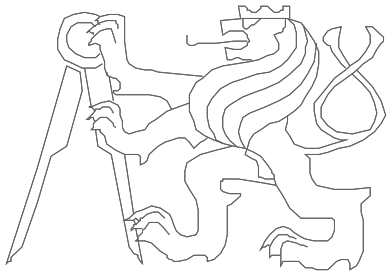
Architektura počítačů

Počítačová aritmetika a úvod

Pavel Píša, Michal Štepanovský, Miroslav Šnorek



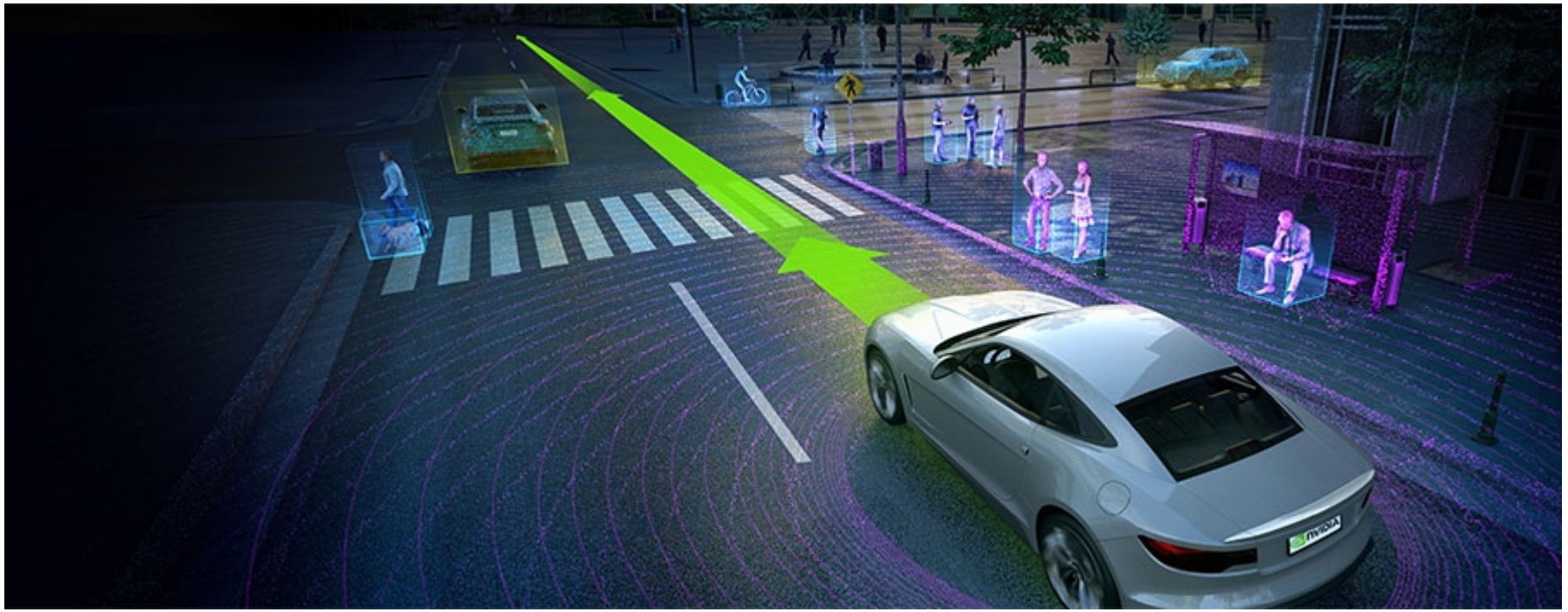
České vysoké učení technické, Fakulta elektrotechnická

Základní cíl předmětu

- Cíl je porozumět struktuře počítače, abyste mohli lépe využít jeho možností k dosažení jeho vyššího výkonu.
- Dále je probíraná návaznostech/propojení HW/SW (periferie)
- 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/b35apo/start
- Cílem tohoto předmětu není naučit vás počítač navrhnout (viz B4M35PAP – Pokročilé architektury počítačů a B4M38AVS – Aplikace vestavných systémů)

Motivační příklad: (neformální uvedení do probíraných témat)

Autonomní řízení automobilů



Zdroj: http://www.nvidia.com/object/autonomous-cars.html

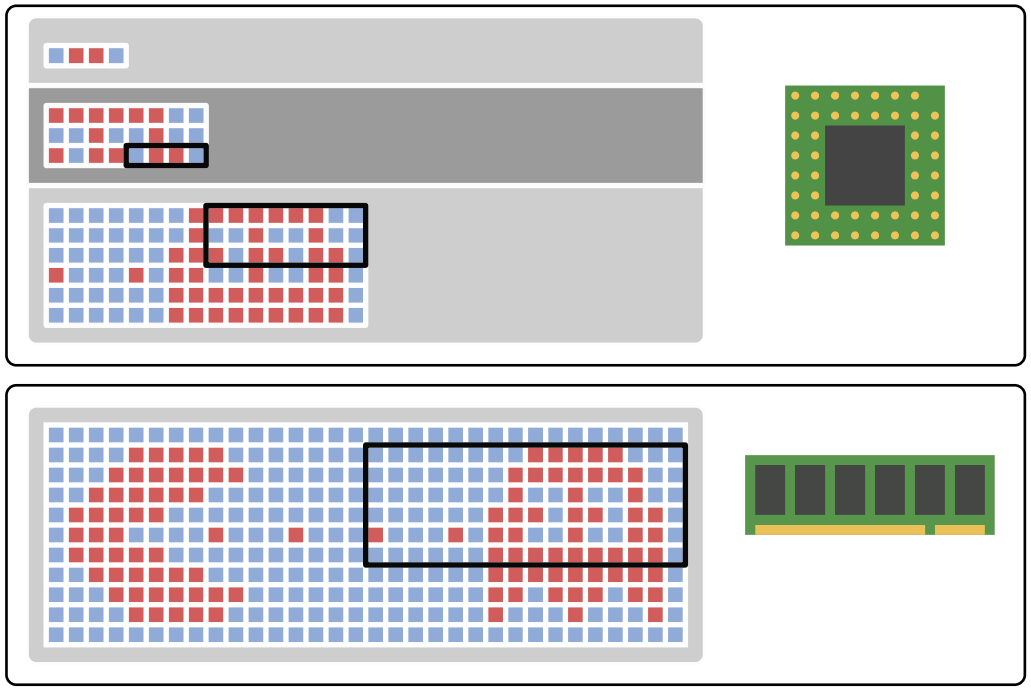
- Mnoho úloh z oblasti umělé inteligence založeno na hlubokých neuronových sítích (deep neural networks)
- Průchod neuronové sítě – maticové násobení

Průchod neuronové sítě – maticové násobení

- Výsledky jednoho z mnoha experiemntů
  - Naivní algoritmus (3 × for) – 3.6 s = 0.28 FPS
  - Optimalizace přístupů k paměti – 195 ms = 5.13 FPS
    (bezpodmínečně nutná znalost HW)
  - Čtyři jádra – 114 ms = 8.77 FPS
    (nutnost výběru minimální nutné synchronizace)
  - GPU (256 procesorů) — 25 ms = 40 FPS
    (znalost předávání dat mezi hlavním CPU a koprocesory)
- Naivním algoritmus, mat. knihovnou Eigen (1 jádro a 4 jádra (2 fyzické) na i7-2520M, kompilace s -O3), GPU na základě měření Joela Matějky ze zkupiny http://industrialinformatics.cz/ kde se v rámci evropských projektů vývojem operačních systémů a budoucích SW platforem pro autonomní řízení zabýváme
- Jak docílit zrychlení?

Optimalizace přístupů k paměti

- Úprava algoritmu s ohledem na paměťovou hierarchii
- Data z (vyrovnávací) paměti blízko procesoru lze získat rychleji (rychlé paměti mají ale malou velikost)



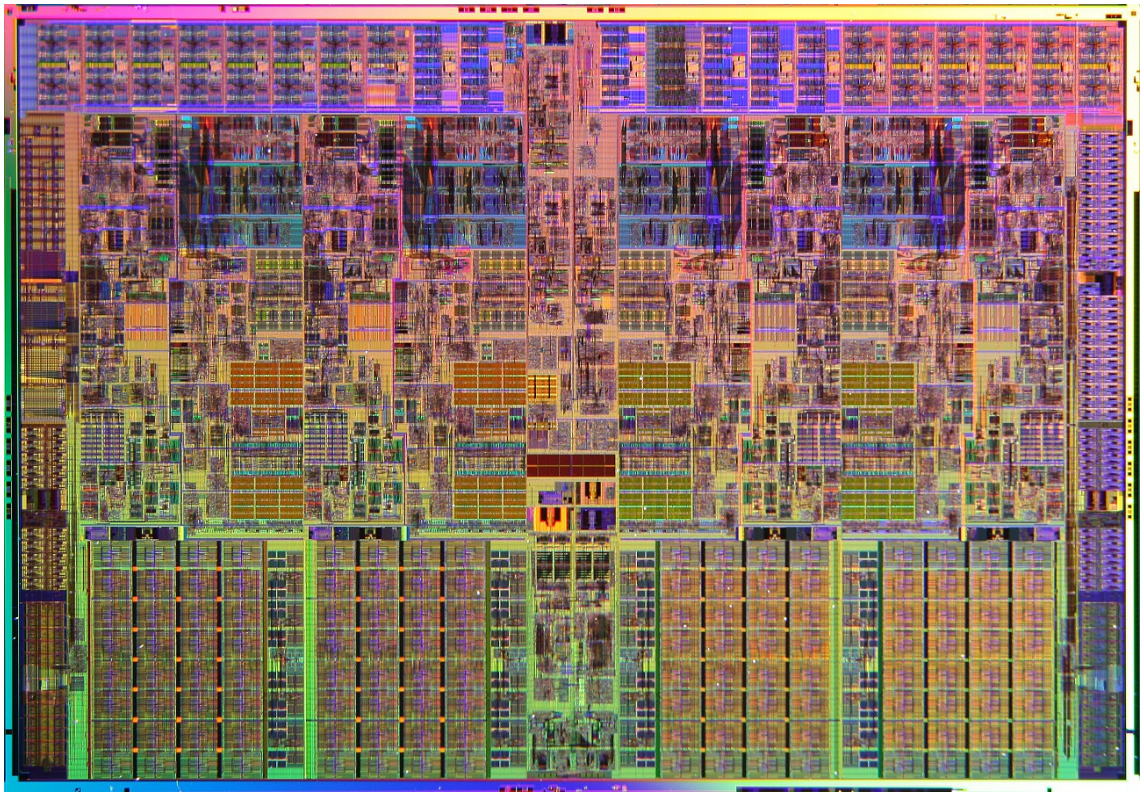
Predikce skoků / přístupů k paměti

- Kvůli zvýšení průměrného výkonu je vykonávání instrukcí rozděleno na několik fází => nutnost načítat několik instrukcí / dat dopředu
- Každý podmínka (if, loop) znamená možný skok – špatná predikce je drahá
- Je dobré mít představu jak predikce fungují a jaké alternativy na daném CPU/HW ke skokům existují. (Např. vektorové/multimediální inst.)

Zdroj: https://commons.wikimedia.org/wiki/File:Plektita_trakforko_14.jpeg

Paralelizace – vícejádrový procesor

- Požadavky na synchronizaci
- Vzájemné propojení a možnosti komunikace mezi procesory
- Přesuny mezi úrovněmi paměti jsou velmi drahé
- Nevhodné sdílení mezi jádry vede k pomalejšímu kódu než na jednom CPU



Intel Nehalem Processor, Original Core i7
Zdroj: http://download.intel.com/pressroom/kits/corei7/images/Nehalem_Die_Shot_3.jpg

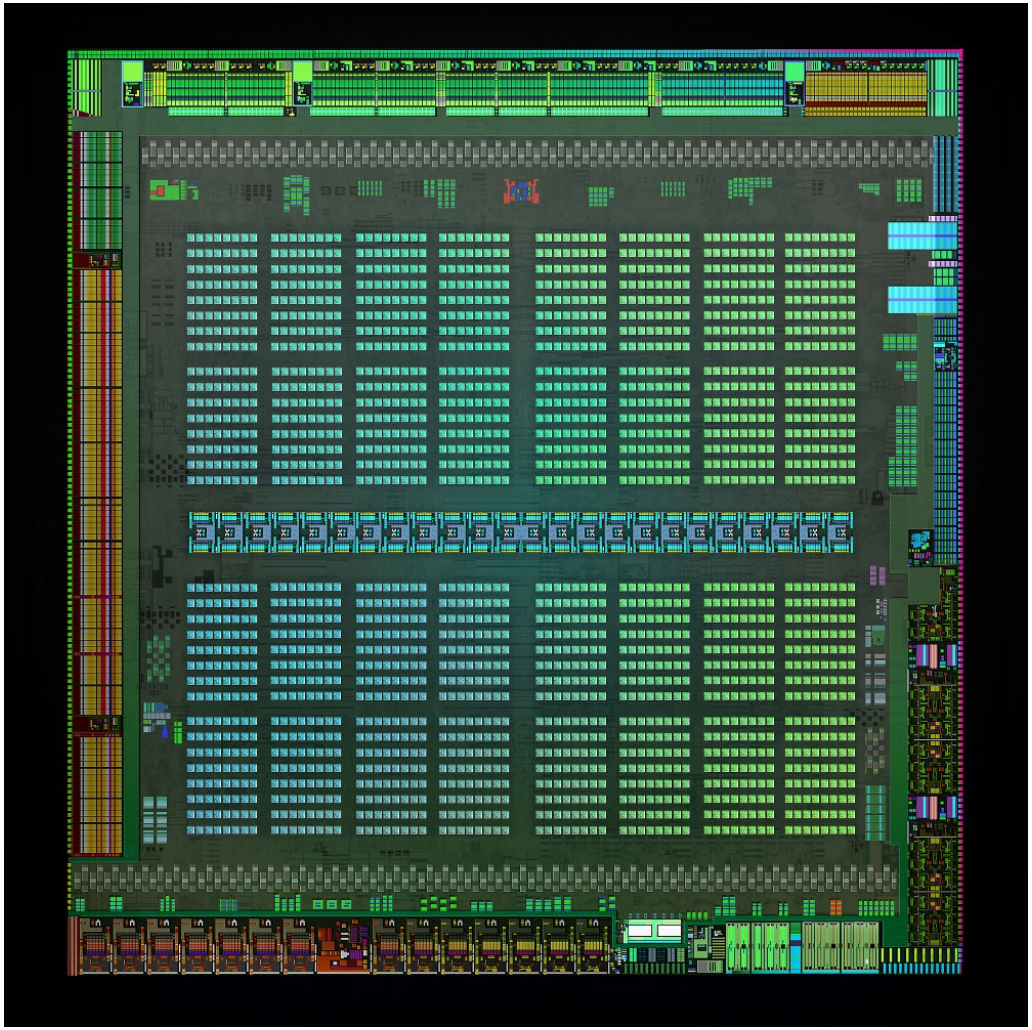
Výpočetní koprocesory – GPU

- Procesor s mnoha jednoduchými jádry (stovky)
- Některé jednotky sdílené
- Pro efektivní využití nutno znát základní hardwarové vlastnosti

Zdroj: https://devblogs.nvidia.com/parallelforall/inside-pascal/

GPU – Maxwell

- GM204
- 5200 miliónů tranzistorů
- 398 mm^2
- PCIe 3.0 x16
- 2048 výpočetních jednotek
- 4096 MB
- 1126 MHz
- 7010 MT/s
- 72.1 GP/s
- 144 GT/s
- 224 GB/s

