

# Základy počítače

Jáchym Löwenhoffer

Gynekologická Evaluace Velkých Obrazů

*jachym.lowenhoffer@gmail.com*

19. září 2024

- 1 Co je v počítači
- 2 FDE cycle
- 3 Instrukce
- 4 Logické obvody
- 5 Paměť
- 6 Motherboard

# Co je v počítači

- **Paměť** je na sobě naházená jako na hromadě, ale každý byte má svojí adresu.
- **Registr** je chlívěček, kam schovám byte a můžu s ním pracovat (porovnávat, sčítat atd.).
- **Address Bus**: tudy se posílají adresy.
- **Data Bus**: tudy se posílají data.
- **PC**(program counter): je speciální registr kam se ukládá adresa další instrukce na vykonání.
- **MBR**(master boot record): je část paměti kam se uloží instrukce na vykonání předtím než se posune do místa kde se opravdu může vykonat.

# FDE cycle

# Fakt Drsná Elipsa



# Program? jako co budou večer dávat?

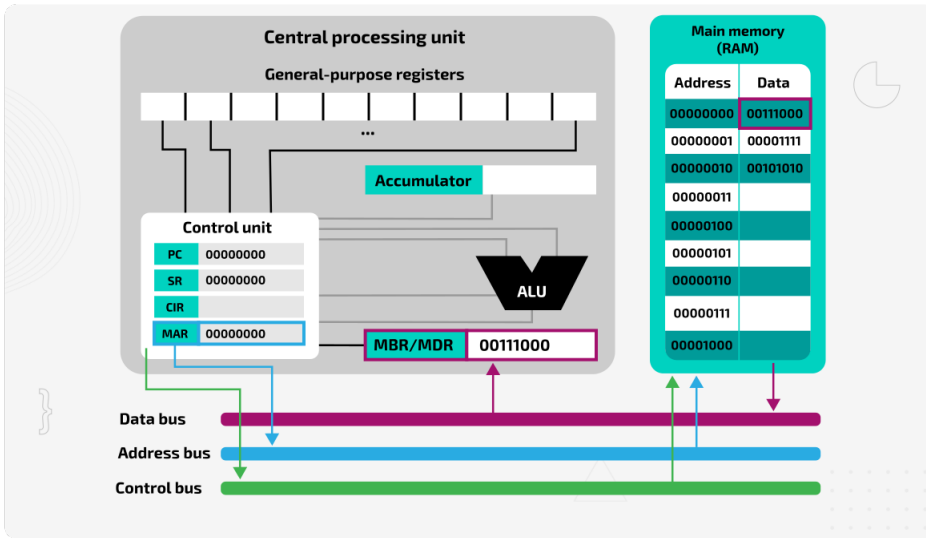
## Počítačový program

je posloupnost instrukcí, která má nějaký cíl. Může mít nějaký vstup, ale výstup mít musí!

Jestliže budeme mít program který hledá třetí mocninu čísel tak mu ale musíme dát nějaké číslo aby nám mohl něco vrátit. Toto číslo je tedy vstup. Výstupem je výsledek nebo error.

- CPU se podívá na svůj PC (program counter), tam je uložena adresa další instrukce.
- Address busem pošle pro tuto instrukci do RAMky (tam už musí být předem celý program načtený)
- Data busem se vrátí hodnota této instrukce.
- Zapiše se do MBR a z něj se zkopíruje do CIR (current instruction register).
- CIR je registr, kde je CPU schopná instrukci vykonat a tím začít vykonávat další registry.





- Jakmile je instrukce v CIR (, CPU začne hledat, o jakou instrukci jde.
- To udělá tak, že ji rozdělí na části.
- Podle typu instrukce vyvěsí *vlajky*<sup>1</sup> nebo připraví potřebné registry.
- Taky se hodnota v PC zvýší o jedna, aby se tím program posunul na další instrukci.

---

<sup>1</sup>Programu říkají, že se děje něco zajímavého a on na ně může reagovat.

- Jakmile víme o jakou instrukci se jedná, můžeme ji vyhodnotit!
- Jaké instrukce existují?

# Instrukce

`jump` přepíše hodnotu v PC (adresa další instrukce na vykonání) na nějakou jinou. Díky tomu můžeme mít věci jako *cykly*.

`jump-if` to samé jako `jump`, jen předtím ověří nějakou podmínku. Porovná hodnoty dvou registrů nebo ověří nějaké *vlajky*. Tedy se tato instrukce nápadně podobá *podmínkám* v programování.

Oba skoky můžou být na jakémkoliv místo v paměti a náš program tak vůbec nemusí být pohromadě tak jak jsme ho napsali. Což vůbec nevadí a jediné co nás zajímá je aby byl proveden v takovém pořadí.

`allocate` vyhradí pro náš program kus prostoru v paměti a všechny ostatní programy vědí, že tuto část nemají používat. Jakmile už ji nepotřebujeme, je třeba tuto paměť uvolnit.

`load` načte určitý úsek z paměti do registrů.

`store` hodnoty z registrů (třeba výsledek výpočtu) uloží do paměti.

`add` sečte hodnotu ze dvou registrů a uloží ji do třetího.

