Programování na strojové úrovni

Cvičení 1: Záporné reprezentace čísel a úvod do programování

Tomáš Goldmann, igoldmann@fit.vutbr.cz
www.fit.vutbr.cz/~igoldmann/education

Rev.: 1A/2019

Jak to bude probíhat?

- Cvičení budou probíhat každý týden (celkem 13) ve **3** budou praktické testy
- 1. test-**6b**, 2. test **11b**, 3. test **7b**, celkem za cvičení **24 bodů**
- Zápočet za **20 bodů**
- Možnost získat bonusové body
- Účast na cvičeních žádaná

Konzultace:

- cvičení, příklady (Ing. Tomáš Goldmann, igoldmann@fit.vutbr.cz nebo osobně po domluvě e-mailem)
- organizační záležitosti (Ing. Filip Orság Ph.D., orsag@fit.vutbr.cz)

Oblíbená otázka – K čemu nám to bude?

- **Cílem tohoto předmětu je pochopit základy vykonávání programu na nejnižší úrovni a naučit se základní instrukce Intel x86.**
- Tyto základy můžete využít v oblastech:

  Optimalizace

  Tvorba překladačů

  Mikroprocesory

- A samozřejmě při zkoušce ☺

Od babylonských číslic po arabská

- **Bybylonské číslice, egyptské číslice, římské číslice**.....

...dneska nepoužijeme, špatně se s nimi pracuje. :)

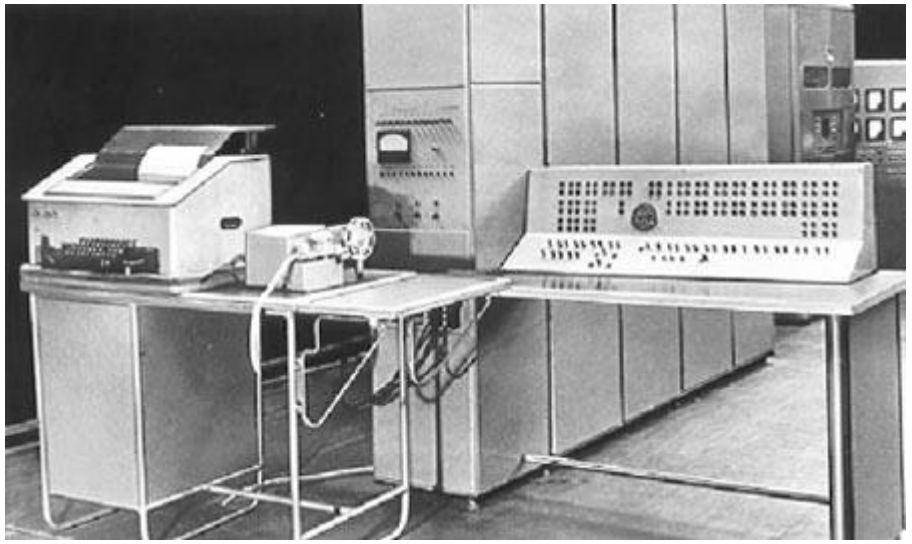
- **Polyadické soustavy**

Každé číslo můžeme rozložit na součet součinů jednotlivých číslic a mocninou jejích základů.

$$x = a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + \cdots + a_1 z^1 + a_0 z^0 + a_{-1}z^{-1} + a_{-2}z^{-2} \cdots = (a_{n-1}a_{n-2}\ldots a_1 a_0, a_{-1}a_{-2}\ldots)_z$$

Nejpoužívanější číselné soustavy

- **Dvojková soustava**
  - Základ 2
  - Pouze 0 a 1, stavy vypnuto a zapnuto
- **Desítková soustava**
  - Základ 10
  - První soustava s kterou jsme se setkali
  - Každodenně ji používáme
- **Osmičková soustava**
  - Základ 8
  - Číslice 0-7
- **Šestnáctková soustava**
  - 0…9…A….F
- **Trojková soustava**
  - Ruský počítač Setuň



