



# Logické obvody

- 1. Pojem digitálneho a logického systému, správanie a štruktúra**
- 2. Analýza-formálne modely správania a opis štruktúry logických obvodov ako sú boolovské funkcie a výrazy a pojmy, konečné stavové stroje a opisné jazyky**
- 3. Syntéza a štruktúrna implementácia kombinačných logických obvodov**
- 4. Syntéza štruktúry synchrónnych sekvenčných obvodov**
- 5. Implementácia obvodov v programovateľnej logike**




# 1. Pojem digitálneho (číslicového) systému logického obvodu

- Vzt'ah modelu a technického zariadenia
- Formálny opis číslicového (digitálneho) systému,
- Triedenie logických systémov
- Pojem štruktúry digitálneho systému a jeho implementácie
- Úrovne implementácie digitálneho systému v oblasti HW implementácií
- Úlohy spojené s návrhom logických obvodov



# Architektúry počítačov

1. **Základná koncepcia počítačových systémov**
2. **Zobrazovanie informácií v počítači**
3. **Logická úroveň počítačového systému**
4. **Architektúra počítačového systému**
5. **Základy počítačových sietí**
6. **Bezpečnosť v počítačových systémoch a sieťach**
7. **Paralelné a neurónové počítače**



# 1. Základná koncepcia číslicového počítača

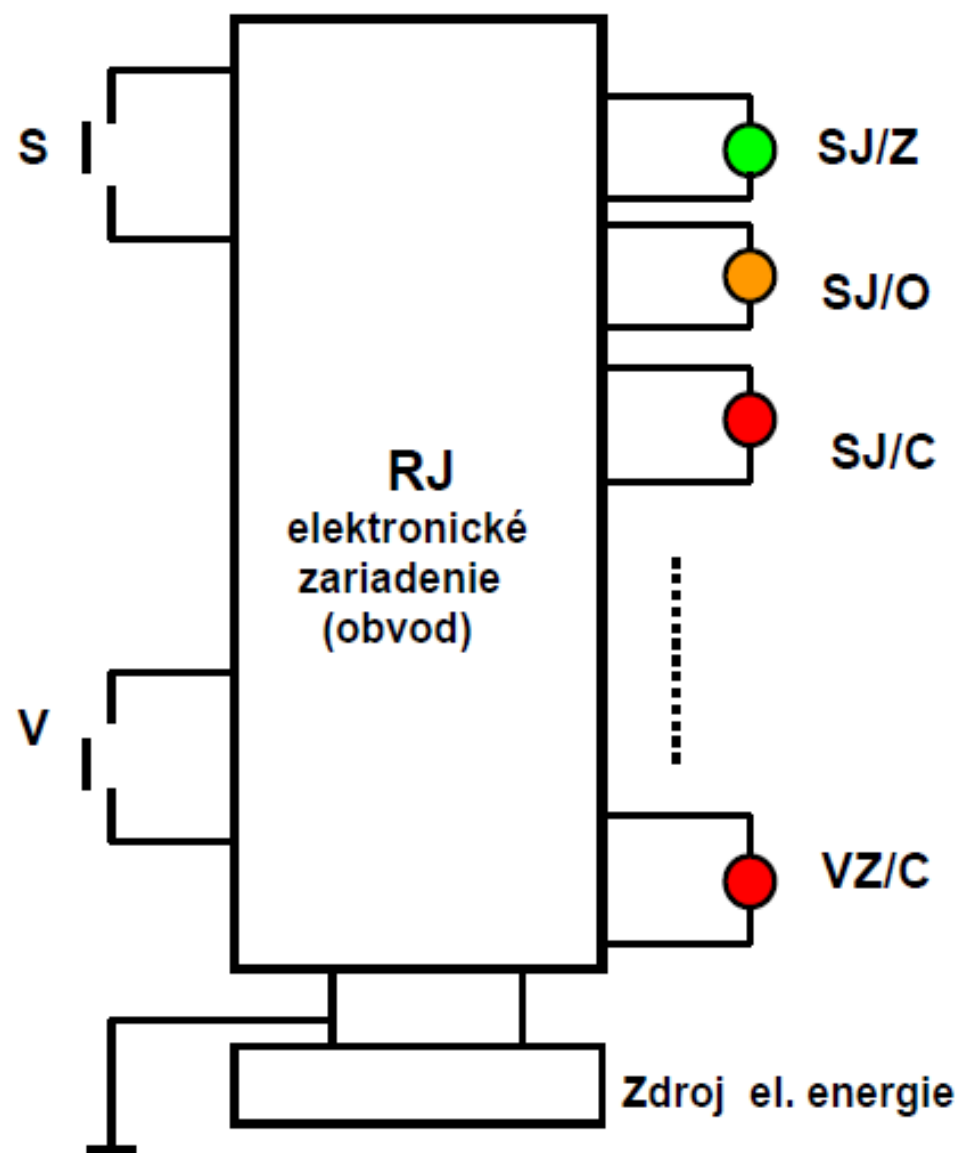
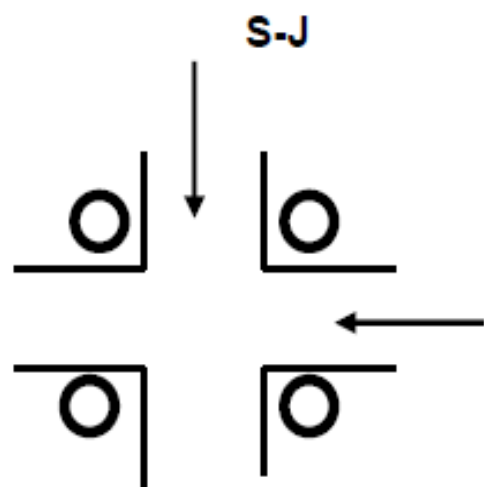
- Základná koncepcia číslicového počítača
- Počítače riadené tokom inštrukcií (von Neumannovské počítače)
- Princetonská a Harvardská architektúra
- Počítače riadené tokom údajov (data-flow systémy)
- Počítače riadené tokom údajov (data-flow systémy)
- Klasifikácia počítačov, Rozdelenie podľa aplikačného určenia, Rozdelenie podľa architektonickej koncepcie