`compare` porovná hodnoty ve dvou registrech a výsledek uloží do třetího. Výsledek je vždy `true` nebo `false`.

# Logické obvody



Kdybychom měli jen dráty uchováající jedničky a nuly, tak je nuda. Potřebujeme ještě způsob, jak mezi sebou tyto signály kombinovat, potřebujeme na nich nějakou logiku.

Proto máme logické brány. Pomocí těchto bran jsme schopni postavit celé moderní počítače.

Tyto brány fungují jako "operace" mezi jednotlivými dráty.

**AND** funguje stejně jako v jazyce. Výrok: "Venku je zima a prší." je pravdivý, jen když venku opravdu prší a opravdu je zima.




**OR** také se dá najít v jazyce. Funguje jako "nebo" ve slučovacím poměru.

**NOT** otáčí původní hodnotu signálu. Jestliže je vstup 0, tak vrátí 1, a naopak.

Než nám to nadcházející obrázek prozradí zamysleme se, nad tím jak bychom logické brány reprezentovali jako aritmetické operace v binární soustavě.

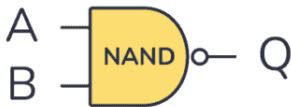
# Tabulky pravdivostních hodnot

Table 4.9 Logical gates and truth table

| Logical gate  | Truth table  |       |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
|---|--|-------|-------------------|-------|---|---|---|---|---|---|---|---|---|---|---|---|
| <p>AND gate</p>  | <table><tr><th>A</th><th>B</th><th>y=AB</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>  | A     | B                 | y=AB  | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| A   | B  | y=AB  |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 0  | 0     |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 1  | 0     |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 0  | 0     |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 1  | 1     |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| <p>OR gate</p>   | <table><tr><th>A</th><th>B</th><th>y=A+B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> | A     | B                 | y=A+B | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| A   | B  | y=A+B |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 0  | 0     |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 1  | 1     |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 0  | 1     |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 1  | 1     |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| <p>NOT gate</p>  | <table><tr><th>A</th><th>y=<math>\overline{A}</math></th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>   | A     | y= $\overline{A}$ | 0     | 1 | 1 | 0 |   |   |   |   |   |   |   |   |   |
| A   | y= $\overline{A}$  |       |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 1  |       |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 0  |       |                   |       |   |   |   |   |   |   |   |   |   |   |   |   |

# NA(n)De vše!

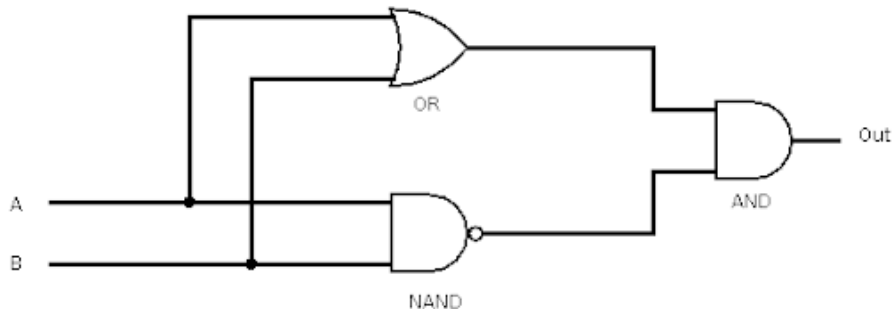
Logických brán je samozřejmě o hodně víc než jen tyto tři. Kdybyste si někdy chtěli stavět vlastní procesor hodí se vědět, že jich tolik vlastně nepotřebujete. Úplně vám stačí brána AND a NOT. Ty když dáme hned za sebe (ve stejném pořadí) vznikne nám NAND.



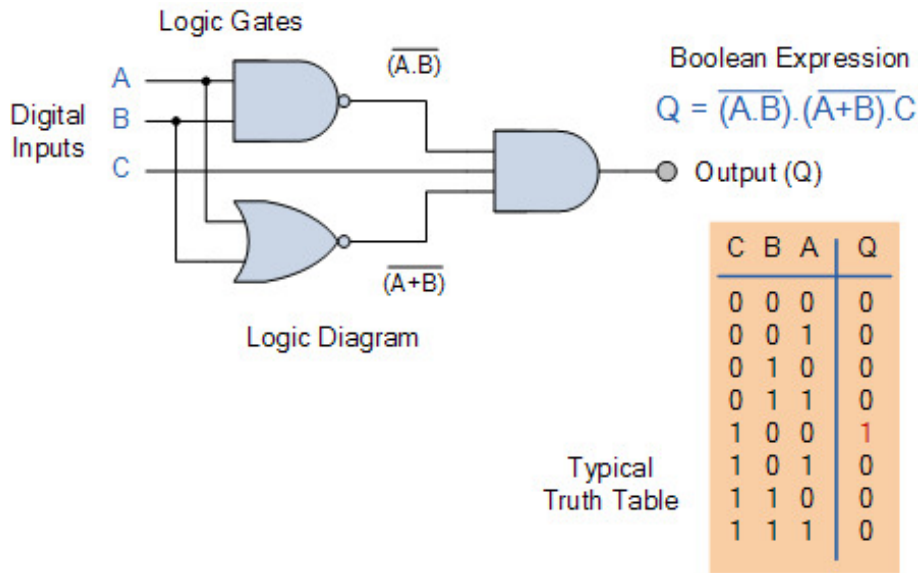
| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Výlučné OR