Dvojková soustava v počítačích

- Bit – binary digit
- Pevná, pohyblivá řadová čárka (m=0)
- Omezená paměť → omezení pro každé číslo
- Často používané rozsahy:

4 bity	–	Nibble
8 bitů	**–**	**Byte**
16 bitů	**–**	**Word**
32 bitů	–	Double Word
64 bitů	–	Quad Word
128 bitů	–	Double Quad Word

Dvojková soustava v počítačích

- **LSB (Least Significant Bit)** – Bit s nejnižším významem. Určuje, zda je číslo liché či sudé.
- **MSB (Most Significant Bit)** – Bit s nejvyšším významem. V případě, že je použité znaménkové číslo v přímém kódu, určuje znaménko

MSB **LSB**
1 0 1 1 0 1 1 **1**

Dvojková soustava v počítačích

Příklad 1: Převod čísla 238

Iterace			
0	238	238 mod 2	=0
1	119	119 mod 2	=1
2	59	59 mod 2	=1
3	29	29 mod 2	=1
4	14	14 mod 2	=0
5	7	7 mod 2	=1
6	3	3 mod 2	=1
7	1	1 mod 2	=1
8	0		

1110 1110

Příklad 2: Převod čísla 10011101

$$x = 1_7 * 2^7 + 0_6 * 2^6 + 0_5 * 2^5 + 1_4 * 2^4 + 1_3 * 2^3 + 1_2 * 2^2 + 0_1 * 2^1 + 1_0 * 2^0 = 157$$

Reprezentace záporných čísel

- **Přímý kód**

  Znaménkem je nejvíce významný bit (MSB)

  Nevýhody: Dva odlišné algoritmy pro sčítání a odčítání
  Obsahuje dvě nuly (kladnou a zápornou)

- **Doplňkový kód**

  Vytvoříme tak, že provedeme inverzi všech bitů a přičteme 1 (k LSB)

  Obsahuje pouze jednu nulu

  Využiti pro zobrazení celých čísel (Pentium)

  Nevýhody: Nesymetrické intervaly (-128 až 127)

Reprezentace záporných čísel

- **Kód transformované nuly (aditivní kód)**

V této reprezentaci záporných čísel je posunutá nula na jinou hodnotu
Např.: 0=128 potom **+1**->129, **-1**->117
Jedna nula

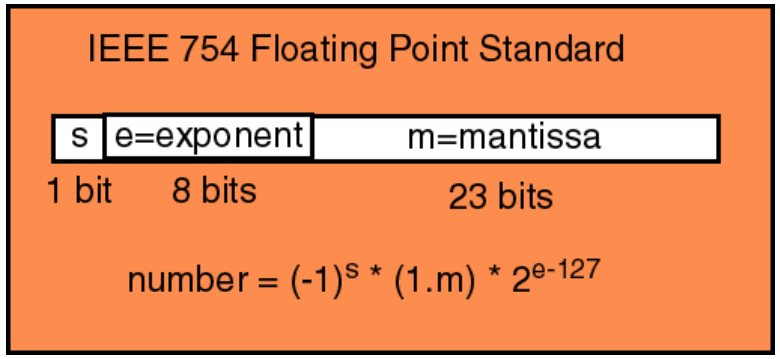
Převod (pro 8 bitová čísla)

- Rozšíříme hodnotu na 8 bitů
- Vezmeme obraz nuly, např.: 0111 1111
- Sečteme hodnotu a obraz nuly -> výsledek bude v doplňkovém kódu
- Pokud chceme změnit znaménko, změníme MSB bit

Zjednodušeně: Oproti doplňkovému kódu bude platit: záporná čísla MSB=0, kladná čísla MSB=1

