



# VON NEUMANNOVA ARCHITEKTURA

Adam Klepáč

20. listopadu 2023

Konečný automat a datová cesta

von Neumannův model

Architektura instrukční sady

# KONEČNÝ AUTOMAT A DATOVÁ CESTA

# KONEČNÝ AUTOMAT A DATOVÁ CESTA

**Konečný automat** (angl. FSM – finite-state machine) je výpočetní model v teoretické informatice.

# KONEČNÝ AUTOMAT A DATOVÁ CESTA

**Konečný automat** (angl. FSM – finite-state machine) je výpočetní model v teoretické informatice. Je to popis velmi jednoduchého počítače, který může být v jednom z několika stavů, mezi kterými přechází na základě informací ze vstupu.

# KONEČNÝ AUTOMAT A DATOVÁ CESTA

**Konečný automat** (angl. FSM – finite-state machine) je výpočetní model v teoretické informatice. Je to popis velmi jednoduchého počítače, který může být v jednom z několika stavů, mezi kterými přechází na základě informací ze vstupu.

**Datová cesta** (angl. Datapath) je sdružení jednotek provádějících výpočetní operace, registrů a řadičů.

## PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .



## PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .

**V pseudokódu:**

$a = 1$ ;

$b = n$ ;

**dokud  $b \neq 0$  opakuj**

$a = a \cdot b$ ;

$b = b - 1$ ;

**konec**



# PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .

**V pseudokódu:**

$a = 1;$

$b = n;$

**dokud  $b \neq 0$  opakuj**

$a = a \cdot b;$

$b = b - 1;$

**konec**

**Průběh pro  $n = 5$ :**

na začátku:  $a = 1$        $b = 5$

# PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .

**V pseudokódu:**

$a = 1;$

$b = n;$

**dokud  $b \neq 0$  opakuj**

$a = a \cdot b;$

$b = b - 1;$

**konec**

**Průběh pro  $n = 5$ :**

na začátku:	$a = 1$	$b = 5$
-------------	---------	---------

---

po 1. iteraci:	$a = 5$	$b = 4$
----------------	---------	---------

# PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .

**V pseudokódu:**

$a = 1;$

$b = n;$

**dokud  $b \neq 0$  opakuj**

$a = a \cdot b;$

$b = b - 1;$

**konec**

**Průběh pro  $n = 5$ :**

na začátku:	$a = 1$	$b = 5$
-------------	---------	---------

---

po 1. iteraci:	$a = 5$	$b = 4$
----------------	---------	---------

po 2. iteraci:	$a = 20$	$b = 3$
----------------	----------	---------

# PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .

**V pseudokódu:**

```

a = 1;
b = n;
dokud b ≠ 0 opakuj
    |   a = a · b;
    |   b = b - 1;
konec
    
```

**Průběh pro  $n = 5$ :**

na začátku:	$a = 1$	$b = 5$
po 1. iteraci:	$a = 5$	$b = 4$
po 2. iteraci:	$a = 20$	$b = 3$
po 3. iteraci:	$a = 60$	$b = 2$

# PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .

**V pseudokódu:**

```

a = 1;
b = n;
dokud b ≠ 0 opakuji
    |   a = a · b;
    |   b = b - 1;
konec
    
```

**Průběh pro  $n = 5$ :**

na začátku:	$a = 1$	$b = 5$
po 1. iteraci:	$a = 5$	$b = 4$
po 2. iteraci:	$a = 20$	$b = 3$
po 3. iteraci:	$a = 60$	$b = 2$
po 4. iteraci:	$a = 120$	$b = 1$

# PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .

**V pseudokódu:**

```

a = 1;
b = n;
dokud b ≠ 0 opakuji
    |   a = a · b;
    |   b = b - 1;
konec
    
```

**Průběh pro  $n = 5$ :**

na začátku:	$a = 1$	$b = 5$
po 1. iteraci:	$a = 5$	$b = 4$
po 2. iteraci:	$a = 20$	$b = 3$
po 3. iteraci:	$a = 60$	$b = 2$
po 4. iteraci:	$a = 120$	$b = 1$
po 5. iteraci:	$a = 120$	$b = 0$

# PŘÍKLAD – FAKTORIÁL

Postavíme FSM, který umí spočítat  $n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$ .