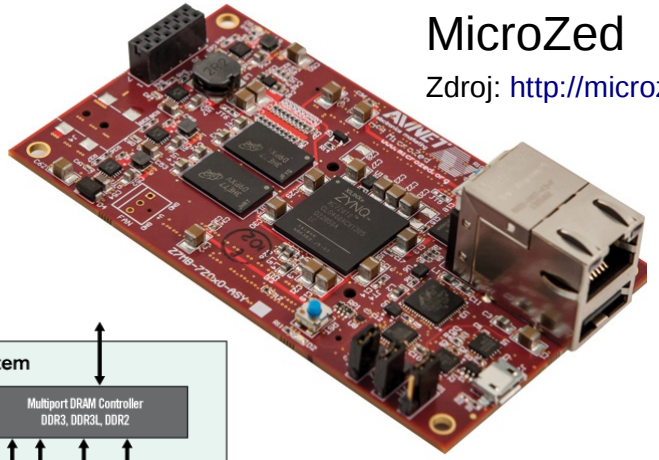


Zdroj: http://www.anandtech.com/show/8526/nvidia-geforce-gtx-980-review/3

FPGA – návrh/prototyp vlastního hardware

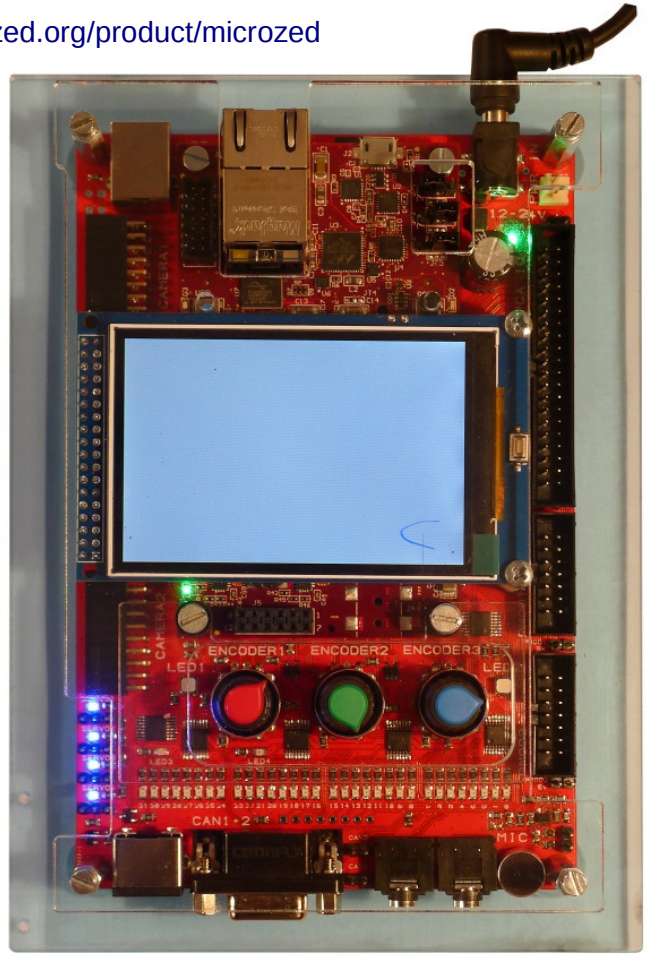
- Programovatelné hradlové pole
- Umožňuje efektivní naprogramování specifických funkcí (filtry – obrazové nebo zvukové, FFT analýzu, vlastní procesor…)
- Připravené bloky na čipu jsou pospojovány programátorem
- Zynq 7000 FPGA – několik ARM jader propojených s FPGA – rychlý a snadný přístup k FPGA/periferiím z programu
- (setkáte se na cvičeních, ale v rámci APO nebudete programovat FPGA, hardware bude již připravený)

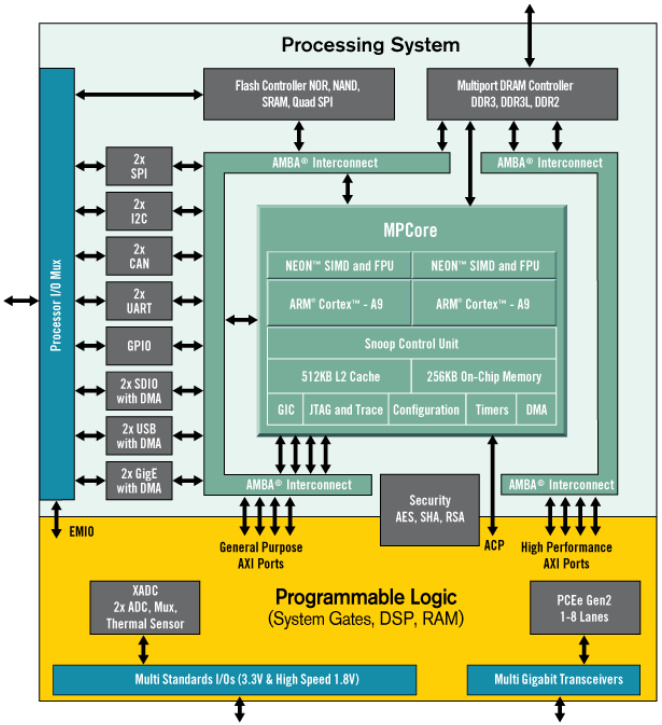
Xilinx Zynq 7000 a MicroZed APO



MicroZed

Zdroj: http://microzed.org/product/microzed





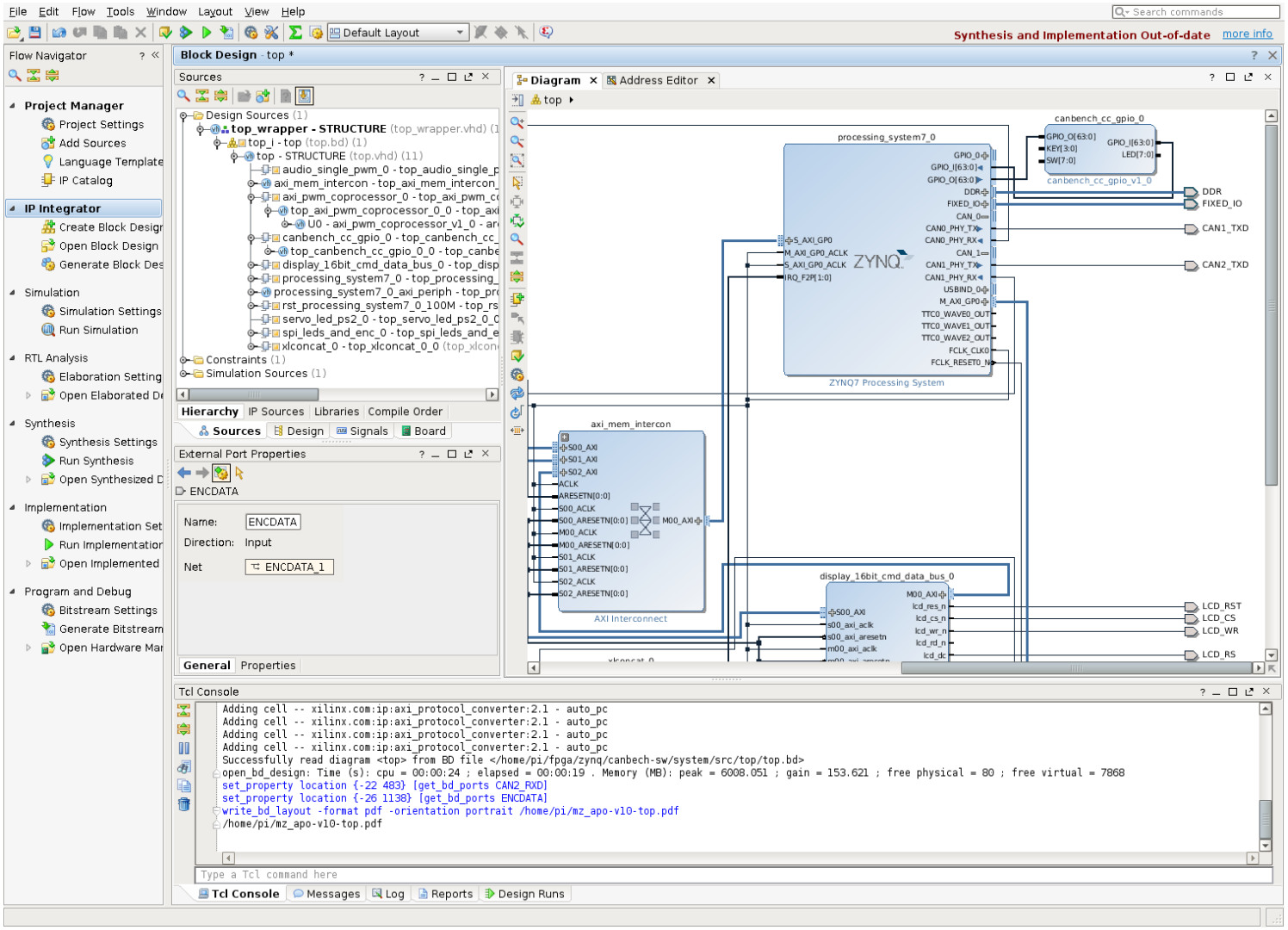
Zdroj: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

Zdroj: https://cw.fel.cvut.cz/wiki/courses/b35apo/start

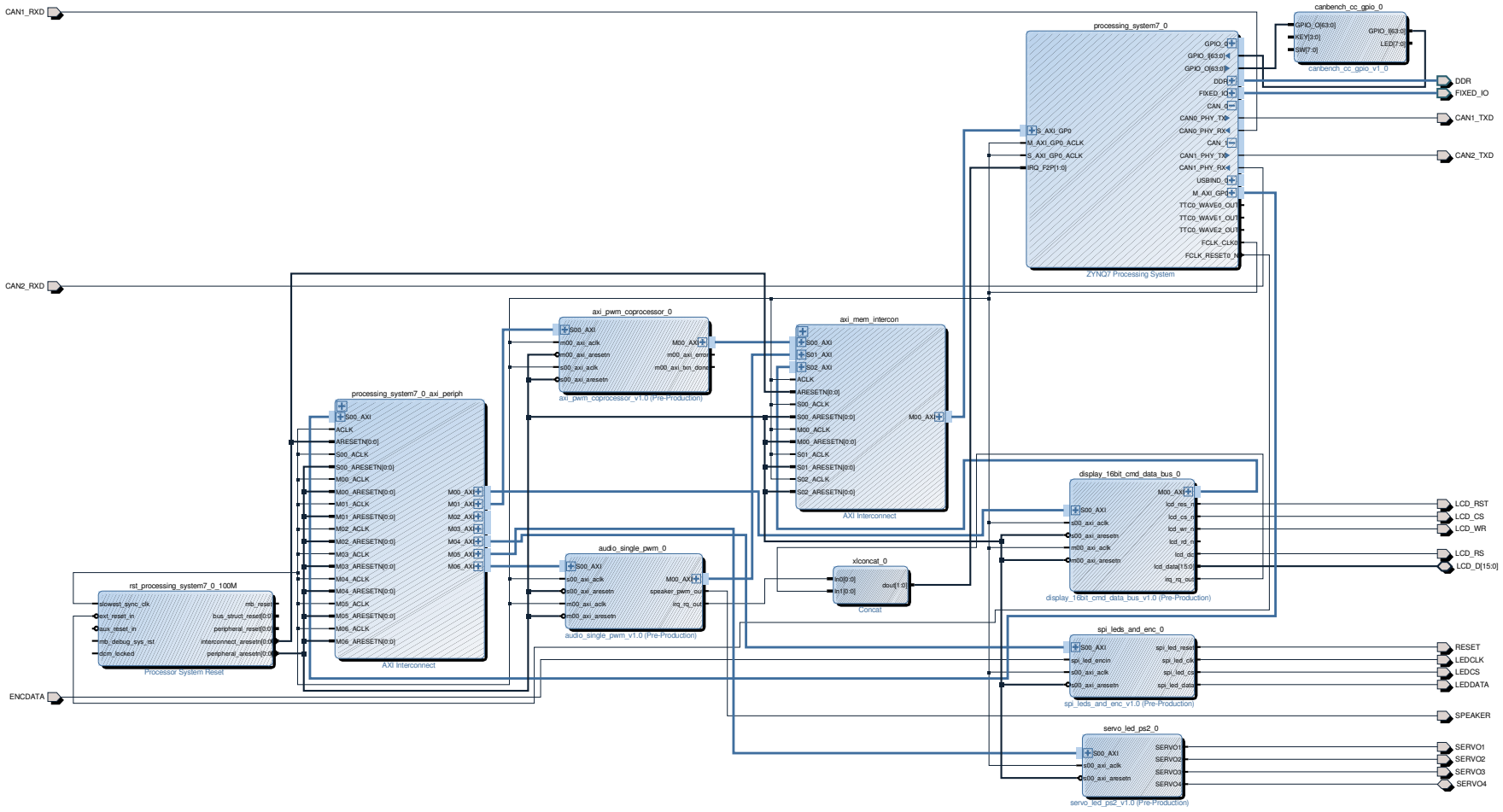
MZ_APO – parametry

- Základní čip: Zynq-7000 All Programmable SoC
- Typ: Z-7010, součástka XC7Z010
- CPU: Dual ARM® Cortex™-A9 MPCore™ @ 866 MHz (NEON™ & Single / Double Precision Floating Point)
  2x L1 32+32 kB, L2 512 KB
- FPGA: 28K Logic Cells (~430K ASIC logic gates, 35 kbit)
- Výpočetní jednotky v FPGA: 100 GMACs
- Paměti v FPGA: 240 KB
- Paměť na desce MicroZed: 1GB
- Operační systém: GNU/Linux
  - GNU LIBC (libc6) 2.19-18+deb8u7
  - Jádro Linux 4.9.9-rt6-00002-ge6c7d1c
  - Distribuce: Debian Jessie

MZ_APO – Logický návrh v SW Xilinx Vivado



MZ_ZPO – propojení sběrnic



Operační systém GNU/Linux – od trpaslíků ...

Linux – od trpaslíků po superpočítače

- TOP500 https://www.top500.org/ (https://en.wikipedia.org/wiki/TOP500)
- Současný první: Sunway TaihuLight, Sunway MPP, SW26010 Sunway, NRCPC (Čína)
- Příklad parametrů: IBM Roadrunner, Los Alamos National Laboratory
- Majitel: National Nuclear Security Administration, USA
- LINPACK/BLAS (Basic Linear Algebra Subprograms) systém
- Architectura: 12,960 IBM PowerXCell 8i CPUs,
- 6,480 AMD Opteron dual-core processors, Infiniband, Linux
- Systém: Red Hat Enterprise Linux a Fedora
- Napájení: 2.35 MW
- Velikost: 296 stojanů
- 560 m2
- Pamět: 103.6 TiB
- Výkon 1.042 petaflops
- Cena: USD $125M

- SGI SSI (single system image) Linux, 2048 Itanium CPU a 4TiB RAM

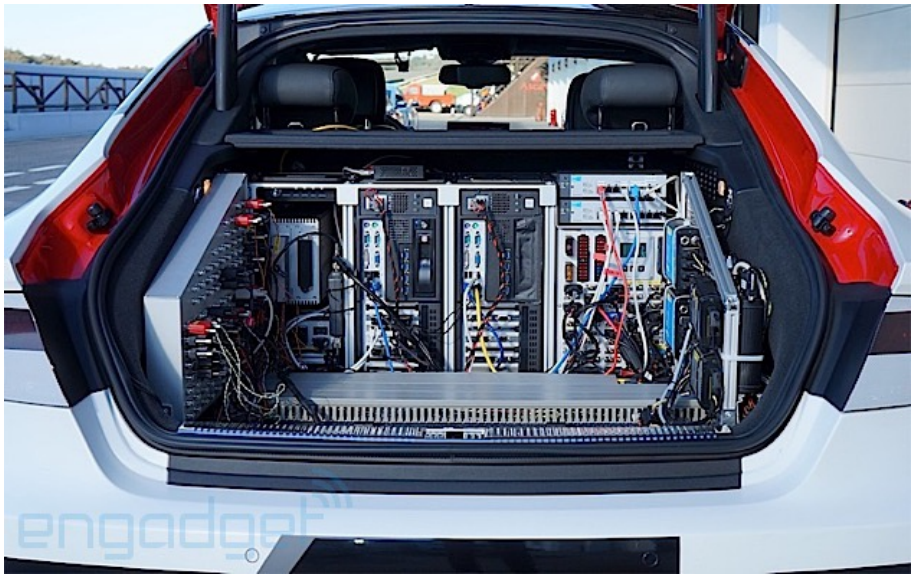
Linux kernel a open-source

- Projekt jádra Linux
  - od roku 2005 přispělo 13,500 vývojářů
  - přidáno 10,000 řádek kódu deně
  - 8,000 zrušeno a 1,500 až 1,800 změněno
  - Správa kódu verzovací systém GIT
- Úspěšných projektů s otevřeným zdojovým kódem je mnoho
- Zapojit se může každý
- Google Summer of Code pro studenty univerzit
  - https://developers.google.com/open-source/gsoc/