Reprezentace hodnot s pohyblivou desetinnou čárkou

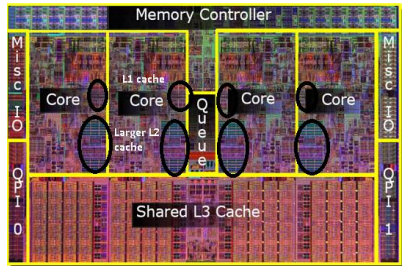
- Používá se u datových typů **float** a **double**
- Příklad.: převod **15,25** (**datový typ float**):
  - Uděláme převod z desítkové soustavy do dvojkové 15,25->1111,01
  - Binární číslo zarovnáme do tvaru 1,xxxx tzn.: **1,11101**× 2^3
  - Jedná se o kladné číslo, proto **s=0**
  - Výpočet exponentu e=127+3, e=130->**1000 0010**
  - Mantissa je shodný s desetinou částí po zarovnání. **111 0100 0000 0000 0000 0000**
  - Výsledek: **0 1000 0010 111 0100 0000 0000 0000 0000**



Zdroj: http://www.c-jump.com/bcc/common/Talk2/Cxx/IEEE_754_fp_standard/IEEE_754_fp_standard.html

Úvod do programování – typy pamětí

Operační paměť



Jedná se o volatilní paměť určenou pro dočasné uložení dat a kódu vykonávaného programu.

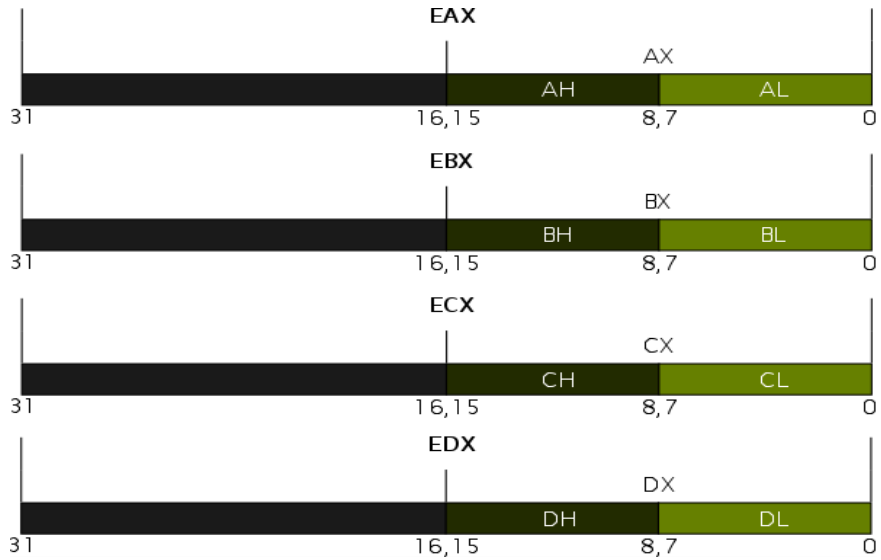
Nevýhody: Doba přístupu
Výhody: Velikost (GB-TB)

Cache

Jedná se o mezipaměť mezi procesorem a pamětí RAM, která slouží pro ukládání dat s kterými procesor aktuálně pracuje nebo v nejbližší době pracovat bude. V CPU můžeme nalézt více úrovni této paměti.