**V pseudokódu:**

$a = 1;$

$b = n;$

**dokud  $b \neq 0$  opakuj**

$a = a \cdot b;$

$b = b - 1;$

**konec**

**Průběh pro  $n = 5$ :**

na začátku:	$a = 1$	$b = 5$
-------------	---------	---------

---

po 1. iteraci:	$a = 5$	$b = 4$
----------------	---------	---------

po 2. iteraci:	$a = 20$	$b = 3$
----------------	----------	---------

po 3. iteraci:	$a = 60$	$b = 2$
----------------	----------	---------

po 4. iteraci:	$a = 120$	$b = 1$
----------------	-----------	---------

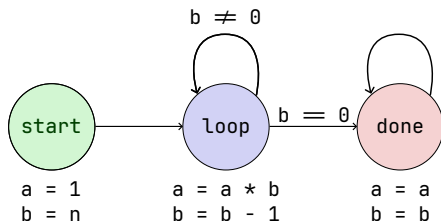
po 5. iteraci	$a = 120$	$b = 0$
---------------	-----------	---------

---

konec

# PŘÍKLAD – FAKTORIÁL

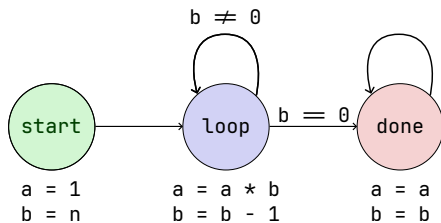
FSM vykonávající tento algoritmus:





## PŘÍKLAD – FAKTORIÁL

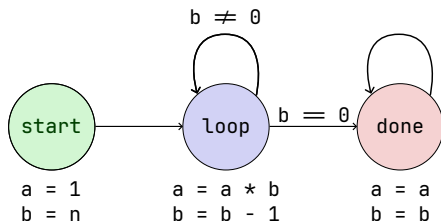
FSM vykonávající tento algoritmus:



Pro převod do hardwarové podoby potřebujeme:

## PŘÍKLAD – FAKTORIÁL

FSM vykonávající tento algoritmus:

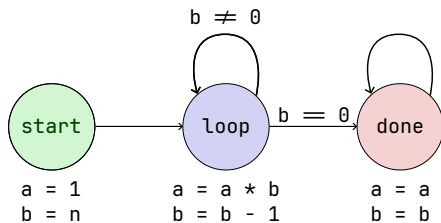


Pro převod do hardwarové podoby potřebujeme:

- dva registry – pro  $a$  a pro  $b$ ,

# PŘÍKLAD – FAKTORIÁL

FSM vykonávající tento algoritmus:

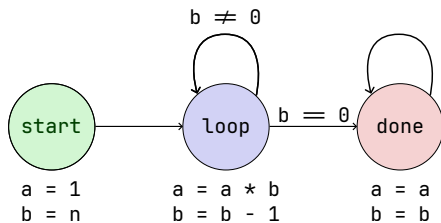


Pro převod do hardwarové podoby potřebujeme:

- dva registry – pro  $a$  a pro  $b$ ,
- dva bity pro uložení stavu – start/loop/done,

# PŘÍKLAD – FAKTORIÁL

FSM vykonávající tento algoritmus:

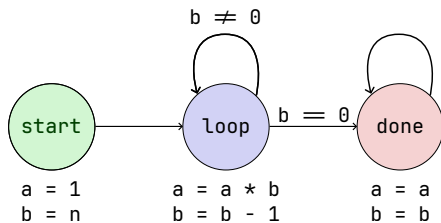


Pro převod do hardwarové podoby potřebujeme:

- dva registry – pro  $a$  a pro  $b$ ,
- dva bity pro uložení stavu – start/loop/done,
- logické přechody –  $b == 0$  a  $b \neq 0$ ,

# PŘÍKLAD – FAKTORIÁL

FSM vykonávající tento algoritmus:

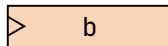
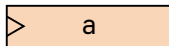


Pro převod do hardwarové podoby potřebujeme:

- dva registry – pro  $a$  a pro  $b$ ,
- dva bity pro uložení stavu – start/loop/done,
- logické přechody –  $b == 0$  a  $b \neq 0$ ,
- přiřazení hodnoty registrům –  $a = a \cdot b$  a  $b = b - 1$ .

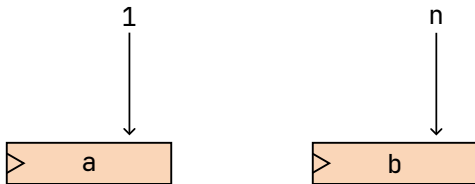
# DATOVÁ CESTA PRO FAKTORIÁL

Nejprve přidáme registry pro  $a$  a  $b$ .