Zdroj: https://www.theregister.co.uk/2017/02/15/think_different_shut_up_and_work_harder_says_linus_torvalds/

Zpět k motivačnímu příkladu autonomního řízení

- Výsledek dobré znalosti hardware
  - Zrychlení (v našem případě 18× při využití stejného počtu jader)
  - Snížení potřebného výkonu
  - Úspora energie
  - Možnost zmenšení aktuálních řešení



Aplikovatelnost znalostí a postupů probíraných v předmětu

- Aplikace nejen v autonomním řízení
- V jakémkoli embedded zařízení – znížení velikosti, spotřeby, spolehlivosti
- V datových vědách – značné zkrácení doby běhu a úspora energie při výpočtech
- V uživatelském rozhraní – zlepšení odezvy aplikace
- Prakticky všude…

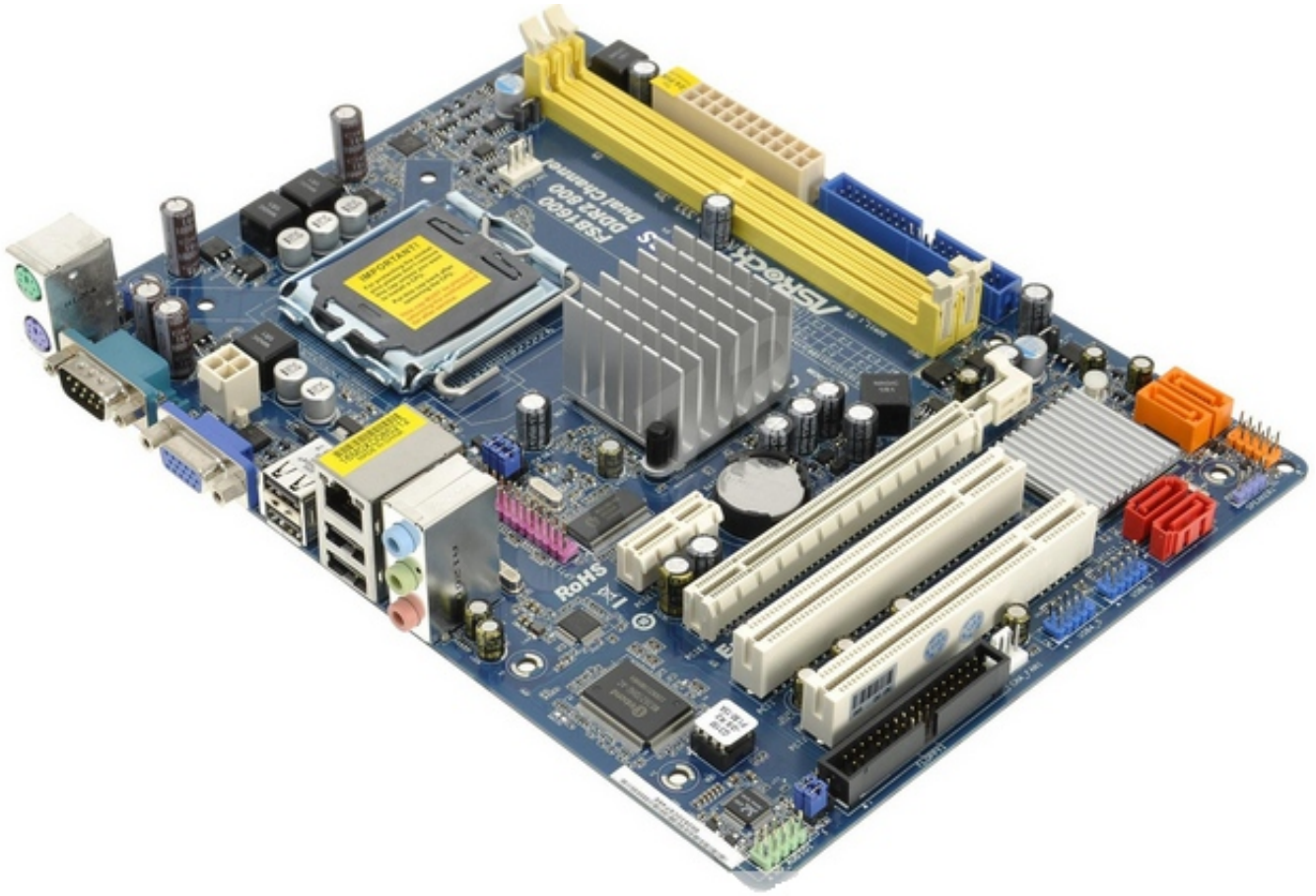
Proč je potřeba studovat i nízkoúrovňovou konstrukci počítače

- Pro návrh nové počítačové/procesorové architektury
- Pro implementaci vybrané architektury v integrovaném obvodu/FPGA
- Pro obvodový návrh hardware/systému (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 subsystému 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 pro ušetření energie. Přitom to je oblast kde probíhá skutečný vývoj a mají z dlouhodobějšího technologického a i vědeckého pohledu smysl.

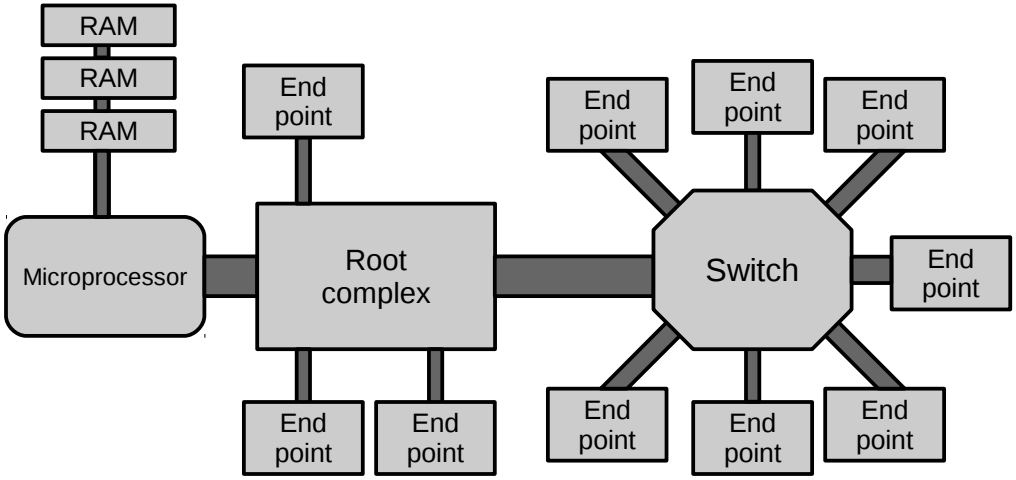
Další motivace a příklady

- 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 multimédii nemůže být vykonána dobře bez takovýchto znalostí
- 1/3 našeho kurzu je zaměřená i na přístup k periferiím
- Další příklady
  - Facebook – HipHop for PHP -> C++/GCC -> machine code
  - 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)

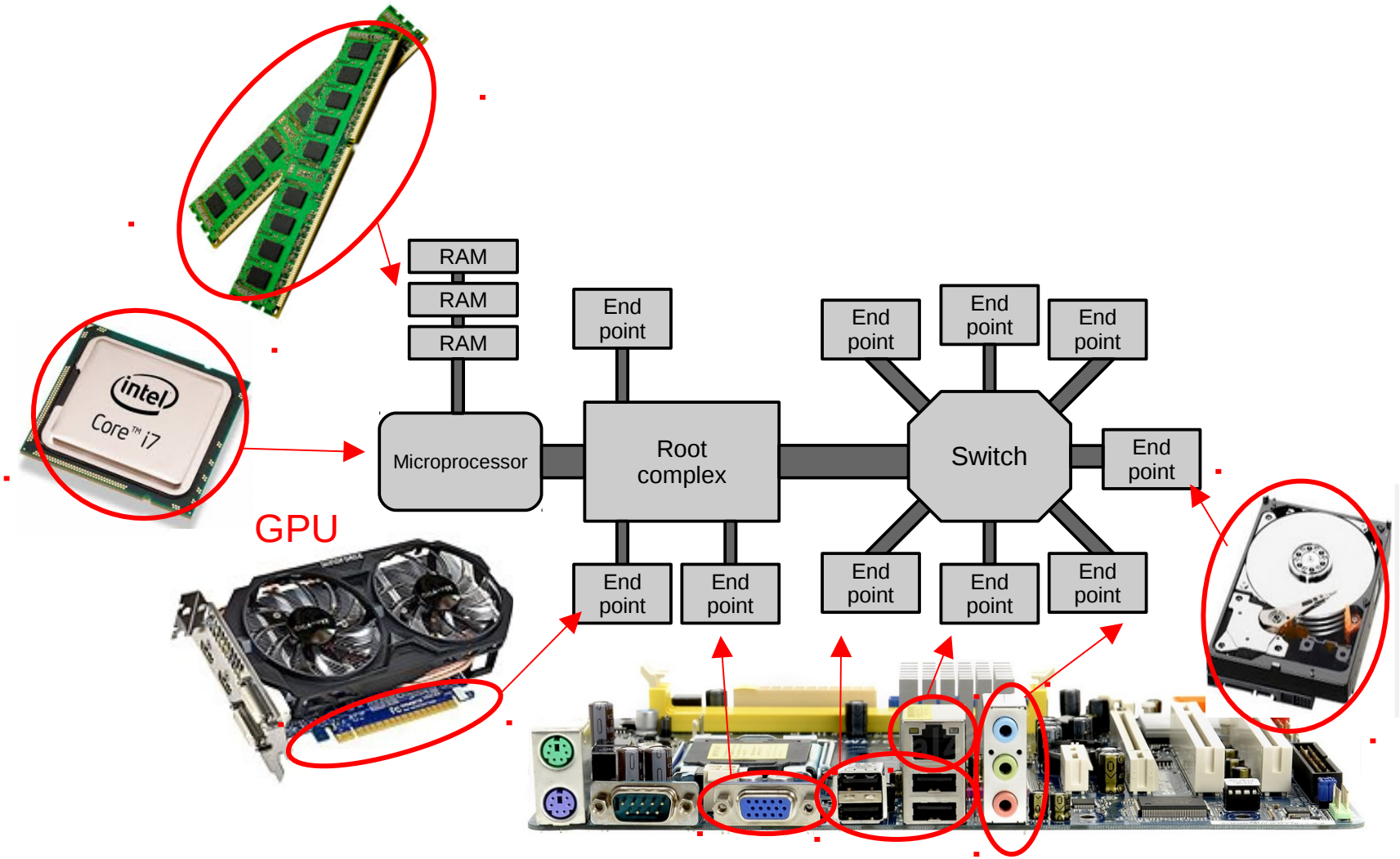
Architektura dnešního PC ???