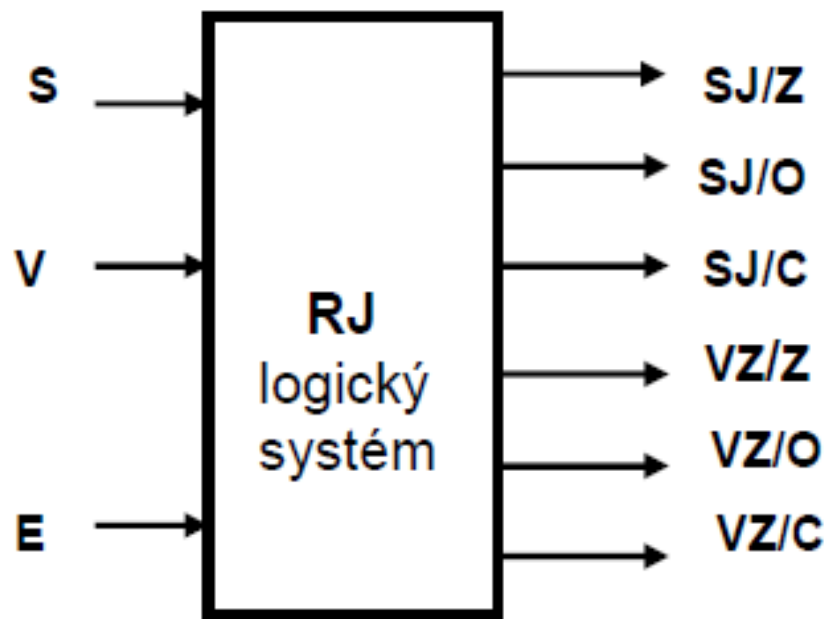
# Príklad LO

- Zariadenie na riadenie procesu prepínania svetiel na križovatke dvoch ciest v dvoch smeroch Z-V (západ - východ) a S-J (sever - juh). Toto zariadenie má "štartovacie tlačidlo", ktorým sa uvedie do činnosti (funkcie) a má "vypínacie tlačidlo, ktorým sa odstaví. Dané zariadenie automaticky ovláda zapínanie a vypínanie známych troch svetiel v každom smere: zeleného, žltého a červeného. [1]

[1] Norbert Frištacký **LOGICKÉ SYSTÉMY**, Návrh digitálnych systémov na úrovni logických obvodov, Katedra informatiky a výpočtovej techniky FEI-STU, Bratislava 2003

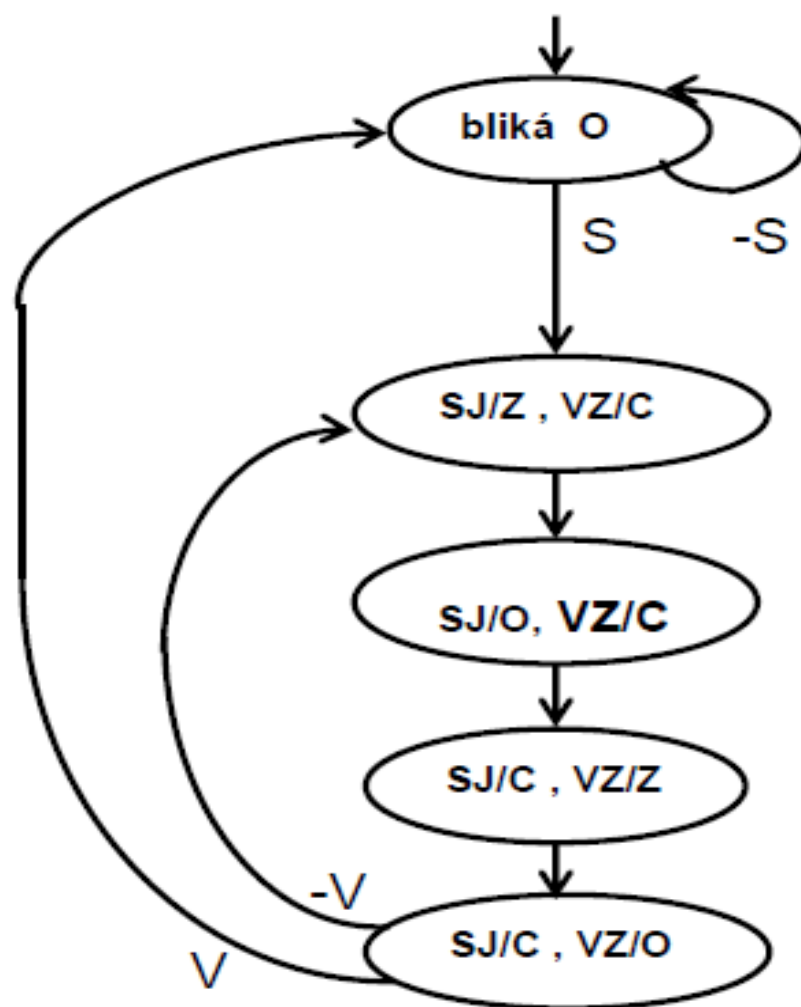


Prepínanie svetiel [1]



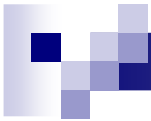
Premenné majú  
sú dvojhodnotové  
ZAP, VYP  $\Rightarrow$  0, 1

Vstupy a výstupy systému ako celku sa alternatívne nazývajú primárne vstupy resp. výstupy ale vstupné a výstupné porty

