

Čísla INTEGER se znaménkem III.

- Je i jiná možnost pro zobrazení čísel se znaménkem?
- Ano, dokonce často používaná (viz dále):
- **kód aditivní** (jinak zvaný s posunutou nulou).



$$K = \frac{1}{2}Z - 1$$

Připomenutí: Z je modul

Sčítání a odčítání v aditivním kódu

- Platí:

$$\begin{aligned}\mathcal{A}(A + B) &= \mathcal{A}(A) + \mathcal{A}(B) - K \\ \mathcal{A}(A - B) &= \mathcal{A}(A) - \mathcal{A}(B) + K\end{aligned}$$

- Detekce přeplnění

- sčítání: stejná znaménka sčítanců a jiné znaménko výsledku,
- odčítání: znaménka menšence a menšitele se liší a liší se znaménka menšence a výsledku.

Iterační dělička - Goldschmidt

$$Q = \frac{N}{D} \quad \text{Předpoklad: } 0,5 \leq N < D < 1 \text{ (tzn. normalizovaný tvar)}$$

$$Q = \frac{N * F_0 * F_1 * F_2 * F_3 \dots}{D * F_0 * F_1 * F_2 * F_3 \dots}$$

Pokud budeme volit F_i tak, aby jmenovatel konvergoval k 1, bude čítec konvergovat k Q .

Z podmínky: $0,5 \leq D < 1$ můžeme D přepsat do tvaru: $D=1-x$

Zvolme $F_0=1+x$. Potom $D * F_0 = (1-x) * (1+x) = 1-x^2$

Zvolme $F_1=1+x^2$. Potom $D * F_0 * F_1 = (1-x^2) * (1+x^2) = 1-x^4$

$$Q = N(1+x)(1+x^2)(1+x^4)(1+x^8) \dots (1+x^{2^i}), \text{ kde } x = 1-D$$

Jak se v počítači zobrazují čísla typu REAL?

- Vědecká, neboli semilogaritmická notace.
 - Dvojice: EXPONENT (E), ZLOMKOVÁ část (nazývaná též mantisa M).
- Notace je normalizovaná.
 - Zlomková část vždy začíná binární číslicí 1,
 - Obecně: nenulovou číslicí.

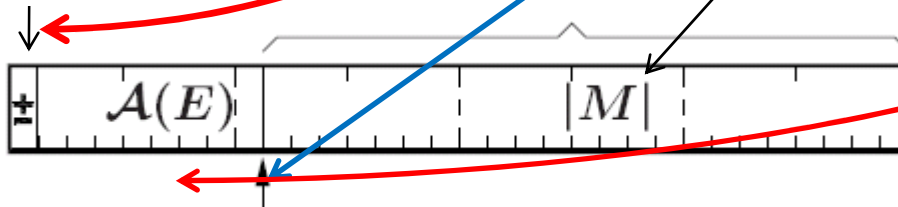
HW/SW interfejs

- Normalizováno jako IEEE754 ve verzích
 - Jednoduchá,
 - Dvojnásobná přesnost.
- V programovacím jazyce C se deklaruje jako `float` a `double`.

Příklady

- -2.34×10^{56} ← normalized
 - $+0.002 \times 10^{-4}$ ← not normalized
 - $+987.02 \times 10^9$ ← not normalized
- binárně
- $\pm 1.xxxxxxx_2 \times 2^{yyyy}$

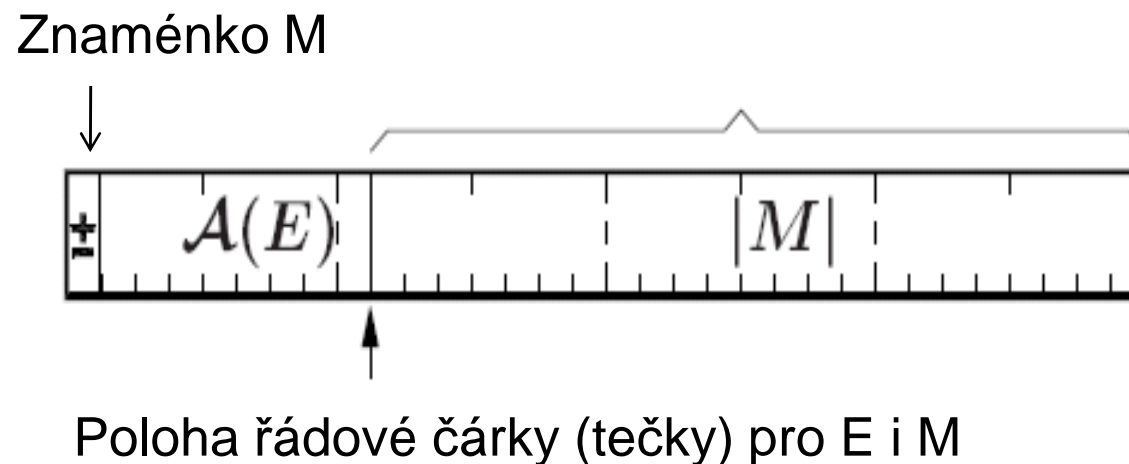
Znaménko M



Poloha řádové čárky (tečky) pro E i M

Formát čísla v pohyblivé řádové čárce

- Kód mantisy: přímý kód — znaménko a absolutní hodnota
- Kód exponentu: aditivní kód (s posunutou nulou).