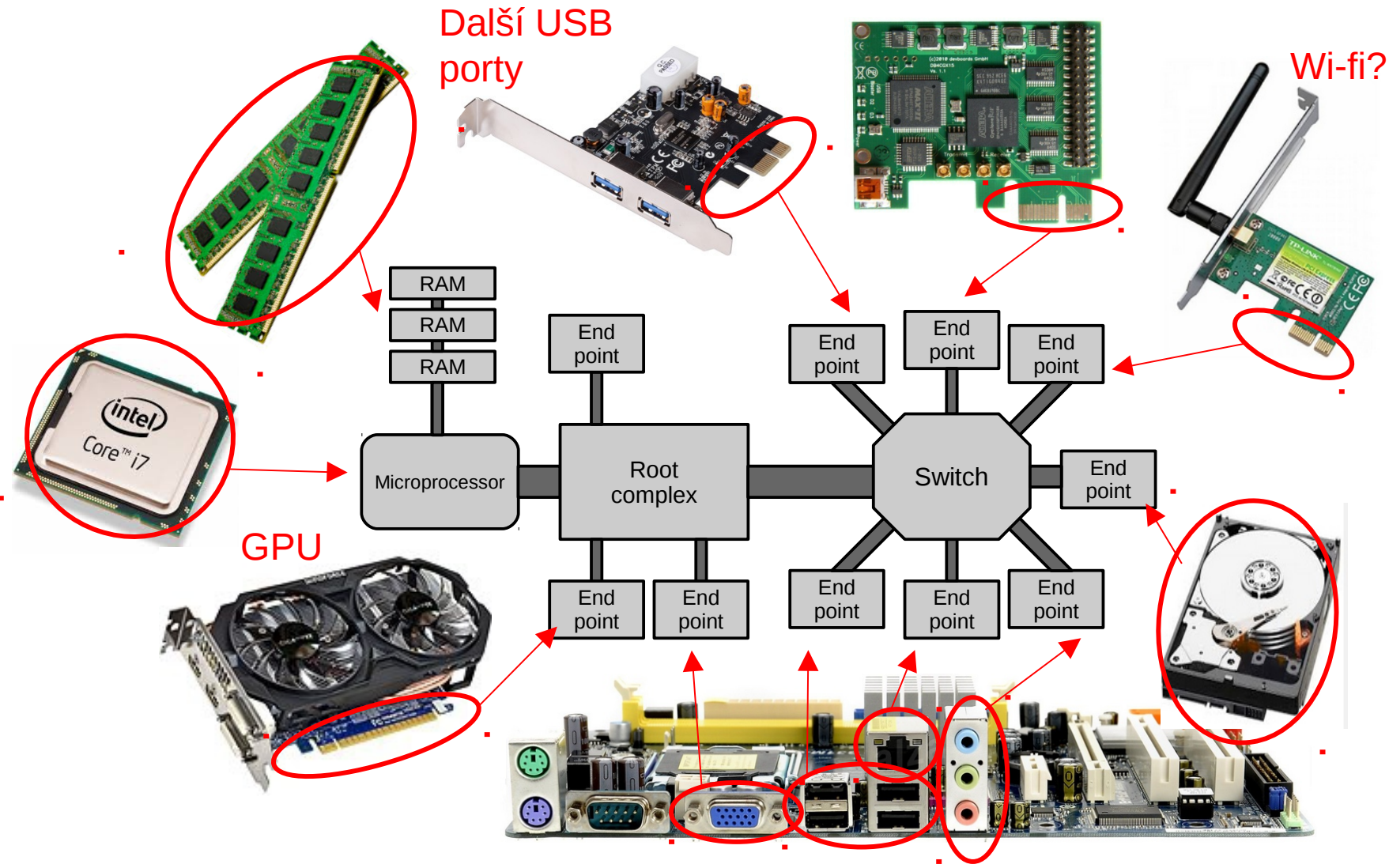
PC – Blokový diagram propojení sběrnic



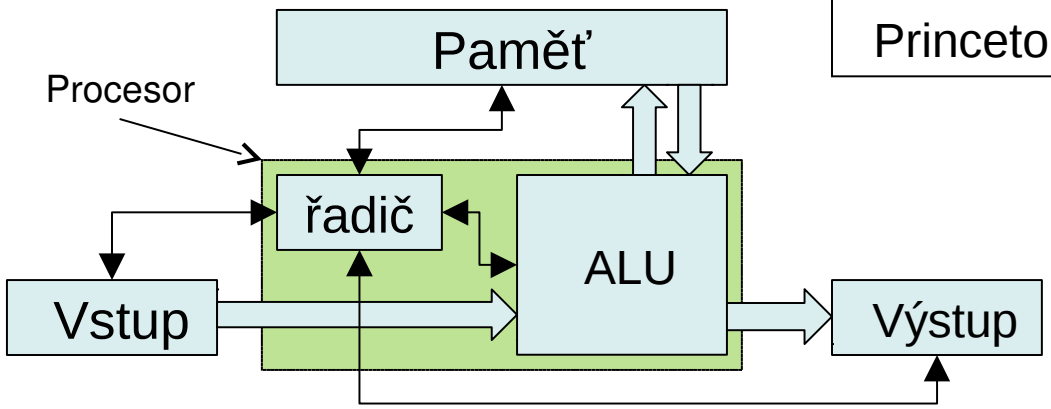
PC – Blokový diagram propojení sběrnic



PC – Blokový diagram propojení sběrnic



John von Neumann, maďarský fyzik



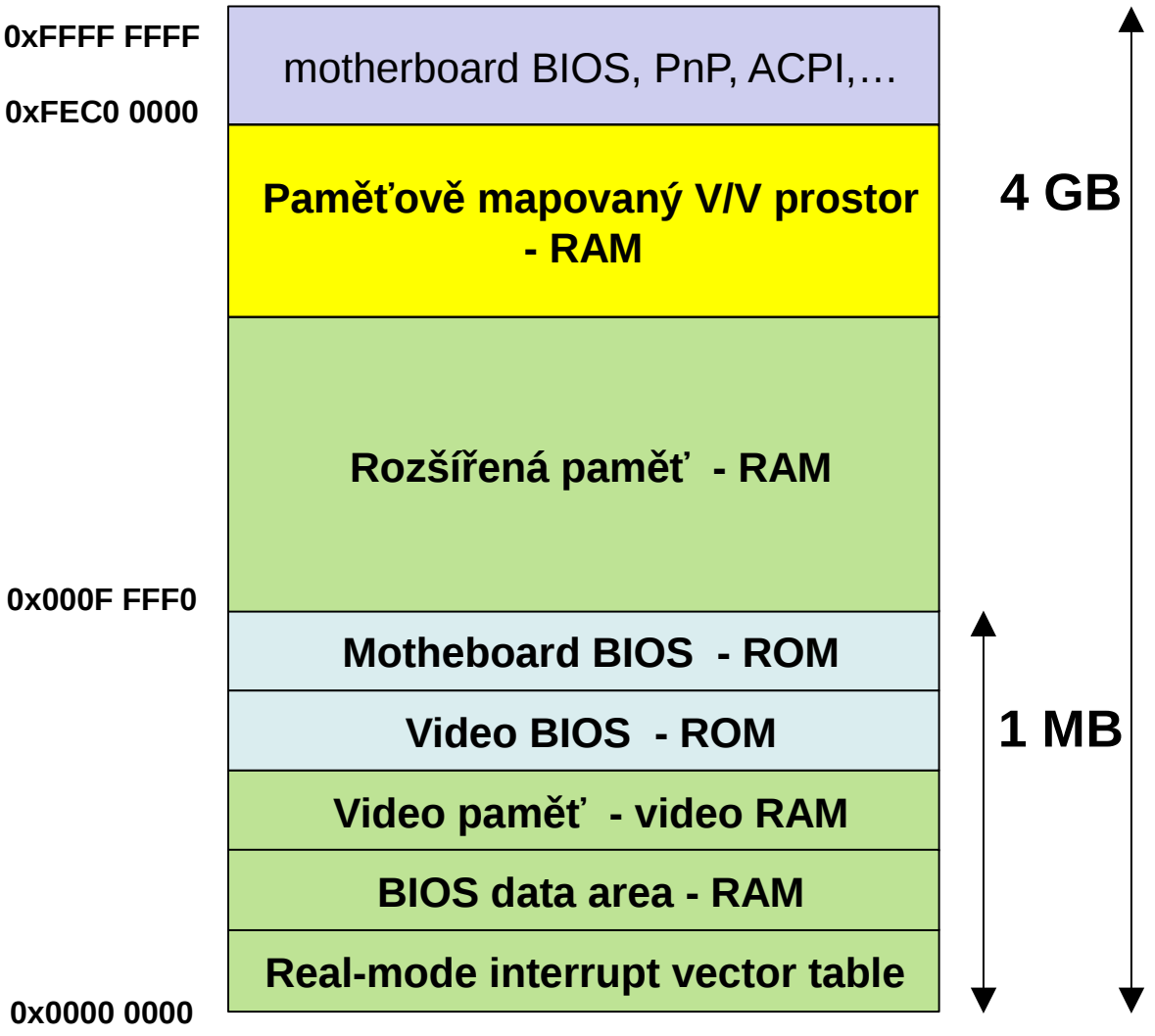
Architektura počítače podle JvN +
Princeton Institute for Advanced Studies

28. 12. 1903 -
8. 2. 1957

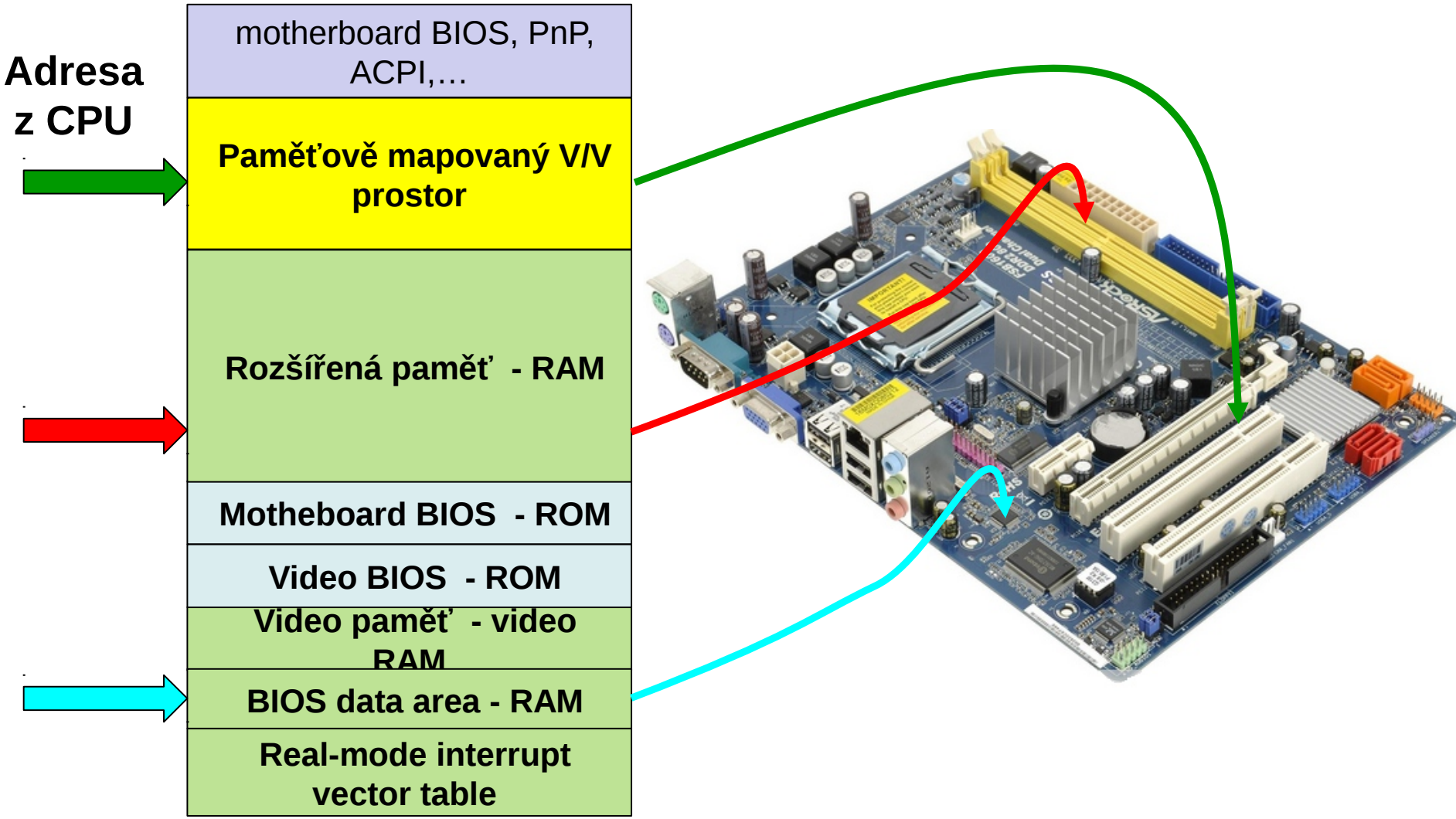
- 5 funkčních jednotek – řídicí jednotka (řadič), aritmeticko-logická jednotka, paměť, vstupní zařízení, výstupní zařízení
- Nezávislost struktury počítače na zpracovávaných problémech. Musí se zavést program a musí se uložit do paměti. Ten řídí činnost počítače.
- **Programy a výsledky (data) se ukládají do téže paměti.** Ta je rozdělena na stejně velké části (buňky), které jsou průběžně očíslované – adresa.
- Po sobě jdoucí instrukce se ukládají do po sobě jdoucích buněk.
- Existují instrukce aritmetické, logické, přenosu, skokové a ostatní.

Fyzický adresní prostor a jeho význam

- Fyzický adresní prostor je prostor, který přímo adresuje samotný procesor.
- Tento prostor může procesor adresovat na jeho adresní sběrnici.



Fyzický adresní prostor a jeho význam



Jak vypadá uvnitř smartphone? Samsung Galaxy S4

- **Android 5.0 (Lollipop)**
  - **Linux 3.5.4 (**2012/2014**)**
  - **Android Runtime (ART)**
- **2 GB RAM**
- **16 GB pro uživatele**
- **1920 x 1080 display**
- **8-jádrový CPU (**čip Exynos 5410**):**
  - **čtyři** 1.6 GHz ARM Cortex-A15
  - **čtyři** 1.2 GHz ARM Cortex-A7

Samsung Galaxy S4 – Mechanická konstrukce

Zdroj: http://www.techinsights.com/about-techinsights/overview/blog/samsung-galaxy-s4-teardown/

Samsung Galaxy S4 – Hlavní deska plošného spoje

Power management

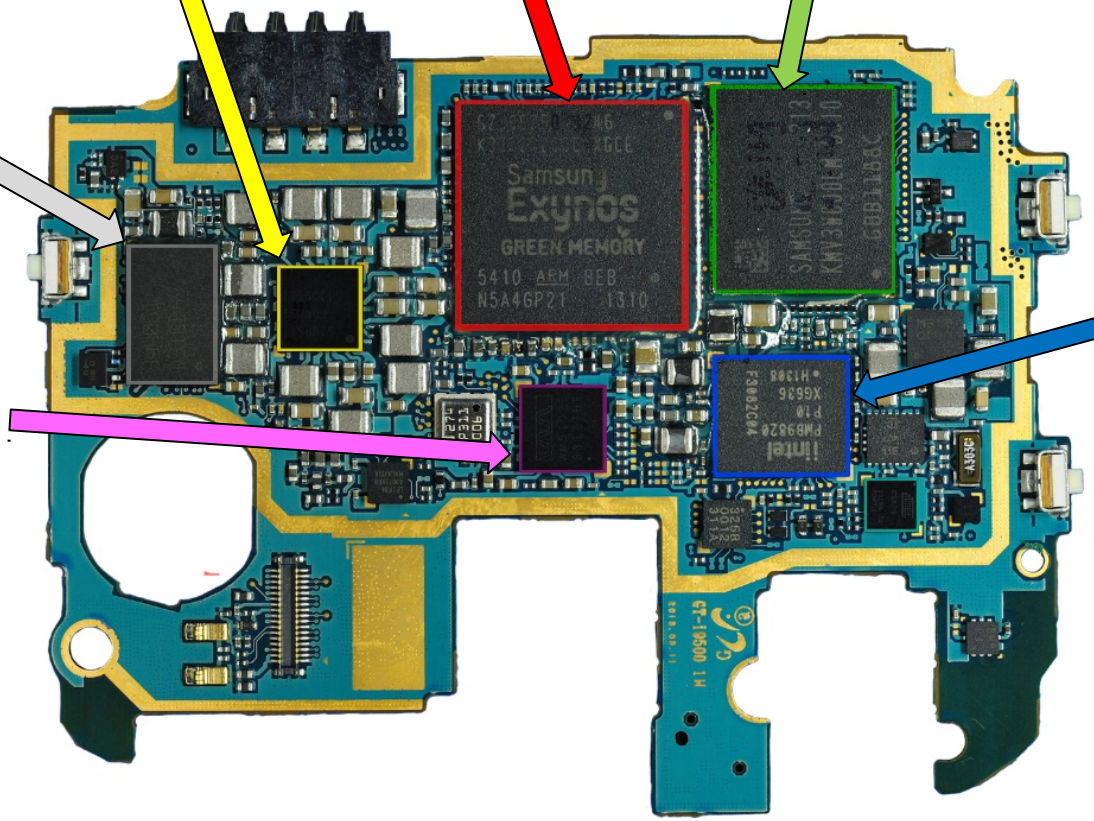
Exynos 5410 (**8-core CPU + 2GB DRAM**)

Multichip memory: 64 MB DDR SDRAM, **16GB NAND Flash**, Controler

Wi-fi (broadcom BCM4335)

DSP procesor pro zpracování hlasu, audio codec

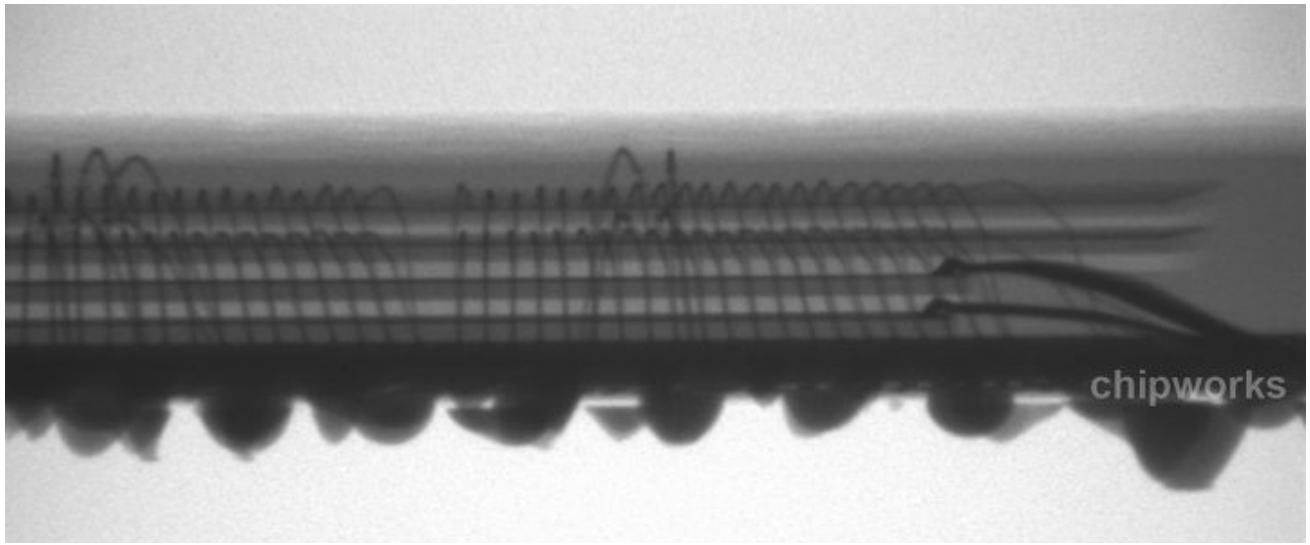
Intel PMB9820 baseband processor (funkce rádia přes anténu - EDGE, WCDMA, HSDPA/HSUPA)



Zdroj: http://www.techinsights.com/about-techinsights/overview/blog/samsung-galaxy-s4-teardown/

Samsung Galaxy S4 – Exynos 5410 – paměť

Rentgenový snímek paměťového modulu PoP (Package on Package), který je přiletovaný na vlastní čip CPU



Čtyři 4Gb(it) čipy propojené dohromady do kapacity 2GB(yte)
QDP – Quad die package – čtyři vrstvy nad sebou