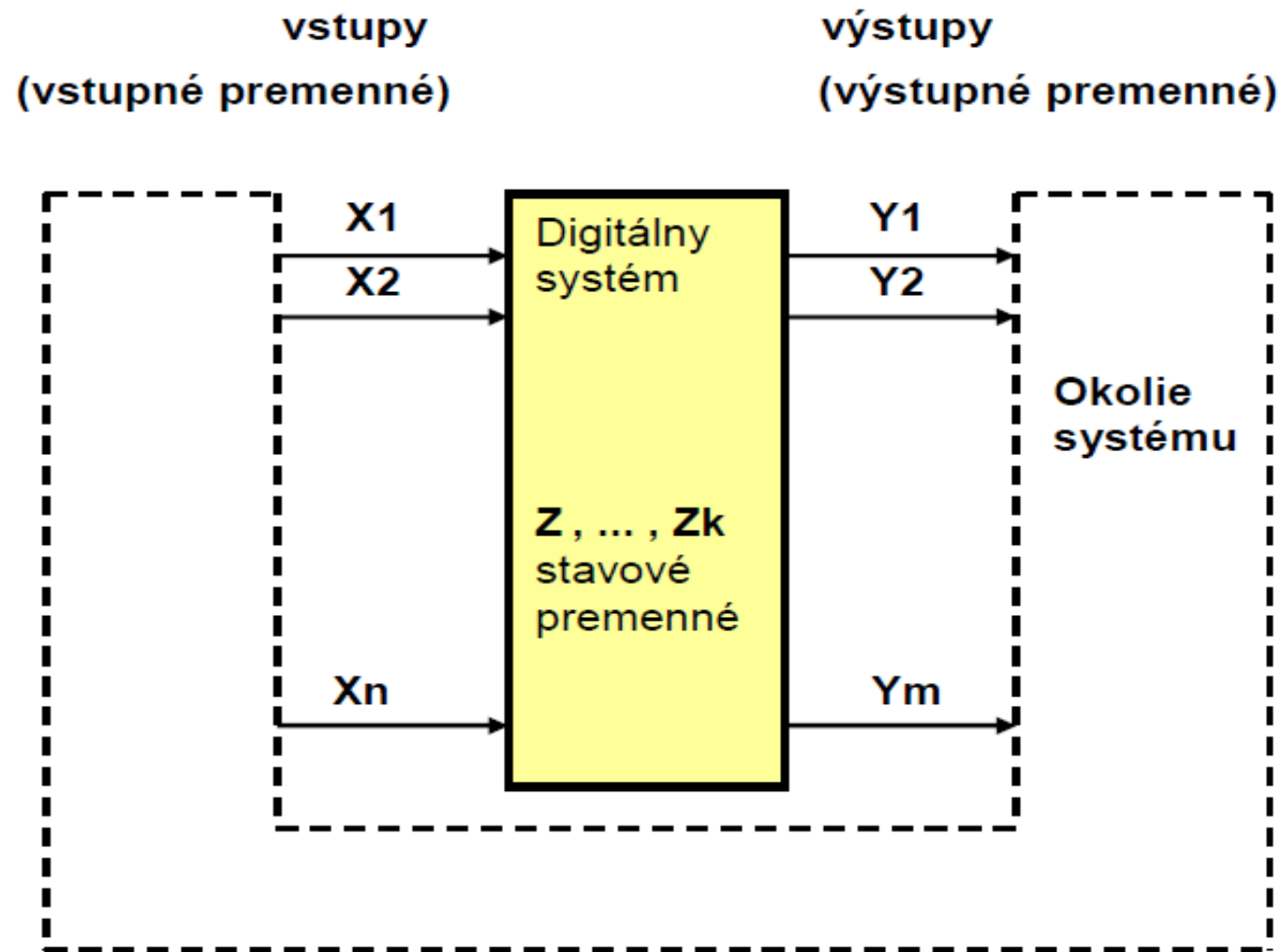


**Systém má 5 stavov, v ktorých sa generujú hodnoty výstupov pre ovládanie svetiel. Pri činnosti RJ prechádza z niektorého stavu do iného stavu podľa pravidiel do vyžadovaného stavu. „Prechody“ medzi stavmi sú vyznačené šípkami.**

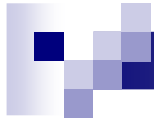




Po pripojení energie a pripravenosti zariadenia pre korektnú funkciu (po nadobudnutí hodnoty  $E = 1$ ) sa systém dostane do začiatového stavu riadiacej jednotky, pri ktorom všetky oranžové svetlá blikajú s periódou 2 s. Po indikovaní vstupného signálu  $S = 1$  začne sa RJ správať tak, že riadi zapnutie svetiel podľa požadovaného poradia s danými časovými odstupmi. RJ prechádza pritom viacerými stavmi (pozri prechodový graf hore). Pri indikovaní signálu  $V=1$  (vypni cyklovanie) RJ prejde z ľubovoľného stavu do jej začiatového stavu, v ktorom všetky oranžové svetlá blikajú. Takáto situácia je tu zakreslená (ako príklad) iba v „dolnom“ stave (v poslednom stave cyklu križovatky). Pri  $V=1$  v dolnom stave systém prechádza do začiatového stavu; pri  $V=0$  v dolnom stave sa pokračuje v činnosti cyklovania svetiel. Avšak v skutočnosti z každého stavu by mala vychádzať dvojica šípok označená  $V$  resp  $-V$ : a to pri  $V$  do začiatového stavu a pri  $-V$  do nasledujúceho stavu svetelného cyklu. Kvôli zjednodušeniu sme to v grafe vynechali. Po vypnutí energie, t. j. ak dôjde k zmene z  $E=1$  do  $E=0$  je RJ nefunkčná, nenachádza sa ani v jednom z uvedených 5 stavov.



1. Logický systém [1]
2. Cvienia rozpoznávanie vstupnej postupnosti 0,1



# Informatika je veda o:

Získavanie, zbere

Prenose

Triedení

Ukladanie

Uchovávaní

Spracovávaní, aktualizovaní

Vyhodnocovaní

Využívaní

## Informácií

signálov

údajov

symbolov

správ

poznatkov

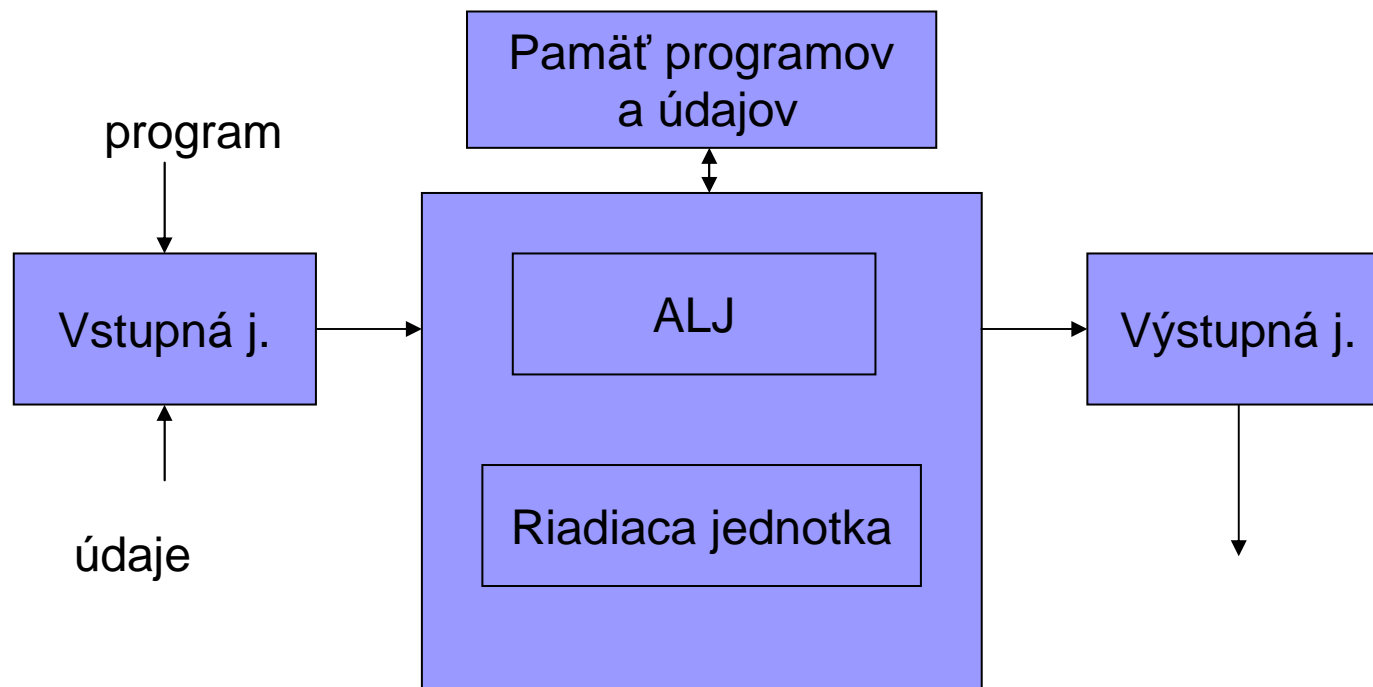
znalostí.