Příklady reprezentace některých důležitých hodnot

Nula

| | | |
|--------------|-------------------------------------|------|
| Kladná nula | 0 00000000 000000000000000000000000 | +0.0 |
| Záporná nula | 1 00000000 000000000000000000000000 | -0.0 |

Nekonečno

| | |
|-------------------|-------------------------------------|
| Kladné nekonečno | 0 11111111 000000000000000000000000 |
| Záporné nekonečno | 1 11111111 000000000000000000000000 |

Některé krajní hodnoty

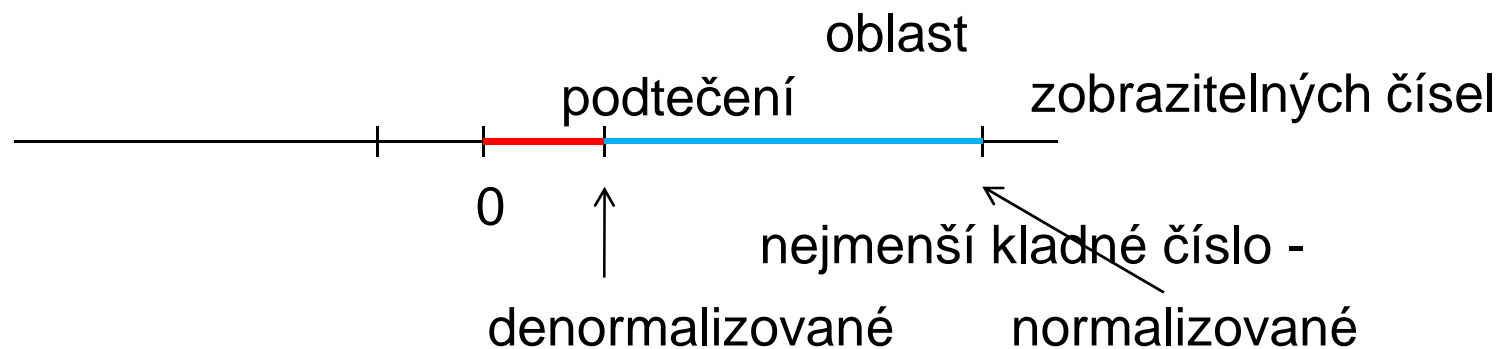
| | | |
|--------------------------------|-------------------------------------|--------------------------------|
| Nejmenší normalizované číslo | * 00000001 000000000000000000000000 | $\pm 2^{(1-127)}$ |
| Největší denormalizované číslo | * 00000000 111111111111111111111111 | $\pm (1 - 2^{-23}) * 2^{-126}$ |
| Nejmenší denormalizované číslo | * 00000000 000000000000000000000001 | $\pm 2^{-23} * 2^{-126}$ |

Denormalizované číslo?

- Smyslem zavedení denormalizovaných čísel je rozšíření reprezentovatelnosti čísel, která se nacházejí blíže k nule, tedy čísel velmi malých (v následujícím obrázku oblast označena modře).
- Denormalizovaná čísla mají nulový exponent a i skrytý bit před řádovou čárkou je implicitně nulový.
- Cenou je nutnost speciálního ošetření případu nulový exponent, nenulová mantisa.

Podtečení

- Jde o situaci, kdy zobrazované číslo není rovno nule, ale nedosahuje hodnoty nejmenšího zobrazitelného normalizovaného čísla.
- Mnoha podtečením lze předejít právě implementací čísel denormalizovaných.