Zdroj: http://gamma0burst.tistory.com/m/600

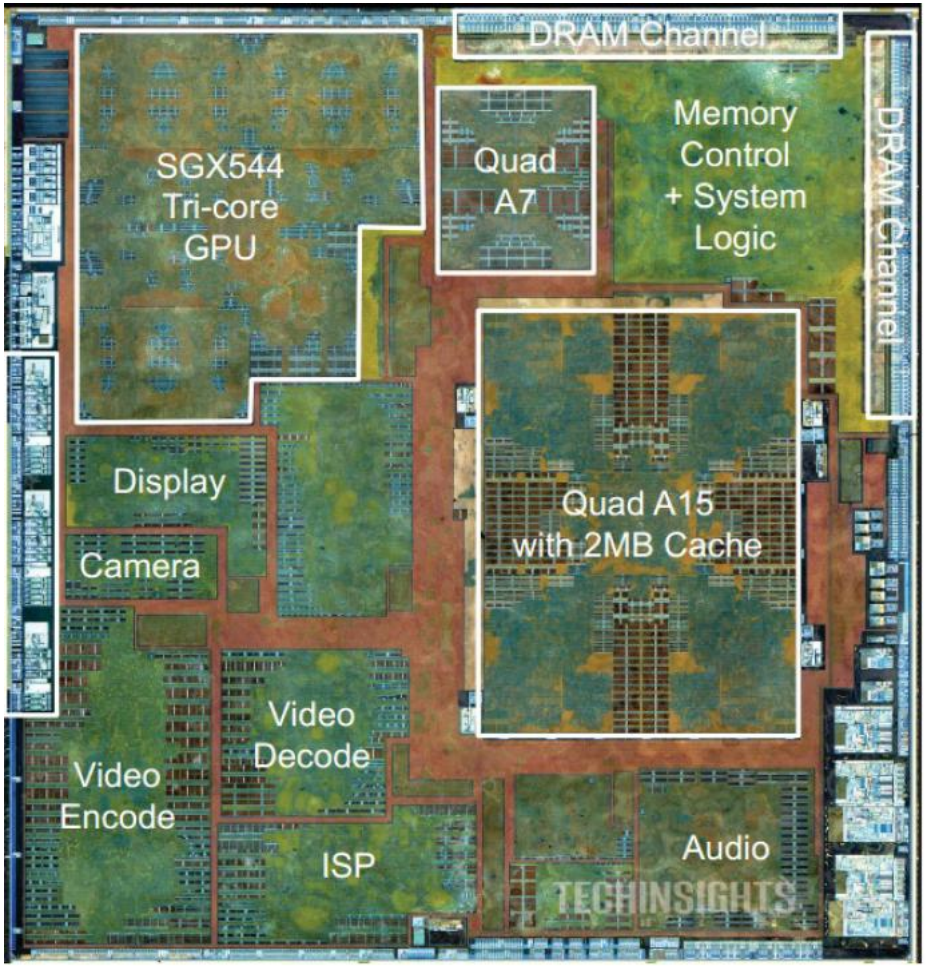
Samsung Galaxy S4 – Exynos 5410

Paměť DRAM při pohledu shora

Zdroj: http://www.embedded-vision.com/platinum-members/embedded-vision-alliance/embedded-vision-training/documents/pages/computational-photography-part-2

Samsung Galaxy S4 – Exynos 5410 – procesory

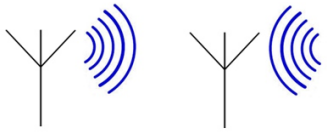
Řez čipem Exynos 5410 (v jiné úrovni)



- Všimněte si rozdílných velikostí 4 jáder A7 a 4 jáder A15
- Na čipu jsou mimo vlastního procesoru integrovány i další součásti: GPU, Video coder a decoder a další. Jedná se tedy o **SoC** (System on Chip)

Zdroj: http://www.embedded-vision.com/platinum-members/embedded-vision-alliance/embedded-vision-training/documents/pages/computational-photography-part-2, http://gamma0burst.tistory.com/m/600

Samsung Galaxy S4 – propojení komponent



NAND flash (16GB)

Accelerometer | **GPS** | **Wi-fi** | **Baseband processor**

Application processor: Exynos

Memory I/F (LPDDR3, eMMC, SD)

Peripheral I/F

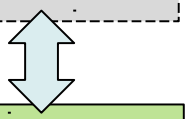
CPU Cortex A15 Quad core

CPU Cortex A7 Quad core

GPU SGX544 Tri core

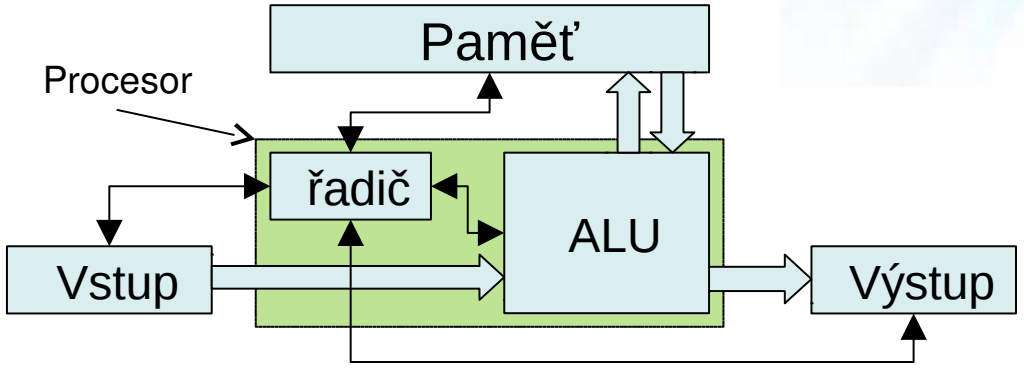
ISP

Camera | **Display** | **High speed I/F (HSIC/ USB)** | **Audio**



DSP procesor pro audio

Společný koncept

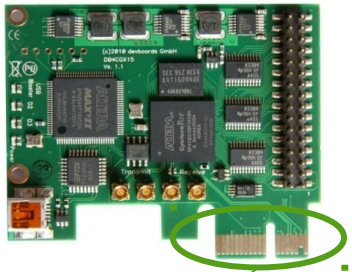


- Procesor vykonává instrukce uložené v paměti (ROM, RAM) tak, aby obsluhoval periferie – reagoval na vnější události a zpracovával data.

Přehled témat přednášek

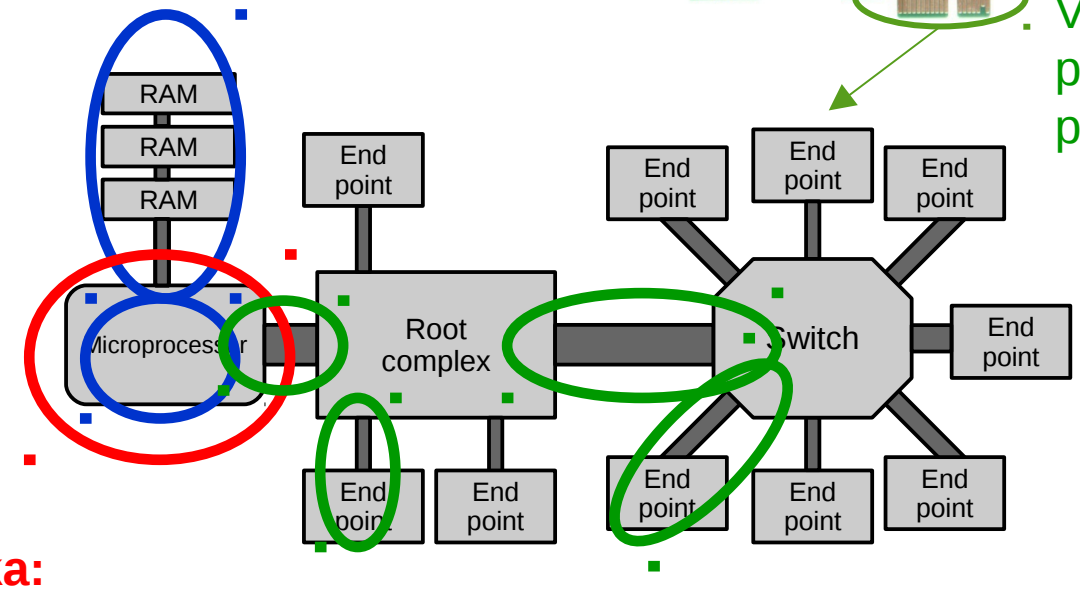
3. přednáška:
Paměťový subsystém – hierarchie pamětí. Statická a dynamická paměť

4. přednáška:
Paměťový subsystém – virtuální paměť



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

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



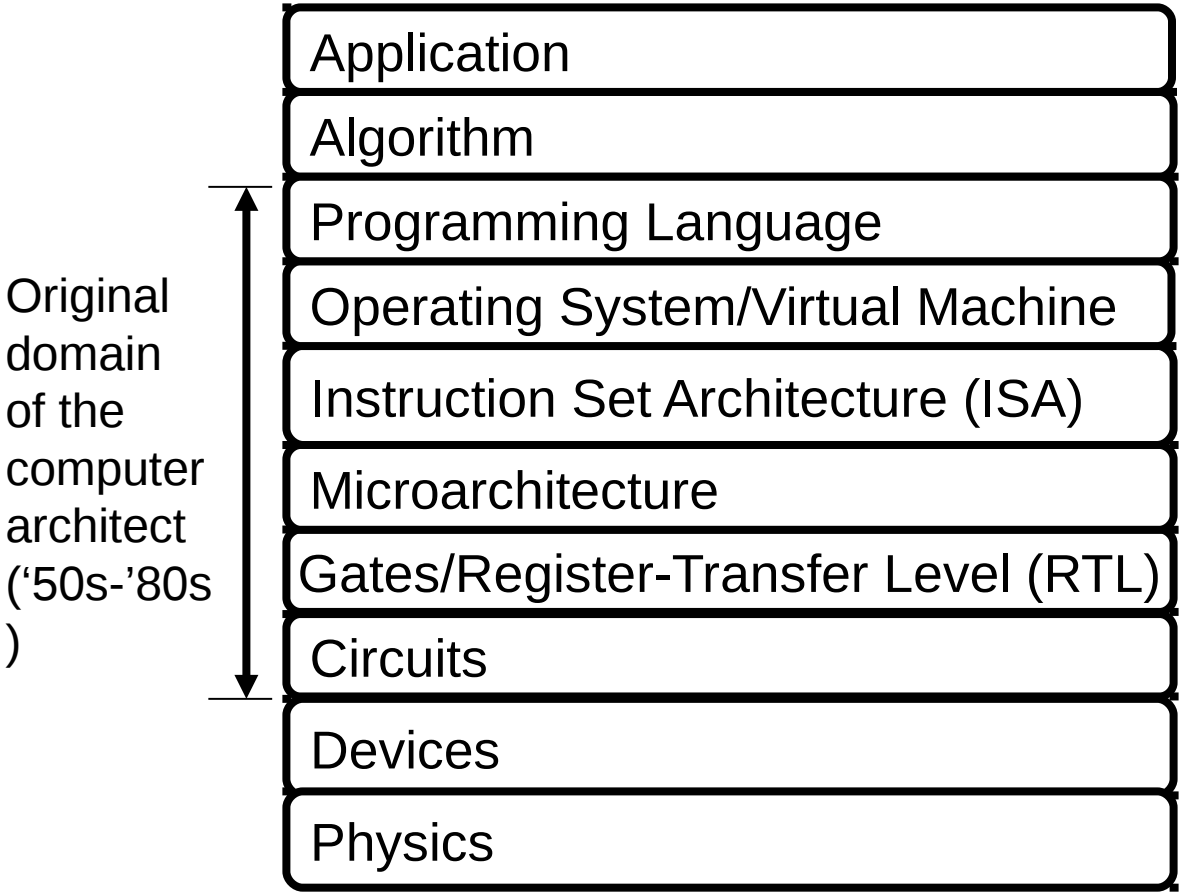
5. přednáška:
Princip zřetězeného zpracování instrukcí, Řešení hazardů uvnitř CPU

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

Přehled témat přednášek

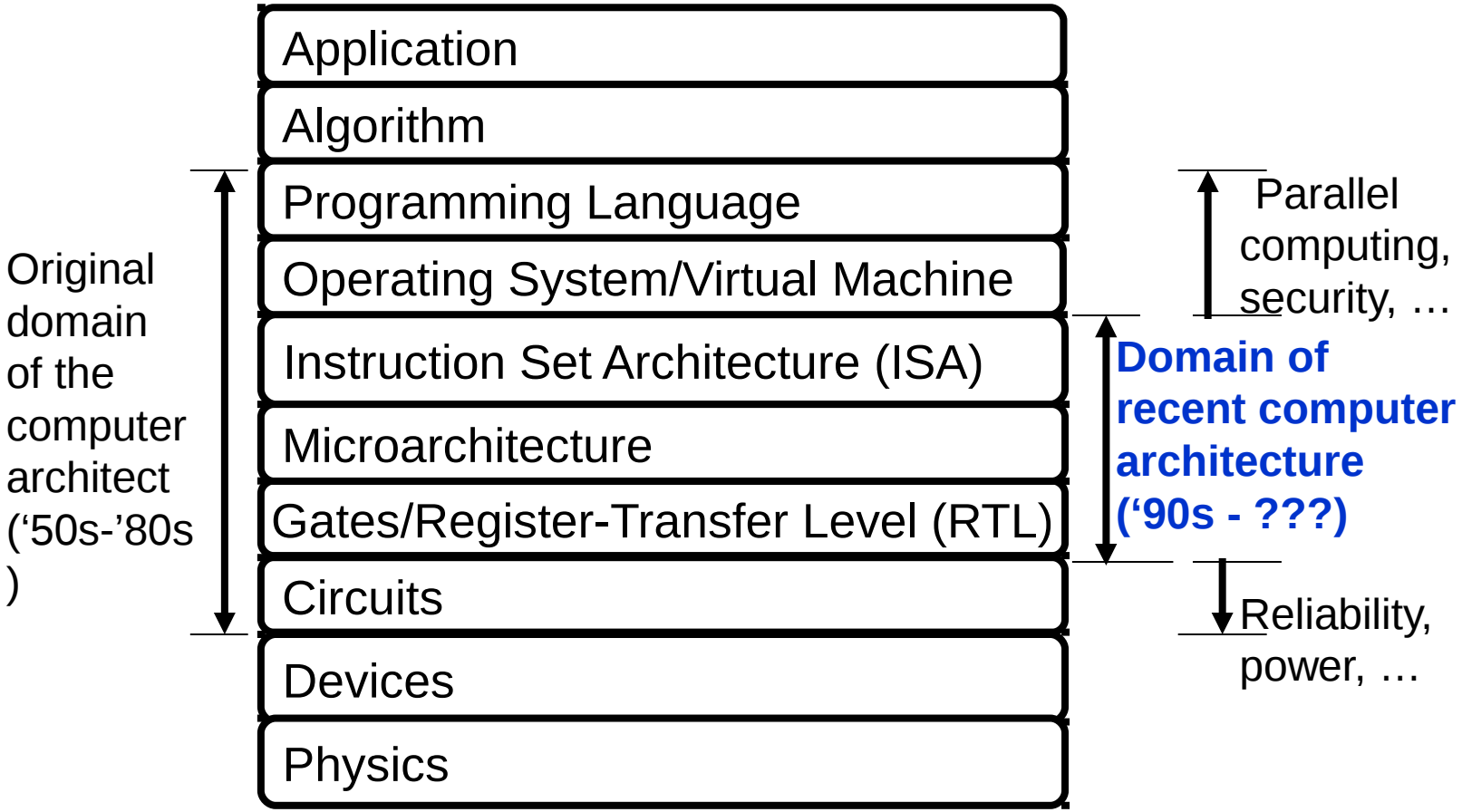
8. Předávání parametrů funkcím a virtuálním instrukcím operačního systému
9. Technické a organizační prostředky (vnější události, výjimky, reálný čas)
10. Sítě procesorů a počítačů
11. Klasická registrově orientované architektury CISC
12. Procesorová rodina INTEL x86, Od 8086 k EMT64
13. Přehled vývoje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

Co je to architektura počítače?