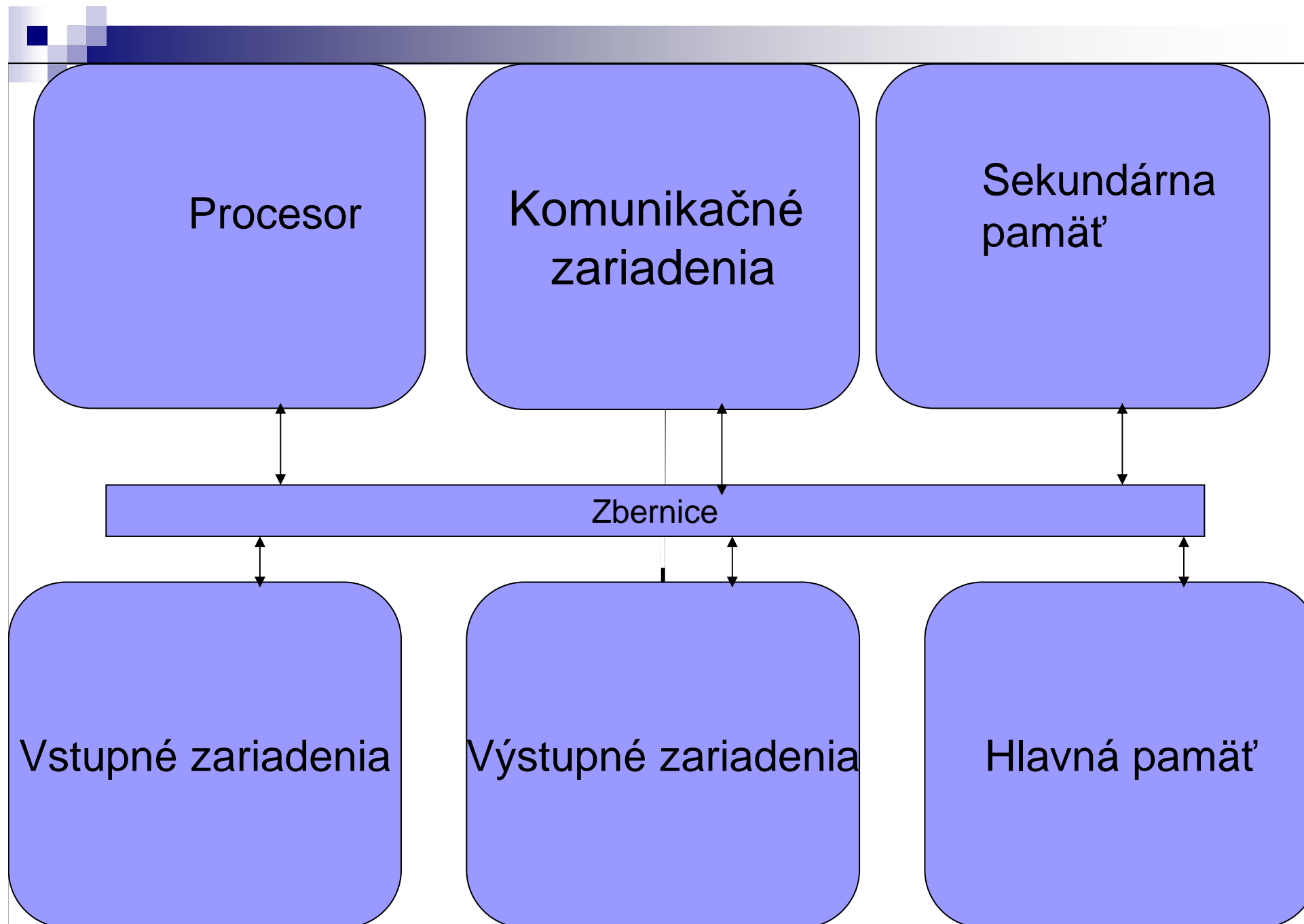
# Klasifikácia počítačov

## Kritériá

- Technické parametre
- Aplikačné určenie
- Architektonická koncepcia
- Používateľsko-aplikačná klasifikácia
- Typ spracovávania informácií
- Konštrukčno-používateľska klasifikácia
- Spôsob riadenia
- Spôsobu pamätania si údajov