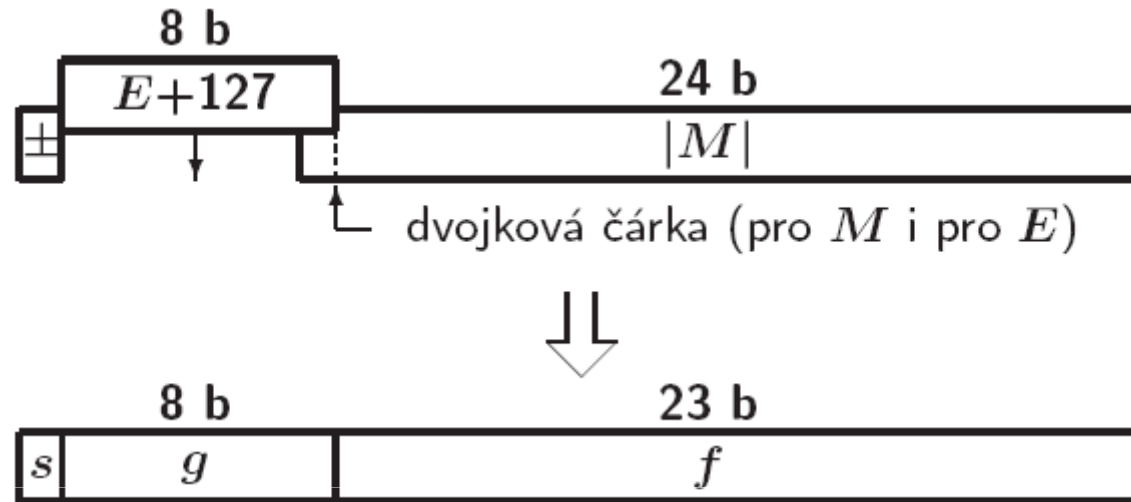
NaN

- Samé jedničky v exponentu,
- Mantisa je nenulová.
- V tom případě není bitový řetězec obrazem žádného čísla, anglicky Not-a-Number, Nan.

Shrnutí – zobrazitelná čísla a výjimky

| s-bit | Obraz exponentu | m | význam |
|-------|-----------------|----------|--|
| 0 | $0 < e < 255$ | > 0 | normalizované kladné číslo |
| 1 | $0 < e < 255$ | > 0 | normalizované záporné číslo |
| 0 | 0 | > 0 | denormalizované kladné číslo |
| 1 | 0 | > 0 | denormalizované záporné číslo |
| 0 | 0 | 0 | kladná nula |
| 1 | 0 | 0 | záporná nula |
| 0 | 255 | 0 | kladné nekonečno (overflow) |
| 1 | 255 | 0 | záporné nekonečno (overflow) |
| 0 | 255 | $\neq 0$ | NaN – not a number – nečíselná hodnota |
| 1 | 255 | $\neq 0$ | NaN – not a number – nečíselná hodnota |

ANSI/IEEE Std 754-1985 formáty – 32b a 64b



ANSI/IEEE Std 754-1985 — dvojitý formát — 64b

$g \dots 11b$

$f \dots 52b$

Skrytý bit

- Nejvyšší platný bit mantisy (který se do bitové reprezentace operandu neukládá) je
- závislý na hodnotě obrazu exponentu. Jestliže je obraz exponentu nenulový, je tento bit
- 1, mluvíme o **normalizovaných** číslech. Na druhou stranu jestliže je obraz
- exponentu nulový, je skrytý bit 0.
- Pak mluvíme o **denormalizovaném** čísle.