Nevýhody: Velikost
Výhody: Zmenšení latence

Úvod do programování – typy pamětí



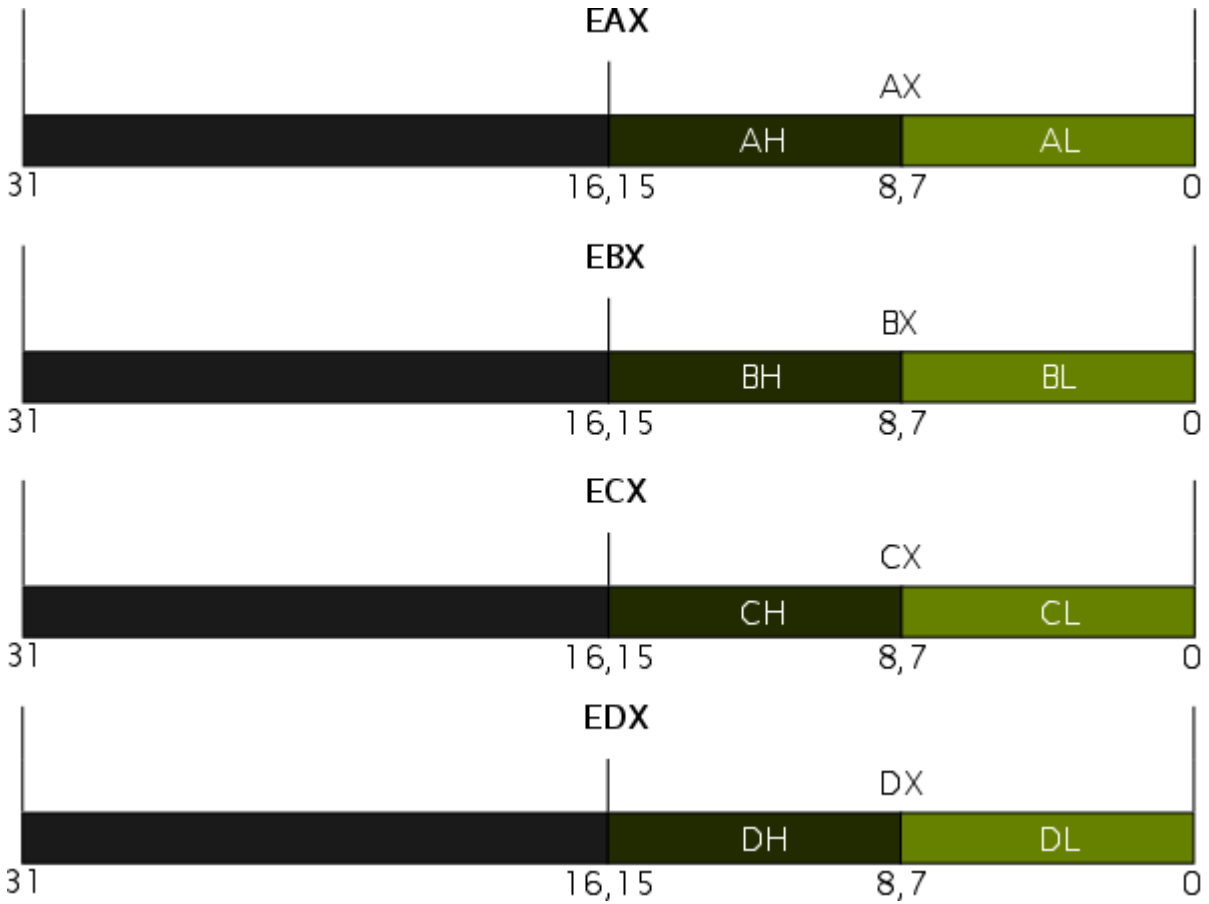
Zdroj: http://superuser.com/questions/196143/where-exactly-l1-l2-and-l3-caches-located-in-computer

Registry

Jedná se o paměťové místa, které bezprostředně využívá CPU při vykonávání programu.

Nevýhody: Omezené množství
Výhody: Minimální latence

Registry architektury x86



První program v jazyku symbolických instrukcí

- Instrukce **MOV** (zjednodušeně)

  MOV operand1, operand2

  Přesune hodnotu z operandu2 na místo určené operandem1

- Instrukce **ADD** (zjednodušeně)

  ADD operand1, operand2

  operand1= operand1+ operand2

```
bits 32
section .data


section .text
start:
        mov eax, 0
        mov al, 15
        add al, -30
ret
```

Úkoly

1. Převeďte do dvojkové a šestnáctkové soustavy: -5-<číslo_pocitace>

2. Napište program, který vynuluje registrxy EAX, EBX, ECX, EDX, do registru al zapište -5, do registru bl -<číslo_pocitace> a ověřte pomocí instrukce add, zda hodnota vypočítaná v 1) je správná (použijte debug výpis hodnot v registru)