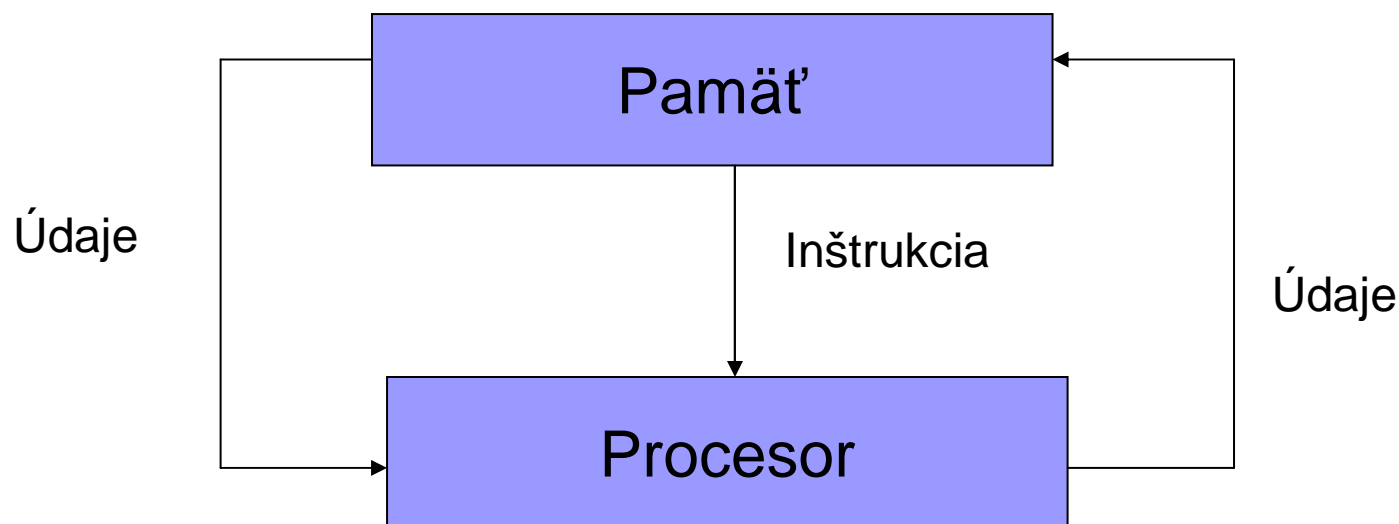


János von Neuman 1946  
Princetonská architektúra



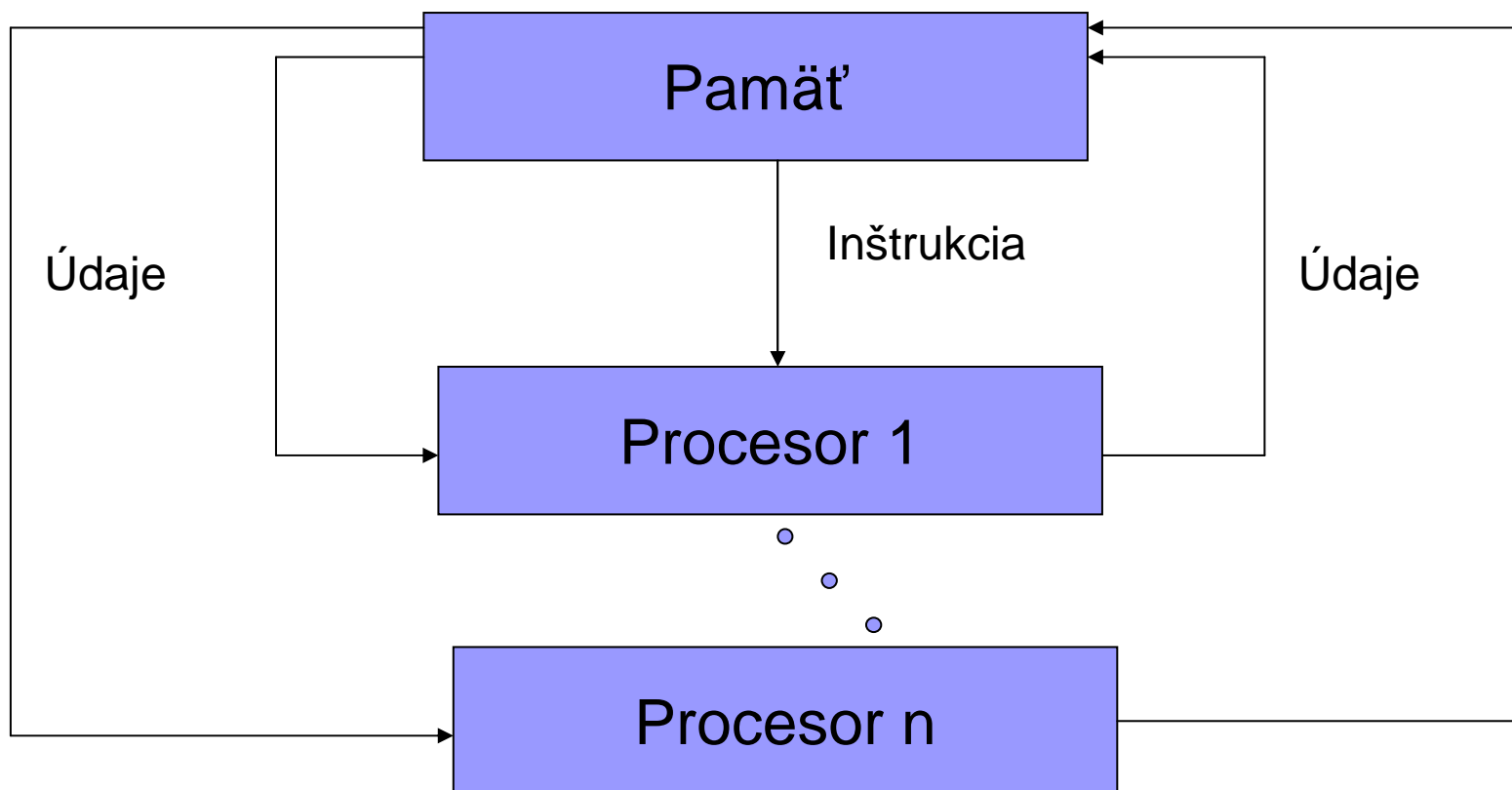
# Klasifikácia podľa architektonickej koncepcie