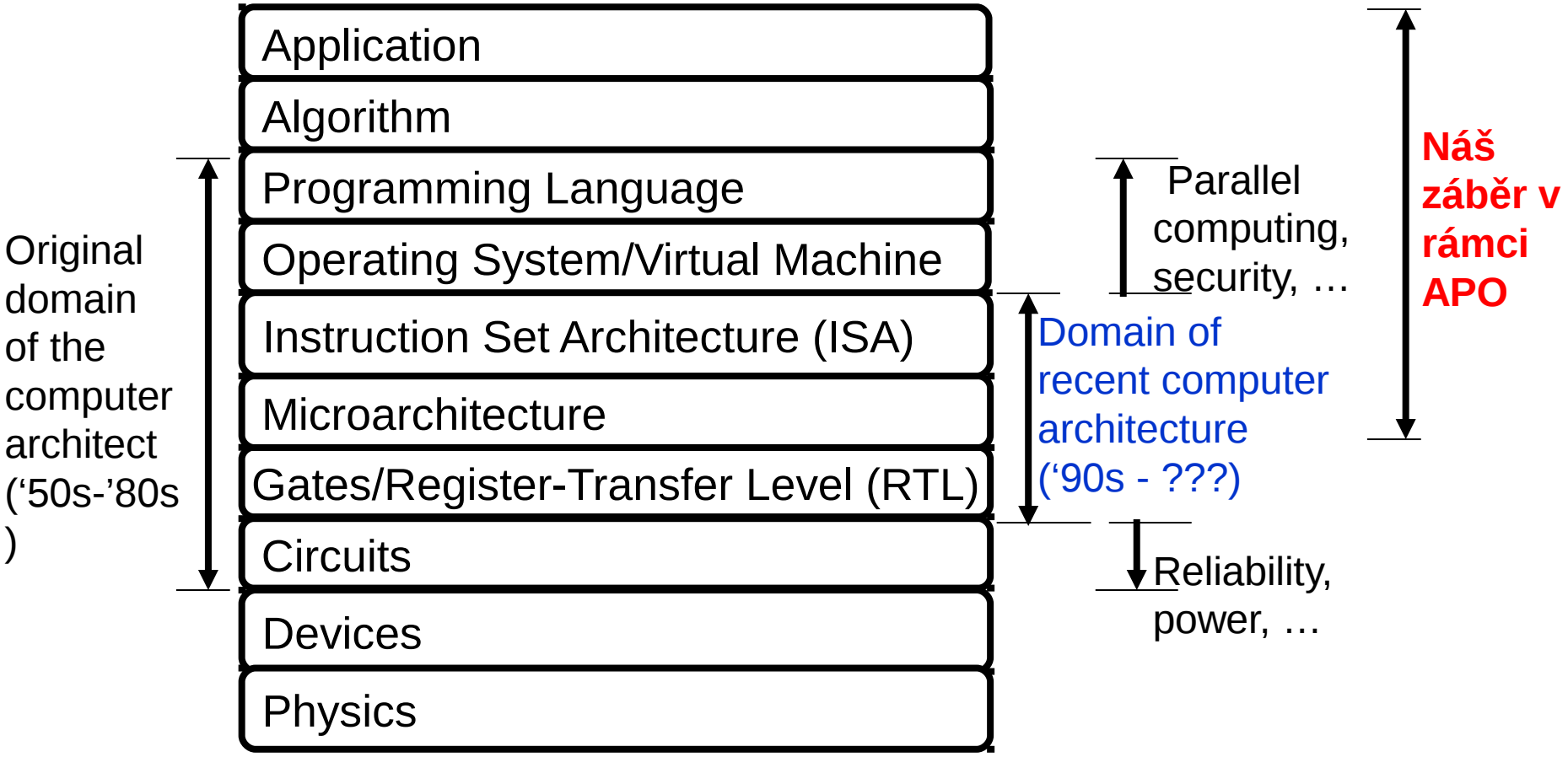
Reference: John Kubiatowicz: EECS 252 Graduate Computer Architecture, Lecture 1. University of California, Berkeley

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

Hodnocení a podmínky absolvování

Zápočet:

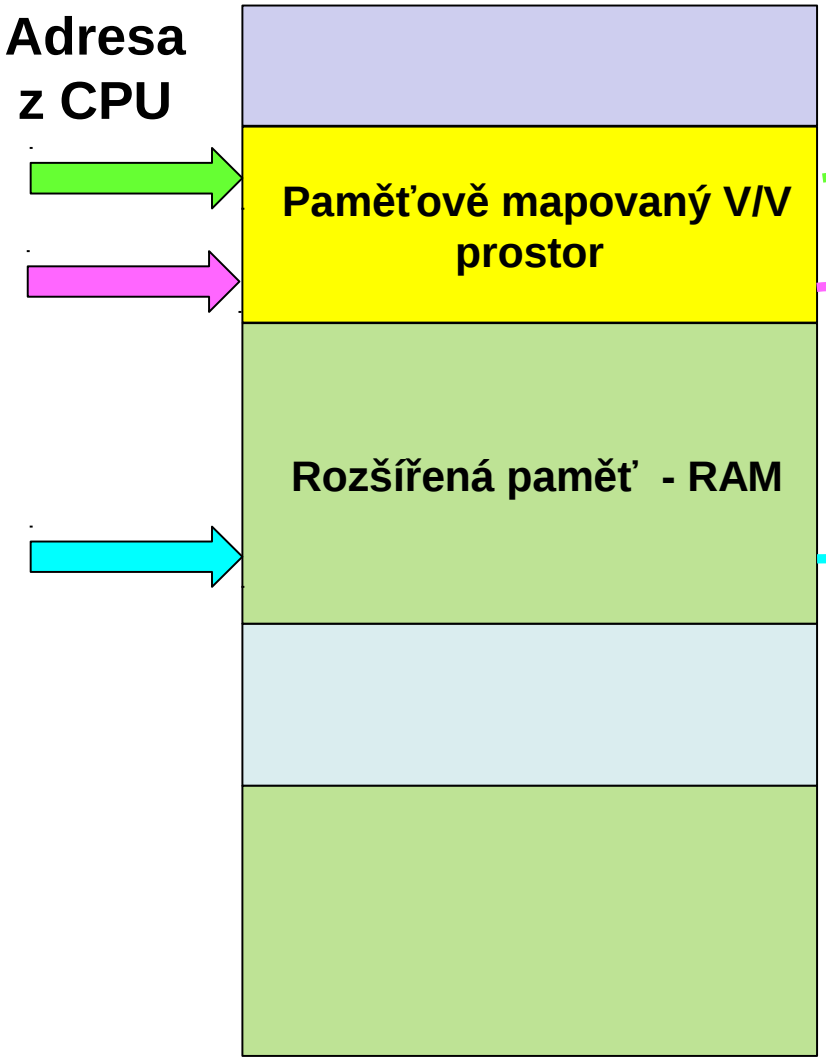
Kategorie	Body	Nutné minimum
4 domácí úkoly	24	10
Hlavní test (9. přednáška)	20	10
Týmový projekt	20	5
Celkem	**64**	

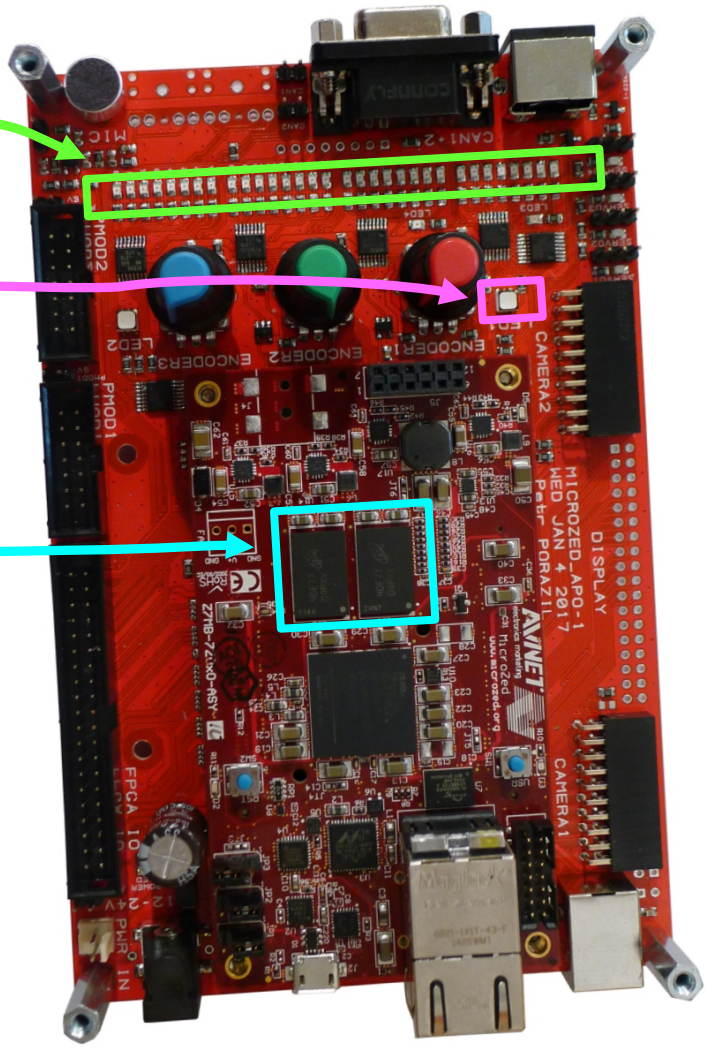
Zkouška:

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

První cvičení – fyzický adresní prostor na MZ_APO





Přístup k LED diodám z C programu na MZ_APO

```
int main(int argc, char *argv[])
{
  unsigned char *mem_base; /* virtuální adresa oblasti registrů */
  volatile unsigned char *led_port; /* adresa portu s diodou*/

  mem_base =
     map_phys_address(LED_REG_BASE_PHYS, SPILED_REG_SIZE, 0);
  led_port = mem_base + SPILED_REG_LED_LINE_o;

  While(1) {
    *led_port = 1; /* rozsviť diodu */
    sleep(1);
    *led_port = 0; /* zhasni diodu */
    sleep(1);
  }
  return 0;
}
```

Obsah látky 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

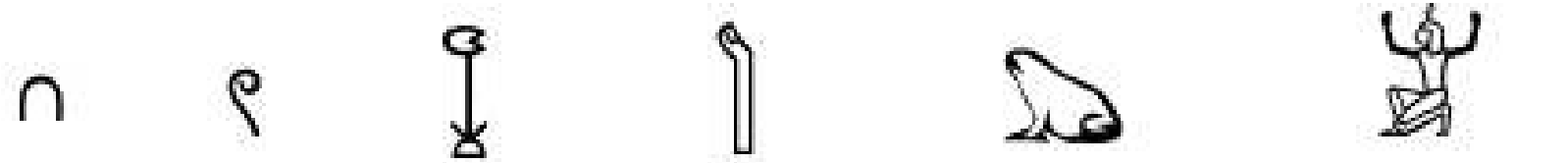
Číselná soustava:

- **Nepoziční číselná soustava** - hodnota číslice není dána jejím umístěním v dané sekvenci číslic, ale jejím vzhledem (zápisem/symbolem)

http://diameter.si/sciquest/E1.htm

- Hodnota čísla může být dána prostým součtem hodnot jednotlivých číslic (Egyptské číslice) nebo je zapotřebí použít nějaká pravidla (Římské číslice)

Poziční číselná soustava



Hodnota čísla tedy je: 1 333 331

Základní terminologie

- **Číselná soustava**:
- **Poziční číselná soustava** - hodnota každé číslice je dána její pozicí v sekvenci symbolů
  - Množina všech symbolů (číslic) se nazývá abeceda
  - Celá část je oddělena od zlomkové speciálním znakem (řádová čárka/tečka)

Λ – abeceda - Například {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} pro desítkovou soustavu

z – základ (radix) soustavy – obvykle přirozené číslo >1

$a \in \Lambda$ - číslice

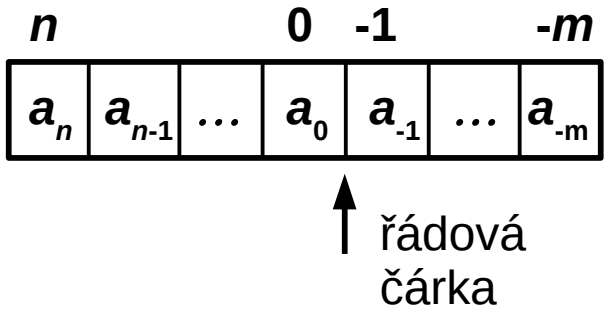
$$A \sim a_n a_{n-1} \dots a_0, a_1 \dots a_{-m}$$
$$A = a_n z^n + a_{n-1} z^{n-1} + \dots + a_0 + a_1 z^{-1} \dots a_{-m} z^{-m}$$

Poziční číselná soustava

Například dekadické číslo 348,31

má hodnotu **3***10^{2} + **4***10^{1} + **8***10^{0} + **3***10^{-1} + **1***10^{-2}

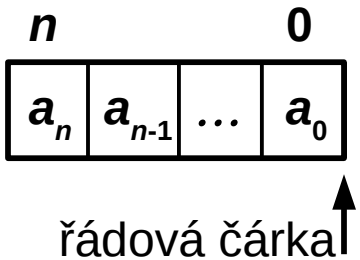
- Pokud si zvolíme **fixně** počet pozic pro celočíselnou část (n+1) a počet pozic pro zlomkovou část (m), bude:



- Nejmenší zobrazitelné číslo: $\varepsilon = z^{-m}$,
- Modul - nejmenší hodnota, kterou již neumíme zobrazit: $M=z^{n+1}$
- Zobrazitelná čísla tedy leží v rozsahu: $0 \leq A < M$

Uložení čísel typu INTEGER **bez znaménka**

- Zvolme si tedy celkově například **8 pozic**, z toho všechny pro celočíselnou část a základ soustavy roven z = 2.

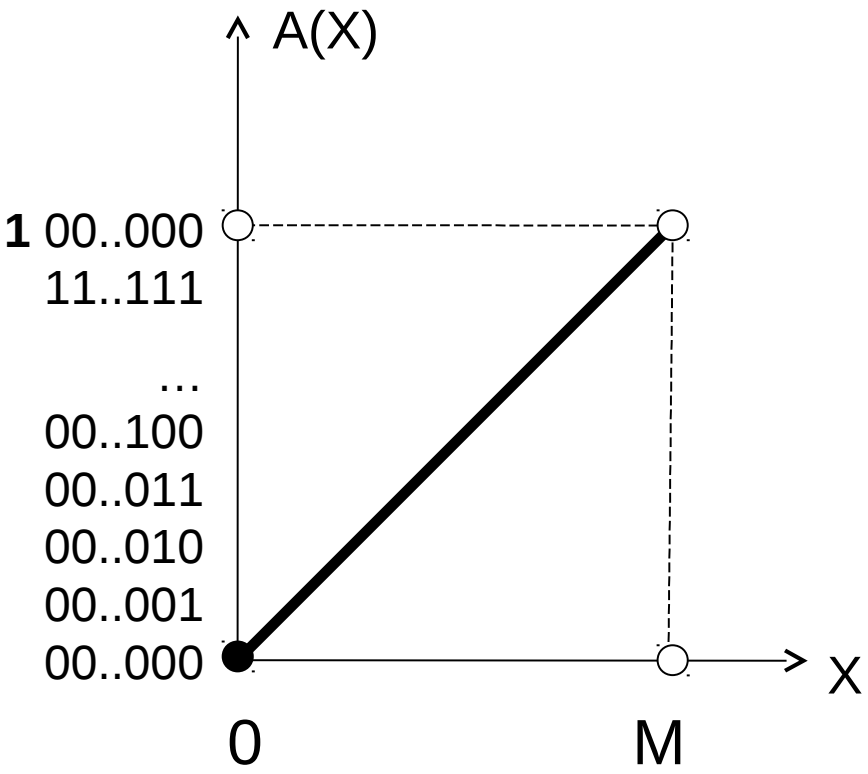


- $2^8 = 256_D$ (desítkově). Rozsah je příliš malý
  - Řešení: proč nepoužít více bajtů?
  - $4B = 2^{32} = 4\ 294\ 976\ 296_D$,
  - Rozsah tedy je: $<0, 2^{n+1}-1>$, nebo pokud N bude počet bitů: $<0, 2^N-1>$

Binární hodnota	Neznaménková reprezentace
00000000	$0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$128_{(10)}$
10000001	$129_{(10)}$
⋮	⋮
11111101	$253_{(10)}$
11111110	$254_{(10)}$
11111111	$255_{(10)}$

Uložení čísel typu INTEGER **bez znaménka**

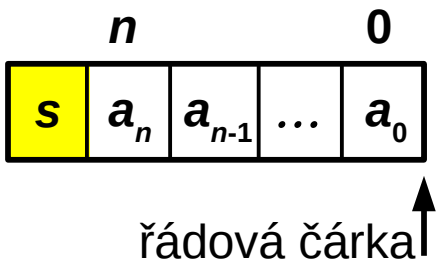
Pořadové číslo kombinace bitů
Binární reprezentace čísla