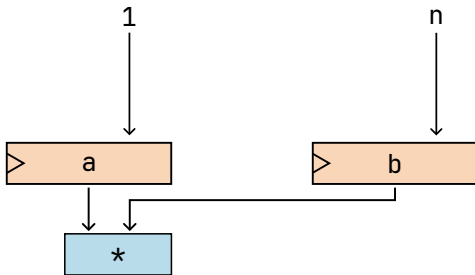
# DATOVÁ CESTA PRO FAKTORIÁL

Na začátku musejí registry obdržet hodnoty 1 a  $n$ .



# DATOVÁ CESTA PRO FAKTORIÁL

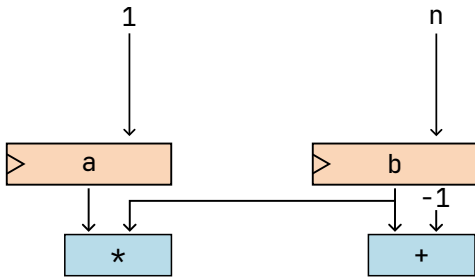
Přidáme logický obvod pro násobení  $a \cdot b$ .





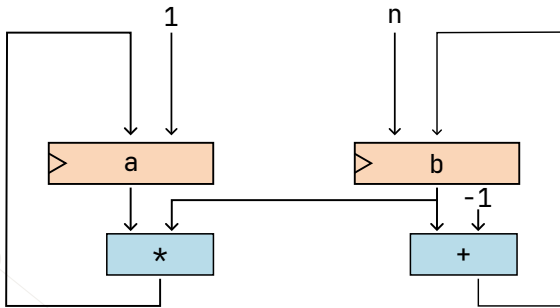
# DATOVÁ CESTA PRO FAKTORIÁL

Přidáme logický obvod pro součet  $b + (-1)$ .



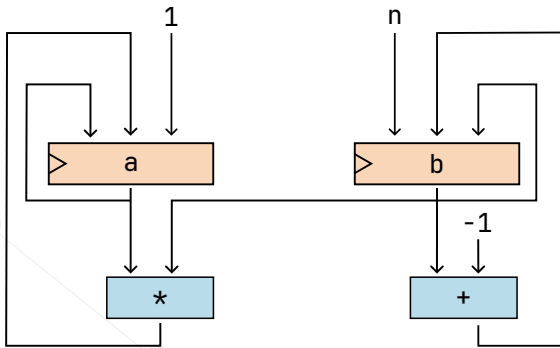
# DATOVÁ CESTA PRO FAKTORIÁL

Přidáme cestu zpět z výpočetních obvodů do registrů pro  $a$  a  $b$ .



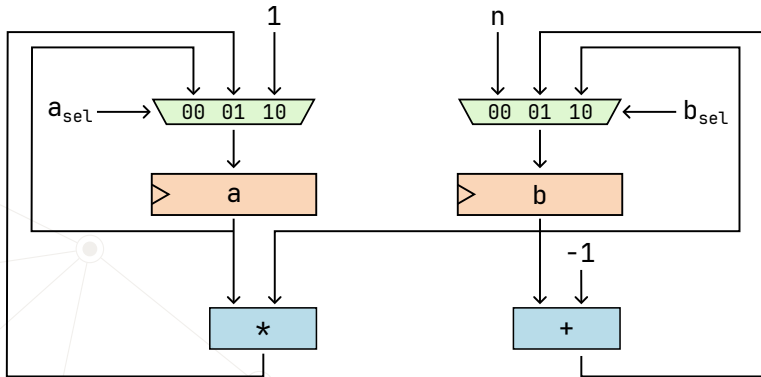
# DATOVÁ CESTA PRO FAKTORIÁL

Přidáme cestu z registrů pro  $a$  a  $b$  zpět do nich samých, aby po skončení výpočtu mohly uchovat informaci.



# DATOVÁ CESTA PRO FAKTORIÁL

Přidáme přepínače stavů, aby vždy do registrů pouštěly jen jeden vstup podle stavu výpočtu.