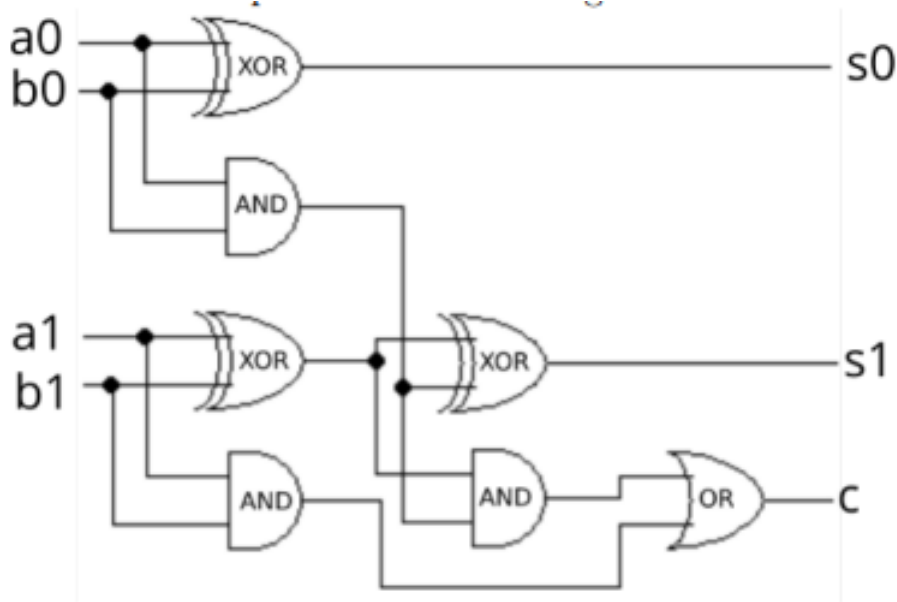
Aby toho nebylo málo ukážeme si ještě jako udělat výlučné OR, tedy XOR. Jedná se o OR jen s tou výjimkou, že když jsou oba vstupy pravda výstupem je nepravda. Ptáme se tedy na otázku: „Je **právě jeden** vstup pravda?“ namísto „Je **alespoň** jeden vstup pravda?“ jak je tomu u OR.



# Jednoduchý logický obvod



# Konečně sčítáme!



# Paměť



## Volatilita

je nutnost paměti být připojena na stálé napětí. Jakmile ho jednou odpojíme, ztrácíme všechna data.

U té nevolatilní ztrácíme jen možnost z ní číst a do ní psát, což je přirozené.

# Proč RAM?

RAMka je nejrozšířenější druh volatilní paměti. Je to takový mezistupeň mezi vnější pamětí (pevný disk nebo CD, jestli ještě někdo ví, co to je) a procesorem. Těchto mezistupňů je ještě víc, ale RAM je hlavní a největší.

Mezistupeň v tom smyslu, že jakýkoliv program uložený na CD musíme nejdřív načíst na RAMku a až potom ho můžeme spustit.

Podle velikosti RAM se odvíjí, jak náročné programy dokážeme spustit a s kolika daty naráz jsme schopni pracovat.

Jedná se tedy o jednu dlouhou pásku bytu s adresami ze které jsme, známe-li adresu, schopni přečíst jakoukoliv hodnotu velmi rychle. U externích pamětí to je často závislé na tom v jakém místě se náš byte nachází.

RAMka má také samozřejmě víc typů.

## Static RAM

Data jsou v ní stále a nikam neutíkají. Používá se na vnitřní paměť CPU.

## Dynamic RAM

Data stále někam utíkají a tak je třeba vrátit je na místo nějakým tím pravidelným šokem. Tato paměť je výrazně složitější na to udržet pohromadě, ale za to je efektivnější.

## Non Volatile RAM

Je typ RAMky, která není volatilní. Tedy i když do ní nejde proud data zachovává. Toho docílila svojí externí baterkou, která ovšem není nekonečná. Proto se používá jen velmi limitovaně (úložiště BIOSu).

## Read Only Memory

Z této paměti, jak napovídá její název můžeme pouze číst. Znamená to tedy, že adresová linka může fungovat jen směrem od ROMky do CPU.

# Motherboard

K čemu je?

- Díky ní si povídají ostatní komponenty (má na sobě již zmíněné autobusové linky).
- Rozděluje napětí do jednotlivých komponent.
- Sama obsahuje některé komponenty, ale hlavní funkce je spojovat externí komponenty. Jako například grafické karty, myši, klávesnice atd.
- Nad všemi těmito komponentami má nadvládu CPU a těm dává instrukce.

**G**raphical **P**rocessing **U**nit je uzpůsobená k tomu udělat spoustu jednoduchých operací. To se děje když vykreslujeme obrázky (proto **graphical**) nebo když těžíme BitCoin.