Vyjadřovaná hodnota
Její matematický význam

Binární hodnota	Neznaménková reprezentace
00000000	$0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$128_{(10)}$
10000001	$129_{(10)}$
⋮	⋮
11111101	$253_{(10)}$
11111110	$254_{(10)}$
11111111	$255_{(10)}$

Uložení čísel typu INTEGER **se znaménkem**



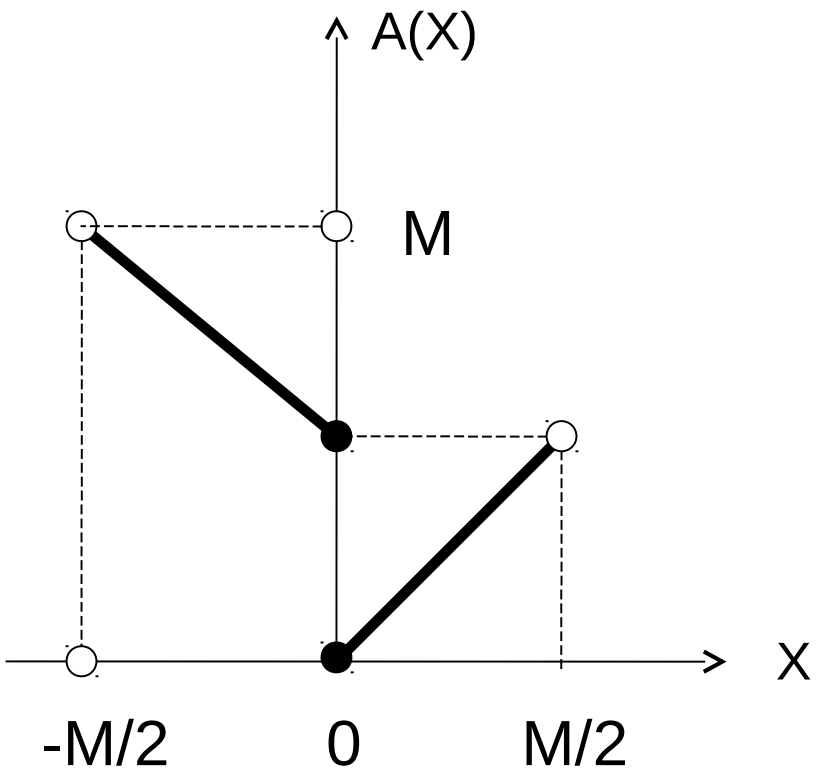
- Znaménko a hodnota. Jde o **přímý kód**.
- Běžně dodržovaná dohoda:
  - 0 ≈ +, 1 ≈ -.
- Nevýhoda: jinak se musí při aritmetických operacích pracovat se znaménkovým bitem, jinak s bity hodnoty.
- Jiná nevýhoda: máme 2 různá vyjádření nuly.

Binární hodnota	Přímý kód
00000000	$+0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-0_{(10)}$
10000001	$-1_{(10)}$
10000010	$-2_{(10)}$
⋮	⋮
11111101	$-125_{(10)}$
11111110	$-126_{(10)}$
11111111	$-127_{(10)}$

Uložení čísel typu INTEGER **se znaménkem**

Přímý kód – pokračování…

- Pokud N bude počet bitů:
  $<-2^{N-1} -1, 2^{N-1} -1>$



Binární hodnota	Přímý kód
00000000	$+0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-0_{(10)}$
10000001	$-1_{(10)}$
10000010	$-2_{(10)}$
⋮	⋮
11111101	$-125_{(10)}$
11111110	$-126_{(10)}$
11111111	$-127_{(10)}$

Propojení hardware, signálů a software

- V programovacím jazyce C se celým číslům 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.

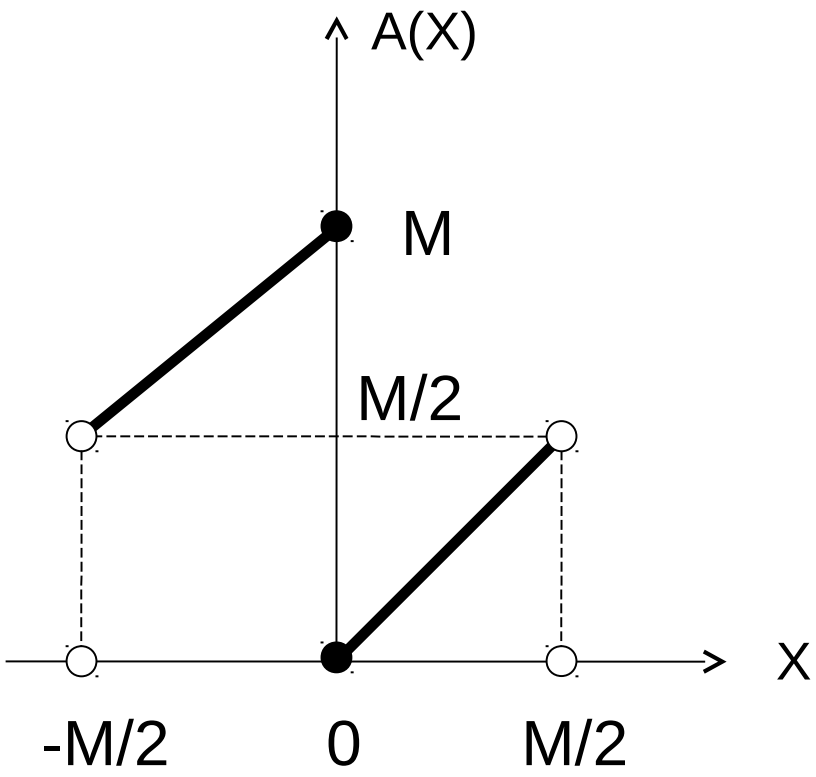
- **Inverzní kód** – jedničkový doplněk (one's complement):
- Výhodnější varianta něž přímý kód
- Proč výhodnější? Úspora v HW!
- Jedničkový doplněk **záporného** čísla se (ve dvojkové soustavě) vytvoří bitovou negací čísla **kladného** => inverze všech bitů (jedničkový doplněk)

Dekadická hodnota	**Reprezentace v inverzím kódu na 4 bity**
6	0110
-6	1001

Uložení čísel typu INTEGER **se znaménkem** II.

Inverzní kód – pokračování…

- Pokud N bude počet bitů:
  $<-2^{N-1} -1, 2^{N-1} -1>$



Binární hodnota	Inverzní kód
00000000	$0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-127_{(10)}$
10000001	$-126_{(10)}$
10000010	$-125_{(10)}$
⋮	⋮
11111101	$-2_{(10)}$
11111110	$-1_{(10)}$
11111111	$-0_{(10)}$

Čísla INTEGER **se znaménkem** III.

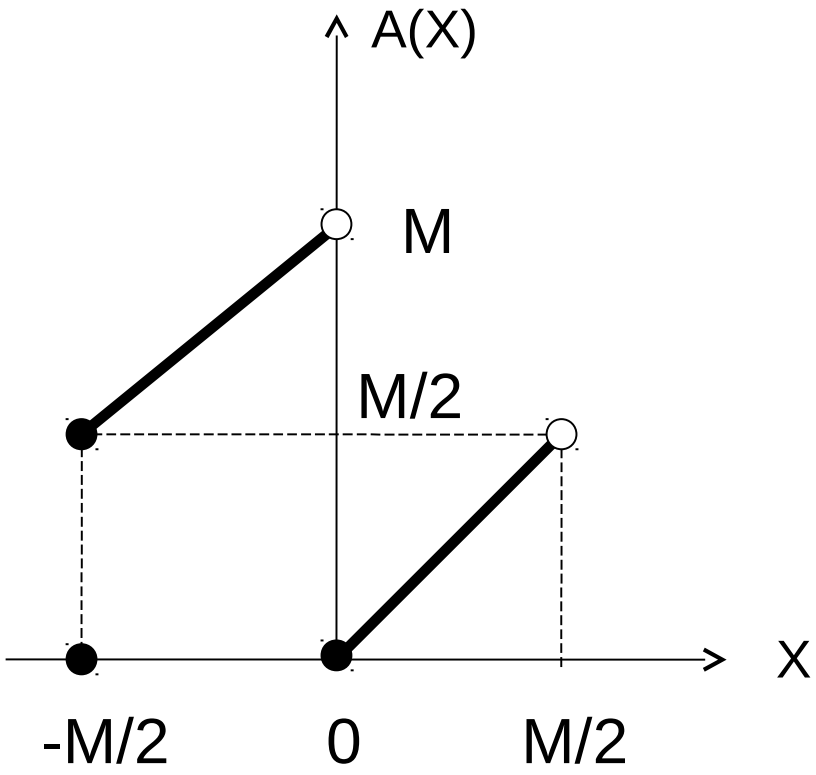
Dvojkový doplněk (two's complement):

- Výhodnější varianta něž inverzní kód
- Proč výhodnější? Úspora v HW!
- Často se označuje **nepřesně** jako „**doplňkový kód**“
- Dvojkový **doplněk** záporného čísla se (ve dvojkové soustavě) připočtením 1 k nejnižšímu bitu záporného čísla reprezentovaného v inverzním kódu
- Jedničkový se od dvojkového doplňku se liší málo, jen o jedničku
- Na rozdíl od jednotkového **inverzního kódu** nevyžaduje přičítat horkou jedničku (Hot-One) při přechodu ze záporné na kladnou hodnotu. Stejná binární sčítačka je použitelná pro signed a unsigned.

Dekadická hodnota	Reprezentace v dvojkovém doplňku na 4 bity
6	0110
-6	1010

Čísla INTEGER **se znaménkem** III.

- **Dvojkový doplněk** – pokračování…
- Pokud N bude počet bitů: $<-2^{N-1}, 2^{N-1}-1>$



Binární hodnota	Dvojkový doplněk
00000000	$0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-128_{(10)}$
10000001	$-127_{(10)}$
10000010	$-126_{(10)}$
⋮	⋮
11111101	$-3_{(10)}$
11111110	$-2_{(10)}$
11111111	$-1_{(10)}$

Doplňkový kód - příklady

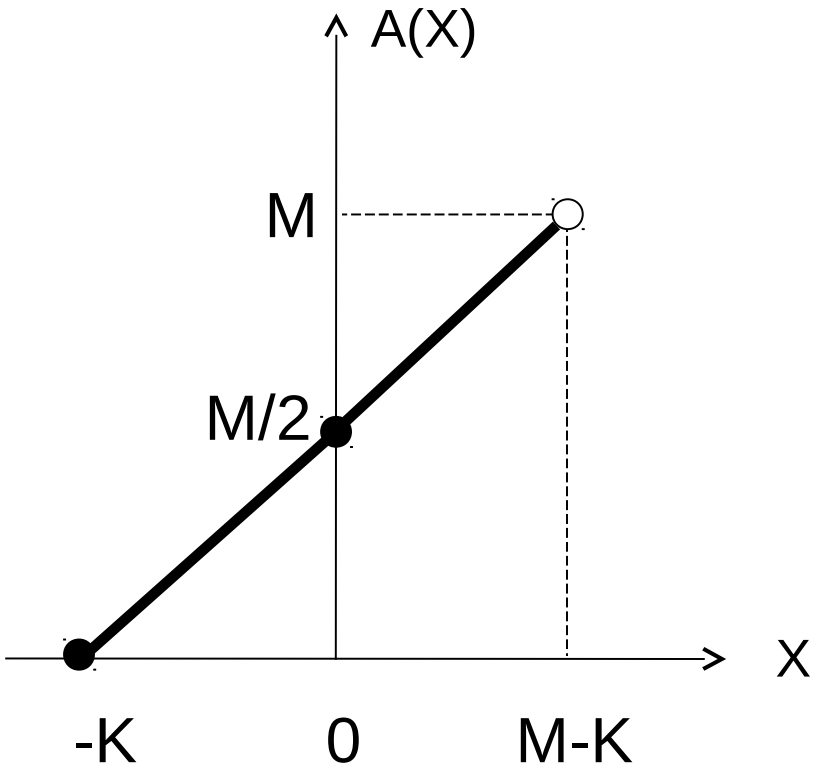
- Příklady reprezentací:

$0_D = 00000000_H$	
$1_D = 00000001_H$	$-1_D = FFFFFFFF_H$
$2_D = 00000002_H$	$-2_D = FFFFFFFE_H$
$3_D = 00000003_H$	$-3_D = FFFFFFFD_H$

- Analogii dvojkového doplňku 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 (bit 32) ignorujeme. Sčítáme vlastně mod 2^{32} .
- Jak je to vlastně v tomto kódování s **přeplněním** (přetečením) rozsahu? Budeme diskutovat později…

Čísla INTEGER **se znaménkem** IV.

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



Obvykle volíme
K = ½ M - 1

Připomenutí: M je modul

Sčítání a odčítání v aditivním kódu

- Platí:

$$\mathcal{A}(A+B) = \mathcal{A}(A) + \mathcal{A}(B) - K$$
$$\mathcal{A}(A-B) = \mathcal{A}(A) - \mathcal{A}(B) + K$$

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

Jak se v počítači zobrazují čísla typu REAL?

- Vědecká, neboli semilogaritmická notace.
  - Dvojice: EXPONENT (E), ZLOMKOVÁ část (nazývaná též mantisa M).
  - **Mantisa x zaklad**$^{\text{Exponent}}$
- Notace je normalizovaná.
  - Zlomková část vždy začíná binární číslicí 1,
  - Obecně: nenulovou číslicí <1, z - 1>.
- Dekadicky: $7{,}26478 \times 10^{3}$
- Binárně: $1{,}010011 \times 2^{1001}$

Propojení hardware, signálů a software

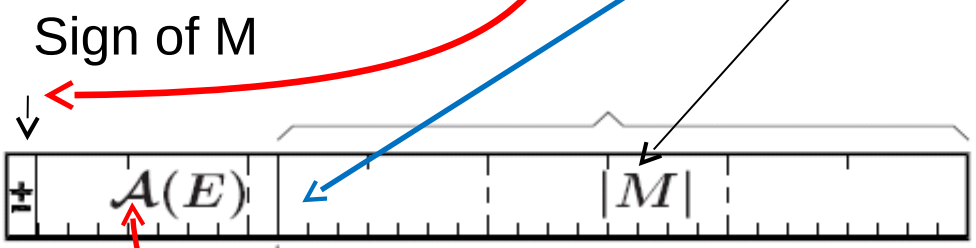
- Reprezentace je definovaná normou **IEEE754** ve verzích
  - jednoduchá (32 bitů)
  - dvojnásobná přesnost (64 bitů)
  - Nově (IEEE 754-2008) i poloviční (16 bitů – především pro hry a barvy), a čtyřnásobná (128 bitů) a osminásobná přesnost (256 bitů) pro speciální vědecké výpočty
- V programovacím jazyce C se proměnné s jednoduchou a dvojnásobnou přesností deklarují jako **`float`** a **`double`**.