# ŘÍDÍCÍ FSM PRO FAKTORIÁL

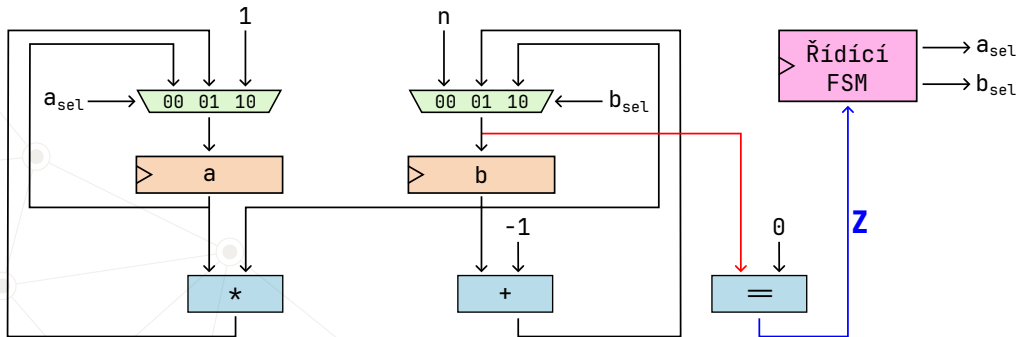


Přepínání vstupů do registrů pro  $a$  a  $b$  musí řídit nějaká jednotka – konečný automat.



# ŘÍDÍCÍ FSM PRO FAKTORIÁL

Přepínání vstupů do registrů pro  $a$  a  $b$  musí řídit nějaká jednotka – konečný automat. Ta musí přepínat  $a_{sel}$  a  $b_{sel}$  během výpočtu.



# VON NEUMANNŮV MODEL

# VON NEUMANNŮV MODEL

Nejpoužívanější model víceúčelového počítače. Skoro všechny moderní počítače jsou postaveny na tomto principu.

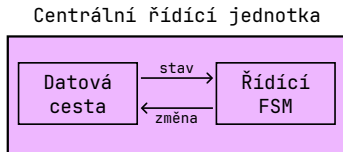




# VON NEUMANNŮV MODEL

Nejpoužívanější model víceúčelového počítače. Skoro všechny moderní počítače jsou postaveny na tomto principu.

Komponenty:



- **CPU:** Provádí aritmetické a logické operace na datech uložených v paměti a registrech.

# VON NEUMANNŮV MODEL

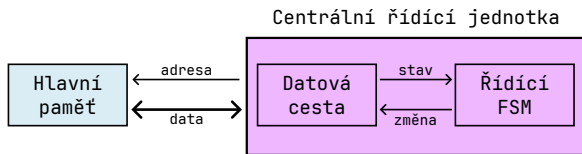
Nejpoužívanější model víceúčelového počítače. Skoro všechny moderní počítače jsou postaveny na tomto principu.



# VON NEUMANNŮV MODEL

Nejpoužívanější model víceúčelového počítače. Skoro všechny moderní počítače jsou postaveny na tomto principu.

Komponenty:



- **CPU:** Provádí aritmetické a logické operace na datech uložených v paměti a registrech.
- **Hlavní (dnes „vnitřní“) paměť:** Seznam **slov** o daném počtu **bitů**

# VON NEUMANNŮV MODEL

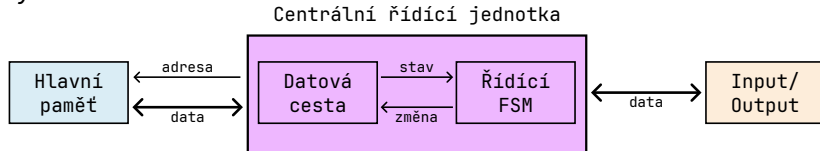
Nejpoužívanější model víceúčelového počítače. Skoro všechny moderní počítače jsou postaveny na tomto principu.



# VON NEUMANNŮV MODEL

Nejpoužívanější model víceúčelového počítače. Skoro všechny moderní počítače jsou postaveny na tomto principu.

Komponenty:



- **CPU**: Provádí aritmetické a logické operace na datech uložených v paměti a registrech.
- **Hlavní (dnes „vnitřní“) paměť**: Seznam **slov** o daném počtu **bitů**.
- **Input/output**: Libovolná zařízení umožňující interakci s vnějším světem.

# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



- Vyjádření programu jako posloupnosti zakódovaných instrukcí.

# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



- Vyjádření programu jako posloupnosti zakódovaných instrukcí.
- Paměť ukládá **jak program, tak instrukce!**

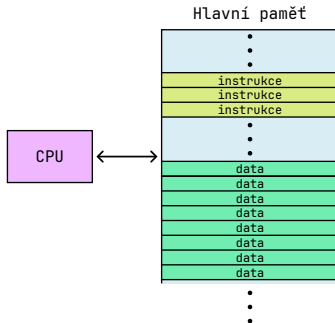
# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