## ■ SISD-sériový počítač



# Klasifikácia podľa architektonickej koncepcie

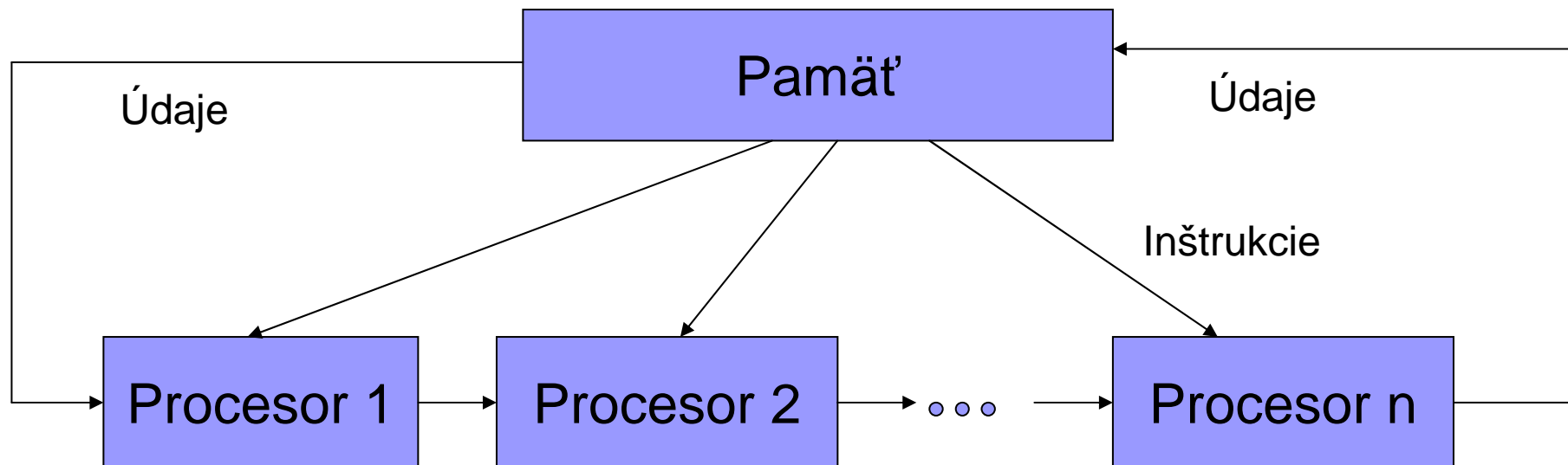
## ■ SIMD-paralelný počítač





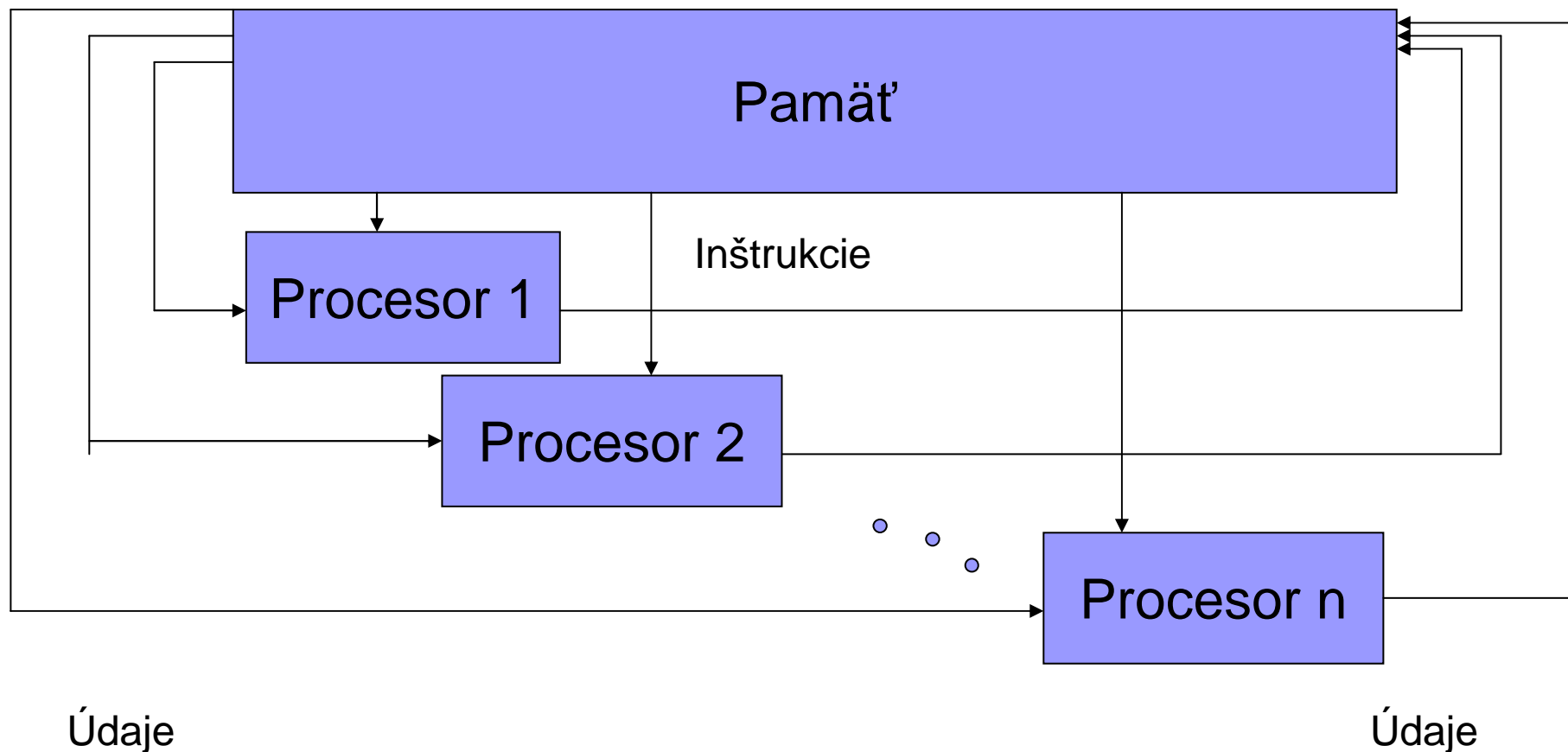
# Klasifikácia podľa architektonickej koncepcie

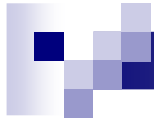
- MISD-paralelný počítač-prúdové spracovanie



# Klasifikácia podľa architektonickej koncepcie

- MIMD-viacprocesorový paralelný počítač-prúdové spracovanie



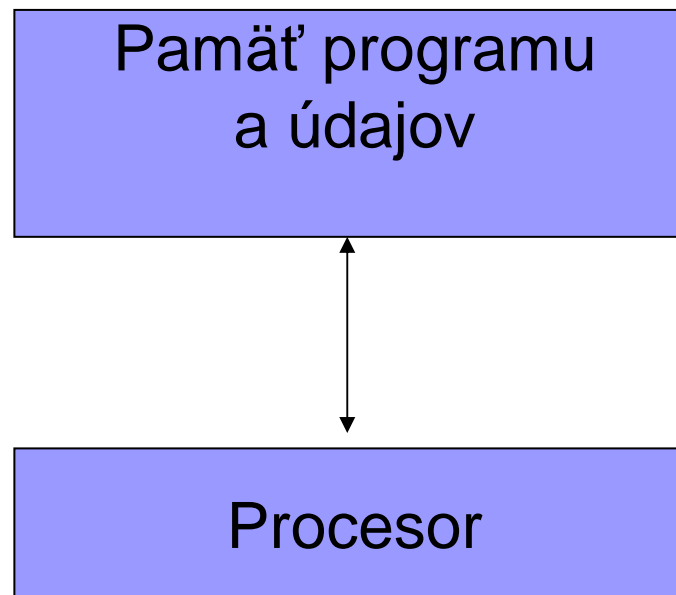


# Klasifikácia podľa spôsobu riadenia

- Počítače riadené tokom inštrukcií
- Počítače riadené tokom údajov
- Počítače riadené tokom požiadaviek

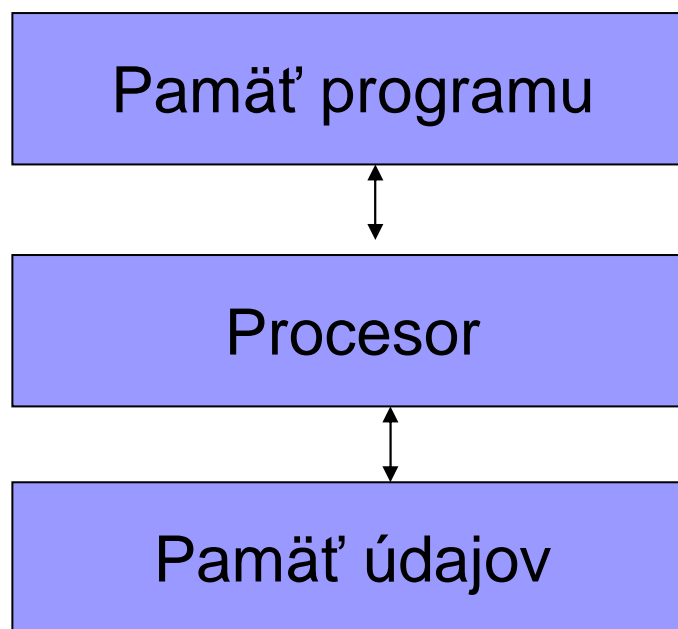
# Klasifikácia podľa spôsobu pamätania si údajov