(De)normalizovaná čísla v desítkové a dvojkové soustavě

- -2.34×10^{56} ← normalized
- $+0.002 \times 10^{-4}$ ← not normalized
- $+987.02 \times 10^{9}$ ← not normalized

binárně

- $\pm 1.xxxxxxx_2 \times 2^{yyyy}$



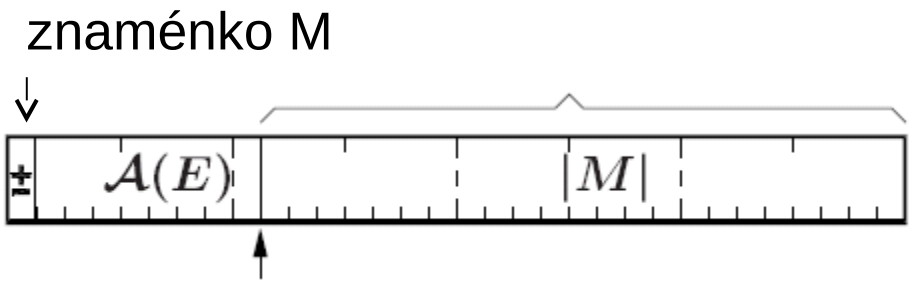
Poloha řádové čárky (tečky) pro E i M

Reprezentace/kódování čísla v pohyblivé řádové čárce

- Kód mantisy: přímý kód – znaménko a absolutní hodnota
- Kód exponentu: aditivní kód (s posunutou nulou) (K=127 pro jednoduchou přesnost)
- Implicitní počáteční jednička může být pro normalizovanou mantisu vynechaná $m \in \langle 1, 2)$
  rozlišení 23+1 implicitní bit pro jednoduchou přesnost

$$X = -1^{s}\ 2^{A(E)-127}\ m \qquad \text{kde } m \in \langle 1, 2)$$

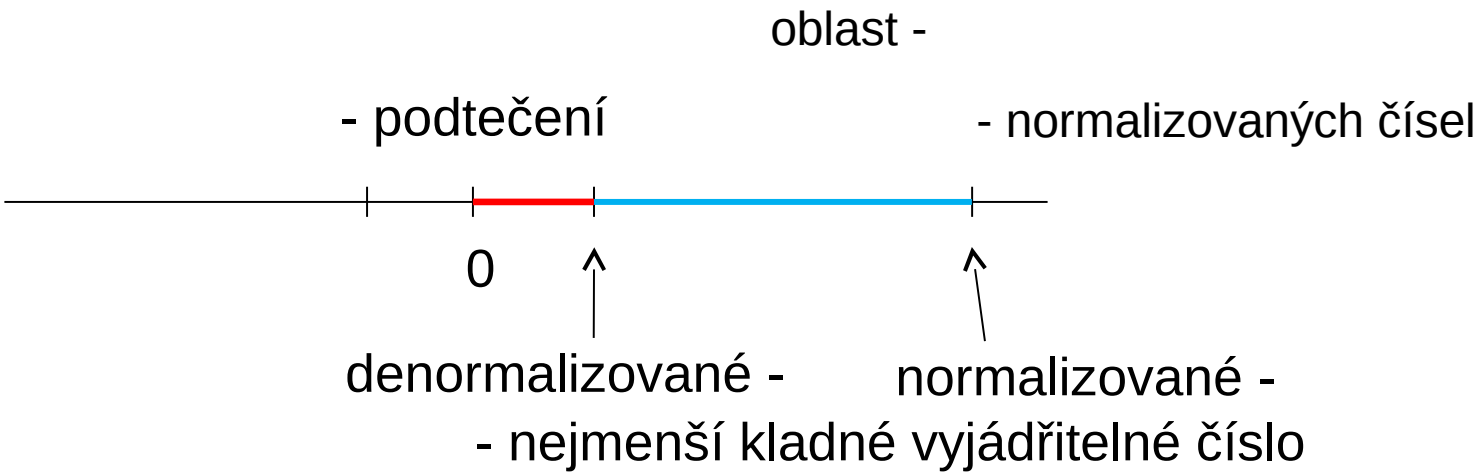
$$m = 1 + 2^{-23}\ M$$



Poloha řádové čárky (tečky) pro E i M

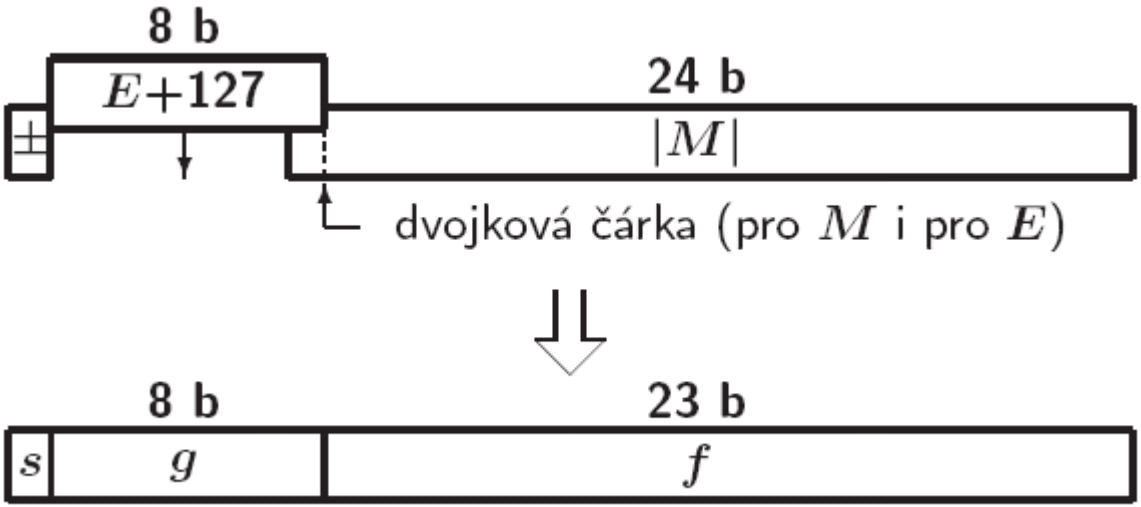
Implicitní (skrytá) počáteční jednička

- Pro každé normalizované číslo je nejvýznamnější bit mantisy jedna a není ho potřeba ukládat (rezervovat pro něj místo)
- Pokud je reprezentace exponentu 0 (-K) nebo pokud je číslo „denormalizované“, tak je prostor pro uložení mantisy využitý pro hodnotu včetně počáteční jedničky nebo nuly
- Denormalizovaná čísla umožňují zachovat rozlišení v rozsahu od nejmenšího normalizovaného čísla směrem k nule



ANSI/IEEE Std 754-1985 (2008) – 32b a 64b formát

ANSI/IEEE Std 754-1985 — jednoduchý formát — 32b



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

g . . . 11b *f* . . . 52b

Příklady reprezentace některých důležitých hodnot

Nula

kladná	**0 00000000 00000000000000000000000**	**+0.0**
záporná	**1 00000000 00000000000000000000000**	**-0.0**

Nekonečno

kladné	**0 11111110 11111111111111111111111**	**+Inf**
záporné	*** 00000001 00000000000000000000000**	**-Inf**

Hraniční hodnoty pro jednoduchý formát

Největší normalizované	**0 11111110 11111111111111111111111**	$(2-2^{-23})2^{(127)}$ **+3.4028 10^{+38}**
Nejmenší normalizované	*** 00000001 00000000000000000000000**	$\pm 2^{(1-127)}$ $\pm 1.1755\ 10^{-38}$
Největší denormalizované	*** 00000000 11111111111111111111111**	$\pm(1-2^{-23})2^{(1-126)}$
Nejmenší denormalizované	*** 00000000 00000000000000000000001**	$(2-2^{-23})2^{(127)}$ **+3.4028 10^{+38}**

Speciální hodnoty NaN, +Inf a -Inf

- Pokud není výsledek matematické operace pro daný vstup definovaný (log -1) nebo je výsledek nejednoznačný 0/0, +Inf - +Inf tak je uložena hodnota NaN (Not-a-Number) – exponent nastavený na samé jedničky, mantisa nenulová
- Výsledkem operací, které pouze přetečou z rozsahu 1/0 (=+Inf), +Inf + +Inf (= +Inf) atd., je reprezentovaný hodnotou nekonečno (+Inf nebo -Inf) – exponent samé jedničky, mantisa nuly

Shrnutí – čísla a výjimky pro jednoduchou přesnost

s-bit	obraz exponentu	m	M	význam
0	0 < e < 255	1≤ m <2	*	normalizované kladné číslo
1	0 < e < 255	1≤ m <2	*	normalizované záporné číslo
0	0	m > 0	≠0	denormalizované kladné (blízko nuly)
1	0	m > 0	≠0	denormalizované záporné (blízko nuly)
0	0	0	0	kladná nula
1	0	0	0	záporná nula
0	255		=0	kladné nekonečno
1	255		=0	záporné nekonečno
0	255		≠0	NaN – nečíselná/nevyjádřitelná hodnota
1	255		≠0	NaN – nečíselná/nevyjádřitelná hodnota

Porovnání dvou čísel ve FP

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

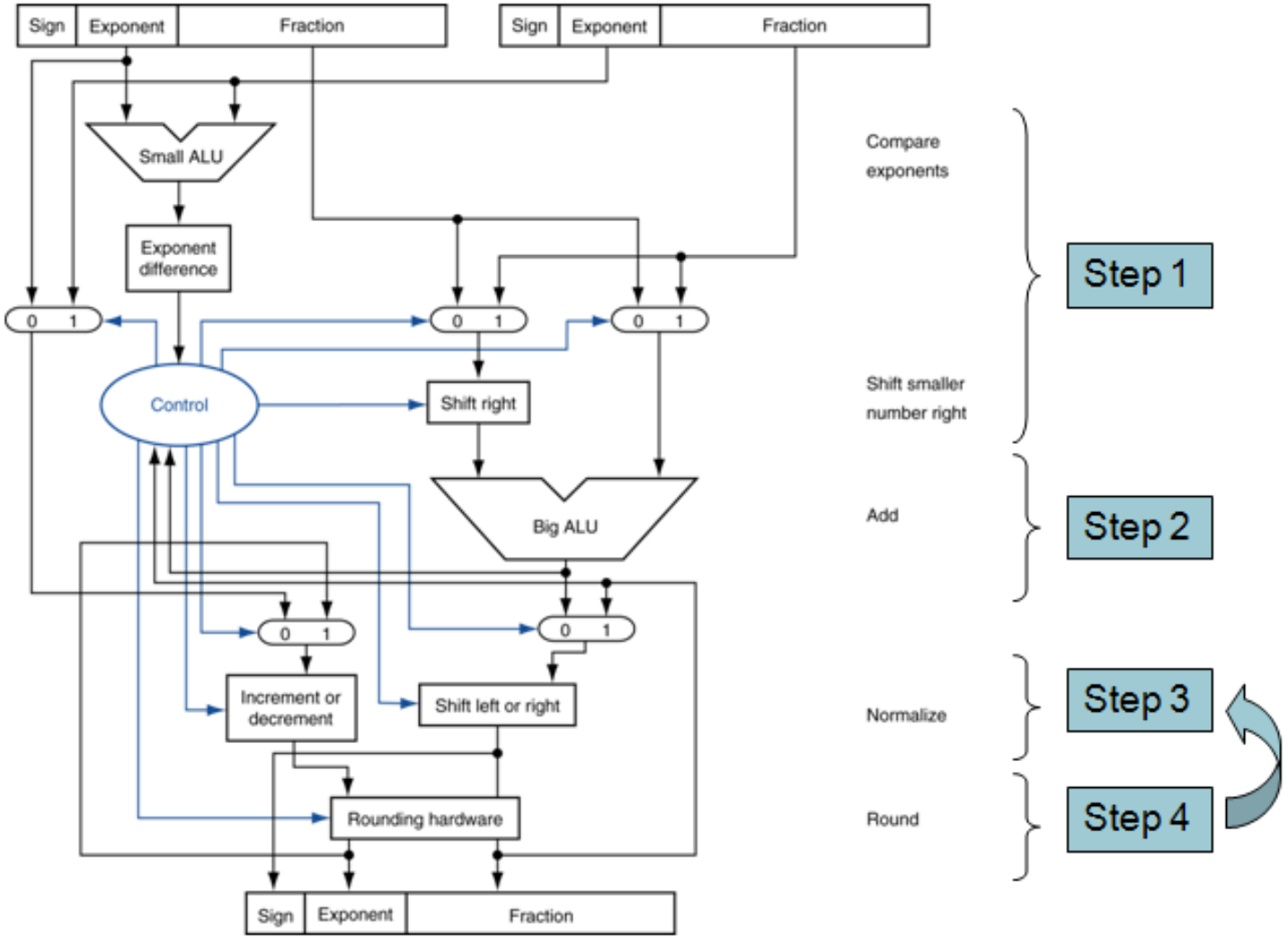
- Nebo je ještě jiný způsob?

Algoritmus sčítání v pohyblivé řádové čárce

- $Ma \times 2^{Ea} + Mb \times 2^{Eb}$

- 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.

Hardware sčítačky v pohyblivé řádové čárce



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.

Iterační dělička - Goldschmidt

$$Q = \frac{N}{D} = \frac{m_N 2^{e_N}}{m_D 2^{e_D}} = \frac{m_N}{m_D} 2^{e_N - e_D}$$

Pokud jsou čísla v normalizovaném tvaru, pak platí:

m_N = 1.???????...? a m_D = 1.???????...?

tzn. $1 \leq m_N, m_D < 2$ pokud uvažujeme celou mantisu, nebo $0{,}5 \leq m_N, m_D < 1$ pokud bereme pouze zlomkovou část.

Uvažujme pouze zlomkovou část (za desetinnou čárkou).

Zřejmě můžeme m_D přepsat do tvaru: $m_D = 1-x$, kde $0 < x \leq 0{,}5$

Jakou hodnotu má x? Spočítejme ji: $x = 1 - m_D$

Zvolme $F_0 = 1+x$. Všimněme si jakou vlastnost bude mít $m_D * F_0$

Iterační dělička - Goldschmidt

Pokud $F_0=1+x$. Potom $m_D*F_0 = (1-x)*(1+x) = 1-x^2$

$$Q = \frac{N}{D} = \frac{m_N 2^{e_N}}{m_D 2^{e_D}} = \frac{m_N}{m_D} 2^{e_N - e_D} = \frac{m_N F_0}{m_D F_0} 2^{e_N - e_D}$$

Pokud budeme volit Fi tak, aby jmenovatel konvergoval k 1, bude čitatel konvergovat k podílu mantis.

Zvolme $F_1=1+x^2$. Potom $m_D*F_0*F1 = (1-x^2)*(1+x^2) = 1-x^4$

$$\frac{m_N}{m_D} = \frac{m_N F_0 F_1 F_2 F_3 \ldots}{m_D F_0 F_1 F_2 F_3 \ldots} = \frac{m_N F_0 F_1 F_2 F_3 \ldots}{\approx 1} \approx m_N (1+x)(1+x^2)\ldots(1+x^{2^i})$$

Iterační dělička – Goldschmidt – vylepšená verze

- S rostoucím x ($0 < x \leq 0{,}5$) se konvergence zhoršuje. Pokud x == 0.5 je nejhorší. Jinými slovy, pro fixní počet iterací se snižuje přesnost výsledku.

- Modifikace Goldschmidtova algoritmu spočívá v odhadu (nepřesném) převrácené hodnoty K hodnoty m_D z look-up tabulky – podle několika málo prvních bitů (~10).
- Místo původního x=1- m_D počítáme s x=1- Km_D

$$\frac{m_N}{m_D} \approx m_N K(1+x)(1+x^2)...(1+x^{2^i})$$

- Tato dělička se používá v moderních CPU.
- Kontrolní otazka: Můžeme tuto deličku použít i pro INTEGER?

Zjednodušený přehled operací v plovoucí řádové čárce

Sčítání: $A \cdot z^a$, $B \cdot z^b$, $b < a$ — sjednocení exponentů

$B \cdot z^b = (B \cdot z^{b-a}) \cdot z^{b-(b-a)}$ — posunem mantisy

$A \cdot z^a + B \cdot z^b = [A+(B \cdot z^{b-a})] \cdot z^a$ — sečtení + normalizace

Odčítání: sjednocení exponentů, odečtení a normalizace

Násobení: $A \cdot z^a \cdot B \cdot z^b = A \cdot B \cdot z^{a+b}$

$A \cdot B$ - normalizace je-li třeba

$A \cdot B \cdot z^{a+b} = A \cdot B \cdot z \cdot z^{a+b-1}$ posunem doleva

Dělení $A \cdot z^a / B \cdot z^b = A/B \cdot z^{a-b}$

A/B - případná normalizace

$A/B \cdot z^{a-b} = A/B \cdot z \cdot z^{a-b+1}$ - posunem doprava

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?

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

$$\sum_{i=1}^{10^{10}} \frac{1}{i^2} \approx 1.644934057830186 5,$$

$$\sum_{i=10^{10}}^{1} \frac{1}{i^2} \approx 1.644934066748226 4$$

typ double pro oba případy.. Proč se liší?

Překvapení na závěr ???

```
#include <stdio.h>
int main()
{
  float x;
  x = 116777215.0;  printf("%.3lf\n", x);
  x = 116777216.0;  printf("%.3lf\n", x);
  x = 116777217.0;  printf("%.3lf\n", x);
  x = 116777218.0;  printf("%.3lf\n", x);
  x = 116777219.0;  printf("%.3lf\n", x);
  x = 116777220.0;  printf("%.3lf\n", x);
  x = 116777221.0;  printf("%.3lf\n", x);
  return 0;
}
```