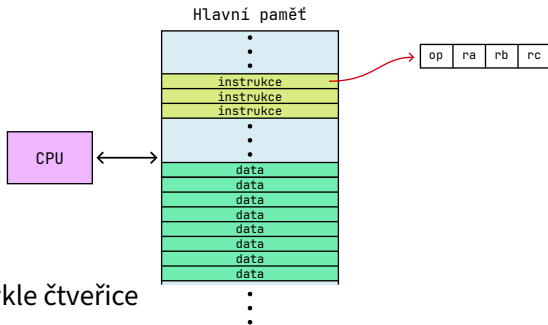
- Vyjádření programu jako posloupnosti zakódovaných instrukcí.
- Paměť ukládá **jak program, tak instrukce!**
- CPU **vyzvedne, přeloží a vykoná** (fetch-decode-execute cyklus) instrukce programu, jak jdou za sebou.



# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI

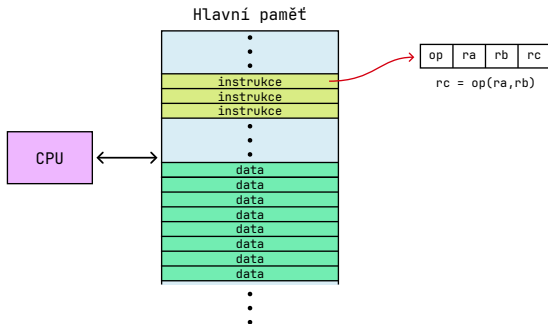


# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



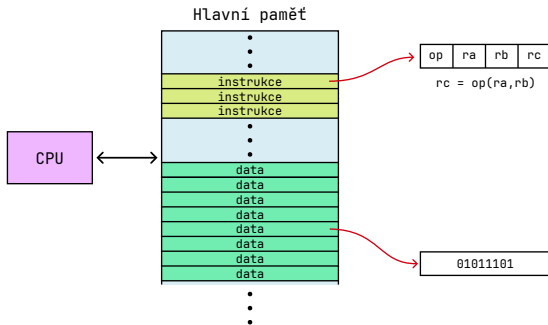
- **Instrukce** je obvykle čtveřice
  - operace (op),
  - registr s prvním operandem (ra),
  - registr s druhým operandem (rb),
  - registr pro uložení výsledku (rc).

# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



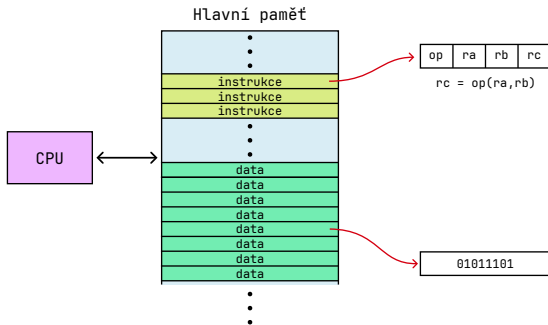
- CPU tuhle instrukci přeloží jako, „Proved' op na hodnotu v ra a hodnotu v rb a výsledek ulož do rc“.

# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



- Bloky (řádky) paměti s daty nemají žádný daný formát. Jsou to zkrátka jen uložené hodnoty vzniklé nějakým výpočtem v minulosti. Mohou to být slova, čísla či cokoli jiného.

# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



- Bloky (řádky) paměti s daty nemají žádný daný formát. Jsou to zkrátka jen uložené hodnoty vzniklé nějakým výpočtem v minulosti. Mohou to být slova, čísla či cokoli jiného.
- Každý blok má daný počet bitů (v tomto příkladě 8).

# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



**Problém:** Jak poznám, co je instrukce a co datum?



# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



**Problém:** Jak poznám, co je instrukce a co datum?

- Pamatujte, že CPU i paměť jsou pořád jenom logické (elektrické) obvody.

# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI



**Problém:** Jak poznám, co je instrukce a co datum?

- Pamatujte, že CPU i paměť jsou pořád jenom logické (elektrické) obvody.
- To znamená, že je možné ukládat instrukce tak, aby šly přímo do **řídící FSM** a data naopak přímo do **datové cesty (a tam do registrů)**.