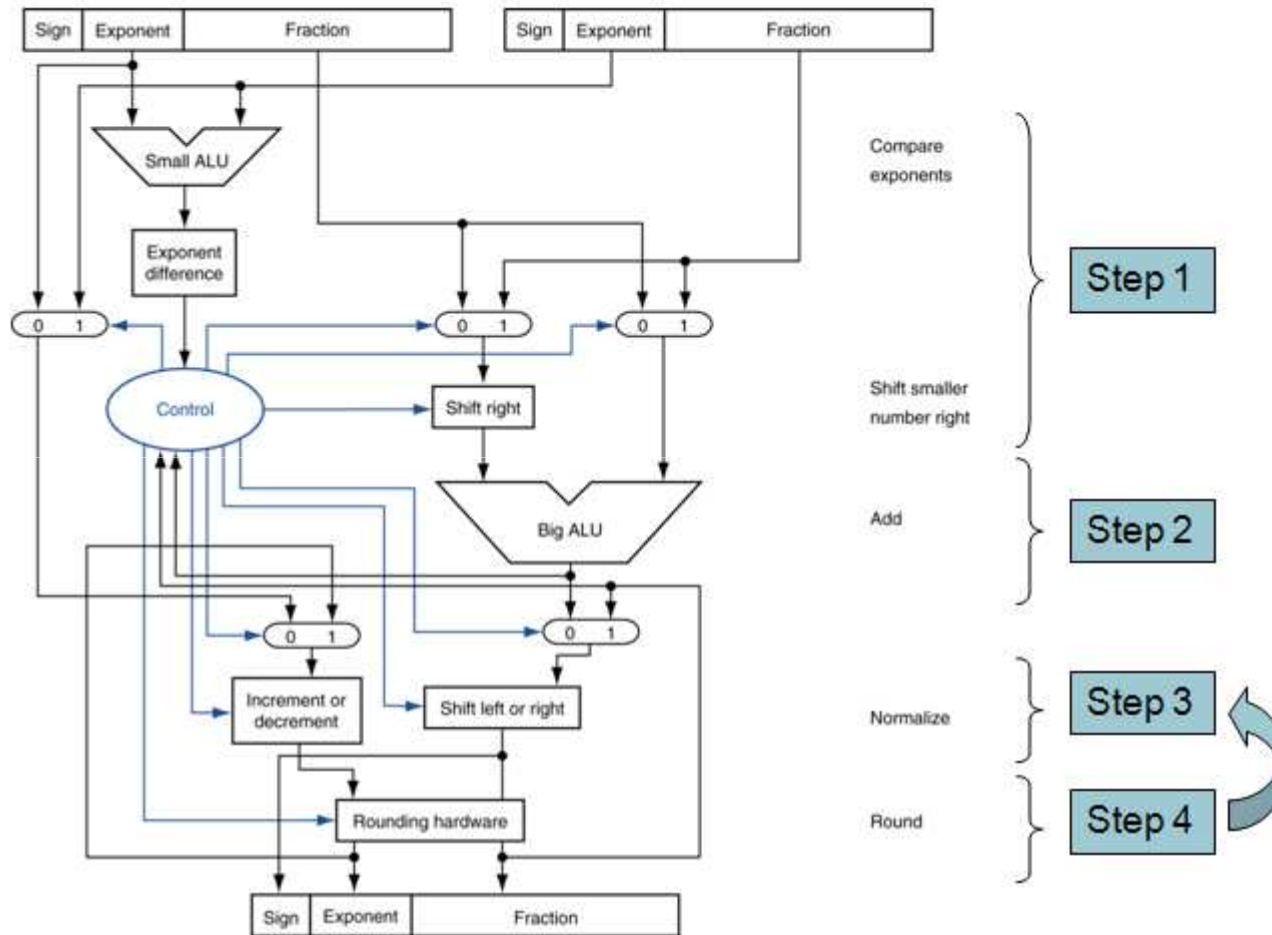
Porovnání dvou čísel ve FP

- Porovnání: je-li $A \geq B \iff A - B \geq 0$.
- Obrazy čísel A a B se odečtou jako čísla v přímém kódu a v pevné řádové čárce.
- To je výhodou zvoleného zobrazení čísel.

Algoritmus sčítání v pohyblivé řádové čárce

- Odečteme exponenty.
- Mantisu čísla s menším exponentem posuneme doprava o počet bitů, který je roven rozdílu exponentů.
- Sečteme mantisy obou čísel.
- Určíme počet nul mezi řádovou čárkou a první platnou číslicí součtu mantis.
- Posuneme součet doleva o tolik míst, kolik nul bylo nalezeno za řádovou čárkou.
- Zmenšíme původní exponent o počet nalezených nul.
- Zaokrouhlíme.

HW FP sčítačky



Násobení čísel v pohyblivé řádové čárce

- Exponenty sečteme.
 - Mantisy vynásobíme.
 - Normalizujeme.
 - Zaokrouhlíme.
-
- HW FP násobičky je srovnatelně složitý, jako FP sčítačky. Jen má namísto sčítačky násobičku.

Najdete všechny chyby v programu?

Chceme napsat program
pro zjištění součtu:

$$\sum_{i=1}^N \frac{1}{i^2}$$

```
#include <stdio.h>
int main()
{
    int i, sum=0;
    for(i=1; i<= 10^10; i++)
        sum += 1/i*i;
    printf("Soucet je: %d",sum);
    return 0;
}
```

Najdete všechny chyby v programu?

- Který způsob je nejvýhodnější?

$$\sum_{i=1}^N \frac{1}{i^2} = \sum_{i=N}^1 \frac{1}{i^2} = \sum_{i=1}^N \frac{1}{(N-i+1)^2}$$

$$\sum_{i=1}^{10^{10}} \frac{1}{i^2} \approx 1.6449340578301865,$$

$$\sum_{i=10^{10}}^1 \frac{1}{i^2} \approx 1.6449340667482264$$

} typ double pro oba případy.. Proč se liší?

Překvapení na závěr ???

```
#include <stdio.h>
int main()
{
    float x;
    x = 116777215.0;    printf("%.31f\n", x);
    x = 116777216.0;    printf("%.31f\n", x);
    x = 116777217.0;    printf("%.31f\n", x);
    x = 116777218.0;    printf("%.31f\n", x);
    x = 116777219.0;    printf("%.31f\n", x);
    x = 116777220.0;    printf("%.31f\n", x);
    x = 116777221.0;    printf("%.31f\n", x);
    return 0;
}
```