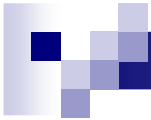
## ■ Princetonská architektúra



# Klasifikácia podľa spôsobu pamätania si údajov


## ■ Havardská architektúra





## ■ Formálne modely správania sa kombinačných obvodov

1. Opis správania – špecifikácia kombinačných obvodov
2. Zápis boolovských funkcií - Logické výrazy
3. Návod na vytvorenie Karnaughovej mapy



**Boolová algebra** je dvojhodnotová logická algebra, ktorá používa disjunkciu (logický súčet), konjunkciu (logický súčin) a negáciu (logická negácia) ako úplný súbor základných logických funkcií a slúži na matematický opis zákonov a pravidiel výrokovej logiky, ktoré riešia vzťahy medzi pravdivými a nepravdivými výrokmi:

- **pravdivý výrok** - priradená hodnota logická 1
- **nepravdivý výrok** - priradená hodnota logická 0.

**Boolovské funkcie** sú také, pri ktorých závislé aj nezávislé premenné môžu nadobúdať len hodnoty 0 alebo 1. Vo všeobecnosti zápis tejto funkcie môže mať tvar:

$$Y = f(A, B, C, \dots)$$

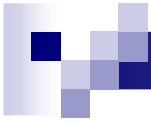
kde :

- A, B, C, ... sú nezávislé premenné (vstupne veličiny)
- Y je závislé premenná (funkčná hodnota).

$$B = \{B^{(n)}, +, *, -, 0, 1\}$$

Funkciu s n nezávisle premennými možno určiť pre všetky možné kombinácie hodnôt n premenných, t.j. pre  $N = 2^n$ . Táto funkcia sa nazýva **úplne zadaná**.

Pre n premenných existuje maximálne  $2^{2^n}$ , t.j.  $2^N$  logických funkcií.



<http://exphys.science.upjs.sk/studenti/ify/text.php?obsah=t2&tlac=0>  
<http://www.project22.sk/download/PPI.docx>

$$1 * 1 = 1$$

$$0 * 0 = 0$$

$$0 + 1 = 1 + 0 = 1$$

$$1 * 0 = 0 * 1 = 0$$

$$0 + 0 = 0$$

$$1 + 1 = 1$$

boolovské funkcie jednej a dvoch  
premených





Pre Boolovu algebru platia nasledovné zákony a pravidlá:

1. Zákon komutatívny	$A + B = B + A$	$A \cdot B = B \cdot A$
2. Zákon asociatívny	$A + (B + C) = (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
3. Zákon distributívny	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + B \cdot C = (A + B) \cdot (A + C)$
4. Zákon idempotentný	$A + A = A$	$A \cdot A = A$
	$A + 0 = A$	$A \cdot 0 = 0$
	$A + 1 = 1$	$A \cdot 1 = A$
5. Zákon doplnku	$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$
6. Zákon involúcie	$A = \bar{\bar{A}}$	
7. Zákon de Morganov	$\overline{A + B} = \bar{A} \cdot \bar{B}$	$\overline{A \cdot B} = \bar{A} + \bar{B}$
8. Zákon absorpcie	$A \cdot (A + B) = A$	$A + A \cdot B = A$
9. Zákon absorpcie negácie	$A + \bar{A} \cdot B = A + B$	$A \cdot (\bar{A} + B) = A \cdot B$
	$\bar{A} + A \cdot B = \bar{A} + B$	$\bar{A} \cdot (A + B) = \bar{A} \cdot B$

### Pierceova algebra

$$a \downarrow b = \overline{a + b}$$

### Prvá pierceova normálna forma

$$f = \overline{(a + c) \cdot (\bar{a} + b)} = \overline{(a + c)} + \overline{(\bar{a} + b)} = (a \downarrow c) \downarrow (\bar{a} \downarrow b)$$

### Druhá pierceova normálna forma

$$f = \overline{\bar{a}c + ab} = (\overline{\bar{a}c} \downarrow \overline{ab}) \downarrow = ((\overline{a + c}) \downarrow (\overline{\bar{a} + b})) \downarrow = ((a \downarrow c) \downarrow (\bar{a} \downarrow \bar{b})) \downarrow$$

### Shefferova algebra

$$a \uparrow b = \overline{a \cdot b}$$

### Prvá shefferova normálna forma

$$f = \overline{\bar{a}c + ab} = \overline{(\bar{a}c) \cdot (ab)} = (\bar{a} \uparrow c) \uparrow (a \uparrow b)$$

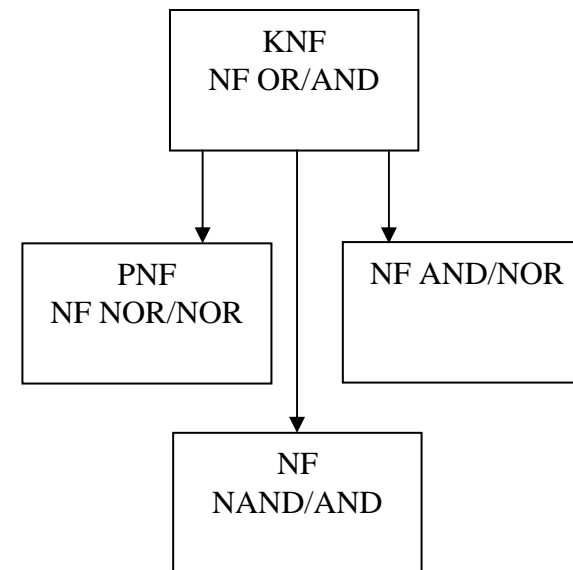
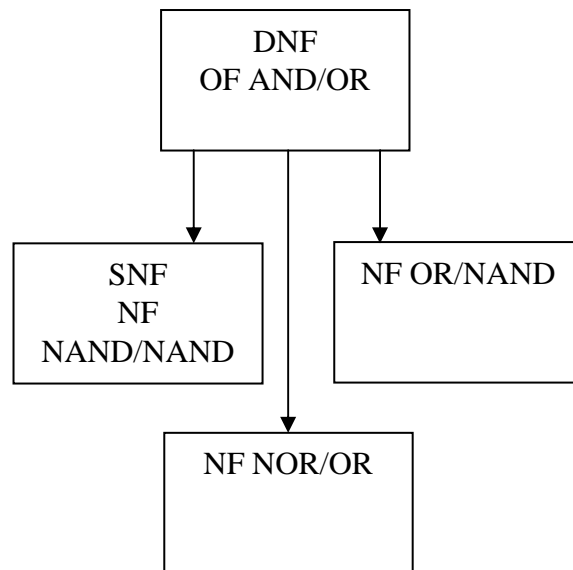
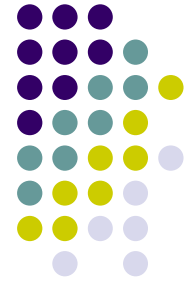
### Druhá shefferova normálna forma