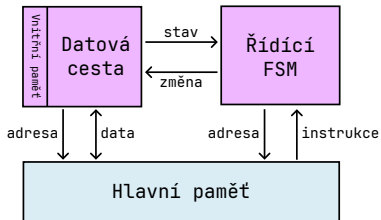


# KLÍČOVÁ IDEA: PROGRAM ULOŽENÝ V HLAVNÍ PAMĚTI

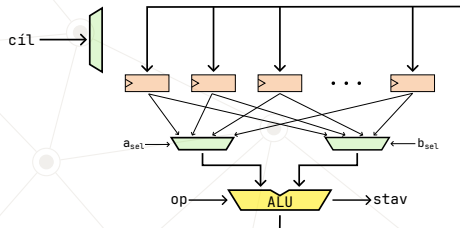
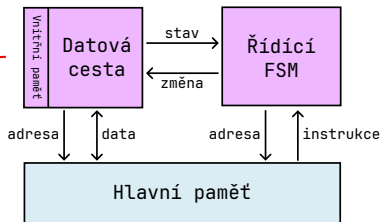
**Problém:** Jak poznám, co je instrukce a co datum?

- Pamatujte, že CPU i paměť jsou pořád jenom logické (elektrické) obvody.
- To znamená, že je možné ukládat instrukce tak, aby šly přímo do **řídící FSM** a data naopak přímo do **datové cesty (a tam do registrů)**.
- CPU je pak navržen tak, že posloupnost bitů v řídící FSM považuje za instrukci, zatímco v datové cestě za datum.

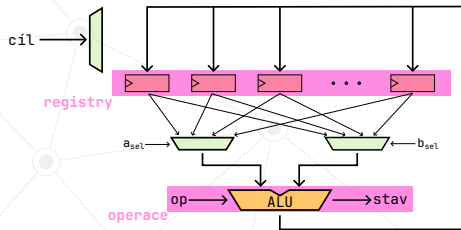
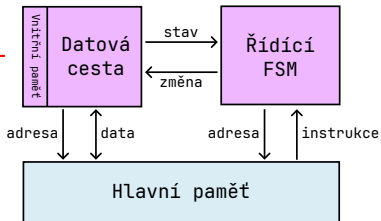
# ANATOMIE VON NEUMANNOVA POČÍTAČE



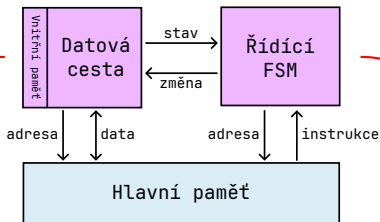
# ANATOMIE VON NEUMANNOVA POČÍTAČE



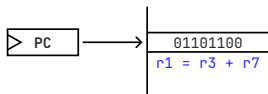
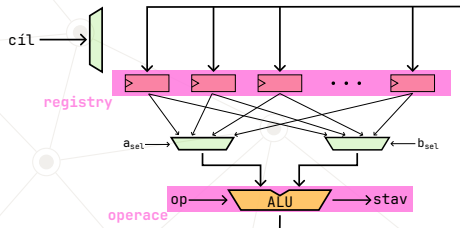
# ANATOMIE VON NEUMANNOVA POČÍTAČE



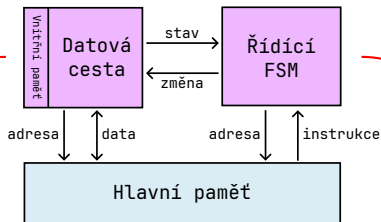
# ANATOMIE VON NEUMANNOVA POČÍTAČE



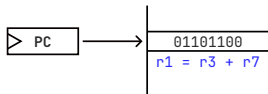
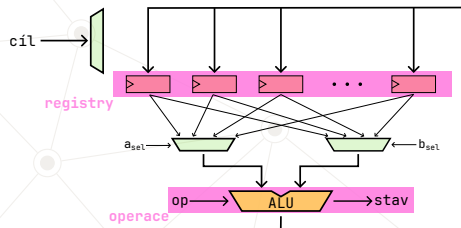
- Instrukce** zakódovány jako binární čísla.



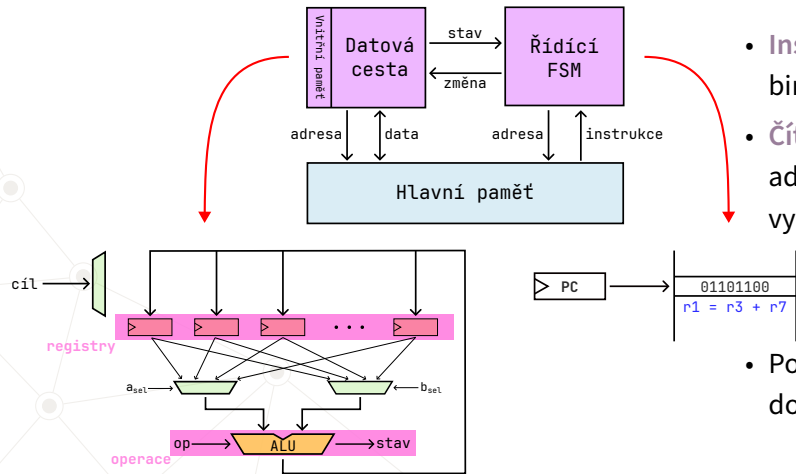
# ANATOMIE VON NEUMANNOVA POČÍTAČE



- **Instrukce** zakódovány jako binární čísla.
- **Čítač** (PC – Program Counter): adresa instrukce, která se má vykonat.



# ANATOMIE VON NEUMANNOVA POČÍTAČE



- **Instrukce** zakódovány jako binární čísla.
- **Čítač** (PC – Program Counter): adresa instrukce, která se má vykonat.
- Postup pro překlad instrukcí do signálů pro datovou cestu.

# ANATOMIE VON NEUMANNOVA POČÍTAČE – INSTRUKCE



- Instrukce jsou základní jednotkou práce.





# ANATOMIE VON NEUMANNOVA POČÍTAČE – INSTRUKCE



- Instrukce jsou základní jednotkou práce.
- Každá instrukce specifikuje
  - **operaci**, která se má vykonat,



# ANATOMIE VON NEUMANNOVA POČÍTAČE – INSTRUKCE

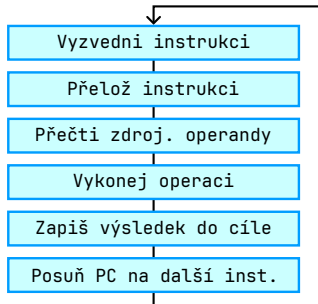


- Instrukce jsou základní jednotkou práce.
- Každá instrukce specifikuje
  - **operaci**, která se má vykonat,
  - registry se zdrojovými **operandy** a **cílový registr** pro výsledek.
- Ve von Neumannově počítači jsou instrukce čteny postupně.

# ANATOMIE VON NEUMANNOVA POČÍTAČE – INSTRUKCE



- Instrukce jsou základní jednotkou práce.
- Každá instrukce specifikuje
  - **operaci**, která se má vykonat,
  - registry se zdrojovými **operandy** a **cílový registr** pro výsledek.
- Ve von Neumannově počítači jsou instrukce čteny postupně.
  - CPU implementuje cyklus napravo:
  - Pokud právě vykonaná instrukce nspecifikuje jinak, je adresa další instrukce v pořadí adresa této + délka instrukce (v bitech).



# ARCHITEKTURA INSTRUKČNÍ SADY

# ARCHITEKTURA INSTRUKČNÍ SADY

Architektura instrukční sady (angl. **ISA** – Instruction Set Architecture) je jakási „dohoda“ mezi softwarem a hardwarem.



# ARCHITEKTURA INSTRUKČNÍ SADY

Architektura instrukční sady (angl. **ISA** – Instruction Set Architecture) je jakási „dohoda“ mezi softwarem a hardwarem.

- **Software** jsou programy uložené ve vnitřní paměti – dají se měnit.

# ARCHITEKTURA INSTRUKČNÍ SADY

Architektura instrukční sady (angl. **ISA** – Instruction Set Architecture) je jakási „dohoda“ mezi softwarem a hardwarem.

- **Software** jsou programy uložené ve vnitřní paměti – dají se měnit.
- **Hardware** jsou logické obvody tvořící počítač – nelze je měnit.

# ARCHITEKTURA INSTRUKČNÍ SADY

Architektura instrukční sady (angl. **ISA** – Instruction Set Architecture) je jakási „dohoda“ mezi softwarem a hardwarem.

- **Software** jsou programy uložené ve vnitřní paměti – dají se měnit.
- **Hardware** jsou logické obvody tvořící počítač – nelze je měnit.
- **ISA** obsahuje
  - definice **operací** a **adres pro ukládání**,



# ARCHITEKTURA INSTRUKČNÍ SADY

Architektura instrukční sady (angl. **ISA** – Instruction Set Architecture) je jakási „dohoda“ mezi softwarem a hardwarem.

- **Software** jsou programy uložené ve vnitřní paměti – dají se měnit.
- **Hardware** jsou logické obvody tvořící počítač – nelze je měnit.
- **ISA** obsahuje
  - definice **operací** a **adres pro ukládání**,
  - **přesný popis** toho, jak je software má **nařídít** a **dostat se k nim**.

# ARCHITEKTURA INSTRUKČNÍ SADY

ISA je další úroveň abstrakce umožňující vývoj hardware **bez ohledu na software.**



# ARCHITEKTURA INSTRUKČNÍ SADY

ISA je další úroveň abstrakce umožňující vývoj hardware **bez ohledu na software**.

- ISA specifikuje, **co** hardware poskytuje, nikoli **jak** (nespecifikuje implementaci logických obvodů v CPU).

# ARCHITEKTURA INSTRUKČNÍ SADY

ISA je další úroveň abstrakce umožňující vývoj hardware **bez ohledu na software**.

- ISA specifikuje, **co** hardware poskytuje, nikoli **jak** (nespecifikuje implementaci logických obvodů v CPU).
- Dokud je ISA dodržena, je možné vylepšovat hardware zcela bez ohledu na software, který ho používá. Např.
  - Intel 8086 (z roku 1978) má 29 tisíc tranzistorů, hodiny s frekvencí 5 MHz a zvládne asi 330 tisíc operací za sekundu;

# ARCHITEKTURA INSTRUKČNÍ SADY

ISA je další úroveň abstrakce umožňující vývoj hardware **bez ohledu na software**.

- ISA specifikuje, **co** hardware poskytuje, nikoli **jak** (nespecifikuje implementaci logických obvodů v CPU).
- Dokud je ISA dodržena, je možné vylepšovat hardware zcela bez ohledu na software, který ho používá. Např.
  - Intel 8086 (z roku 1978) má 29 tisíc tranzistorů, hodiny s frekvencí 5 MHz a zvládne asi 330 tisíc operací za sekundu;
  - Intel Pentium 4 (2003) má 44 milionů tranzistorů, hodiny s frekvencí 4 Ghz a zvládne asi 5 miliard operací za sekundu;

# ARCHITEKTURA INSTRUKČNÍ SADY

ISA je další úroveň abstrakce umožňující vývoj hardware **bez ohledu na software**.

- ISA specifikuje, **co** hardware poskytuje, nikoli **jak** (nespecifikuje implementaci logických obvodů v CPU).
- Dokud je ISA dodržena, je možné vylepšovat hardware zcela bez ohledu na software, který ho používá. Např.
  - Intel 8086 (z roku 1978) má 29 tisíc tranzistorů, hodiny s frekvencí 5 MHz a zvládne asi 330 tisíc operací za sekundu;
  - Intel Pentium 4 (2003) má 44 milionů tranzistorů, hodiny s frekvencí 4 Ghz a zvládne asi 5 miliard operací za sekundu;
 oba procesory používají instrukční sadu x86.

# ARCHITEKTURA INSTRUKČNÍ SADY

Navrhnout ISA je obtížné.

- Kolik operací a jaké?



# ARCHITEKTURA INSTRUKČNÍ SADY

Navrhnout ISA je obtížné.

- Kolik operací a jaké?
- Jaké typy úložišť? Kolik?





# ARCHITEKTURA INSTRUKČNÍ SADY

Navrhnout ISA je obtížné.

- Kolik operací a jaké?
- Jaké typy úložišť? Kolik?
- Jak kódovat instrukce?

# ARCHITEKTURA INSTRUKČNÍ SADY

Navrhnout ISA je obtížné.

- Kolik operací a jaké?
- Jaké typy úložišť? Kolik?
- Jak kódovat instrukce?
- Jak zařídit kompatibilitu s budoucím hardwarem?

# ARCHITEKTURA INSTRUKČNÍ SADY

Navrhnout ISA je obtížné.

- Kolik operací a jaké?
- Jaké typy úložišť? Kolik?
- Jak kódovat instrukce?
- Jak zařídit kompatibilitu s budoucím hardwarem?

## Statistický přístup:

- Vybere se mnoho testovacích programů.

# ARCHITEKTURA INSTRUKČNÍ SADY

Navrhnout ISA je obtížné.

- Kolik operací a jaké?
- Jaké typy úložišť? Kolik?
- Jak kódovat instrukce?
- Jak zařídit kompatibilitu s budoucím hardwarem?

## Statistický přístup:

- Vybere se mnoho testovacích programů.
- Každá verze ISA se otestuje na každém programu.

# ARCHITEKTURA INSTRUKČNÍ SADY

Navrhnout ISA je obtížné.

- Kolik operací a jaké?
- Jaké typy úložišť? Kolik?
- Jak kódovat instrukce?
- Jak zařídit kompatibilitu s budoucím hardwarem?

## Statistický přístup:

- Vybere se mnoho testovacích programů.
- Každá verze ISA se otestuje na každém programu.
- Určí se nejběžněji používané operace – rychlost jejich provedení a spotřeba energie se optimalizuje.