$$f = \overline{(a + c) \cdot (\bar{a} + b)} = ((\overline{a + c}) \uparrow (\overline{\bar{a} + b})) \uparrow = ((\bar{a}c) \uparrow (ab)) \uparrow = ((\bar{a} \uparrow c) \uparrow (a \uparrow \bar{b})) \uparrow$$



		<u>D</u>		<u>C</u>
B	A	1	0	0
	A	1	0	1
	A	1	1	1
	A	0	1	0

		<u>D</u>		<u>C</u>
B	A	1	0	0
	A	1	0	1
	A	1	1	1
	A	0	1	0



Su ekvivalentni výrazy:

$$① \underline{(a+b)(c+\bar{b})(\bar{c}+\bar{a})}$$

$$② \underline{(a \uparrow \bar{b} \uparrow \bar{c}) \uparrow (\bar{a} \uparrow b \uparrow c)}$$

$$\begin{aligned} ① (a+b)(c+\bar{b})(\bar{c}+\bar{a}) &= (ac+bc+a\bar{b}+\bar{b}\bar{c})(\bar{c}+\bar{a}) = \\ &= (a \cdot \cancel{c} \cdot \bar{c} + b \cdot \cancel{c} \cdot \bar{c} + a\bar{b} \cdot \bar{c} + a\bar{b} \cdot \bar{c} + \bar{b} \cdot \bar{c} \cdot \bar{a} + \bar{b} \cdot \bar{c} \cdot \bar{a}) = \\ &= \underline{a\bar{b} \cdot \bar{c} + \bar{a}b\bar{c}} \end{aligned}$$

$$② (a \uparrow \bar{b} \uparrow \bar{c}) \uparrow (\bar{a} \uparrow b \uparrow c) = a\bar{b}\bar{c} + \bar{a}bc$$

$$\begin{aligned} \overline{a \cdot \bar{b} \cdot \bar{c} + \bar{a}bc} &= \overline{a \cdot \bar{b} \cdot \bar{c}} + \overline{\bar{a}bc} = \\ &= \underline{a\bar{b} \cdot \bar{c} + \bar{a}bc} \end{aligned}$$

výrazy sú ekvivalentní





Je ekvivalentní výraz?  $(\bar{a} \wedge \bar{c}) \vee (a \wedge b) \wedge (b \vee \bar{c})$

$$① (\bar{a} \wedge \bar{c}) \vee (a \wedge b) \wedge (b \vee \bar{c}) =$$

$$\bar{a} \cdot \bar{c} + ab + \overline{b \cdot \bar{c}} = \bar{a} \bar{c} + ab + \bar{b} \cdot c =$$

$$\bar{a} \bar{c} (\underbrace{b + \bar{b}}_1) + ab (\underbrace{c + \bar{c}}_1) + (\underbrace{a + \bar{a}}_1) \cdot \bar{b} \cdot c =$$

$$= \bar{a} \bar{c} b + \bar{a} \bar{c} \bar{b} + ab c + ab \bar{c} + a \bar{b} c + \bar{a} \bar{b} c$$

b	a	1.	1.
		1.	0.
		1.	1.
		0.	1.

$$② (a \cdot \bar{c}) \vee (a \vee \bar{b} \vee \bar{c}) =$$

$$= \overline{a \cdot \bar{c}} + \overline{a \vee \bar{b} \vee \bar{c}} = \overline{a \cdot \bar{c}} + \overline{a + \bar{b} + \bar{c}} =$$

$$= (\bar{a} + c) \cdot (a + b + c) = \bar{a}b + ac + \bar{a}\bar{b} + \bar{b}c + \bar{a}\bar{b}c$$

$$= ac + \bar{a}\bar{c} + \bar{a}\bar{b} + \bar{b}c = \underbrace{abc + a\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c}_c$$

$$= abc + a\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c$$

$$= abc + a\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c + \bar{a}\bar{b}c$$

a	b	1.	1.
		1.	0.
		0.	1.
		0.	1.

upravené na VDNF

VDNF má zápis, neboť to

je v mape

výrazy mi nejsou ekvivalentní

# Skupinové minimalizácie - príklady

Minimalizácia petroll. funkcií  $x_3 \rightarrow$  MODROU

$$x_1 \mid \begin{array}{c|c|c|c} & x_2 & x_3 & \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}$$

$$x_1 \mid \begin{array}{c|c|c|c} & x_2 & x_3 & \\ \hline 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}$$

①

$$x_1 \mid \begin{array}{c|c|c|c} & x_2 & x_3 & \\ \hline 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{array}$$

$$x_1 \mid \begin{array}{c|c|c|c} & x_2 & x_3 & \\ \hline 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array}$$

Skupinová minimalizácia  $f_1, f_2 \rightarrow$  červená

$$f_1 = \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 x_3$$

$$f_2 = \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 x_3 + \bar{x}_2 x_3$$

treba 4 + 2 NAND obvodov  
 $\downarrow$   
 1. stupeň  $\rightarrow$  2. stupeň

2. stupeň minimalizácie

②

$$x_1 \mid \begin{array}{c|c|c|c} & x_2 & x_3 & \\ \hline 0 & x & 1 & x \\ x & 1 & 0 & 0 \end{array}$$

$$x_1 \mid \begin{array}{c|c|c|c} & x_2 & x_3 & \\ \hline 1 & 0 & 1 & 0 \\ x & 1 & 0 & x \end{array}$$



**Kombinačný logický systém** - má správanie, ktoré môžeme opísať funkciou  $Y = f(X)$  kde  $X$  je množina vstupných a  $Y$  výstupných premenných (vektorov, výstupné premenné závisia iba od vstupných premenných (vstupných vektorov) v danom čase.

**Sekvenčný logický systém** - je charakteristický tým, že výstupné premenné závisia nielen od vstupných premenných v danom časovom okamihu, ale aj od postupnosti vstupných premenných v predchádzajúcich časových okamihoch. V závislosti od postupnosti vstupných premenných môže teda sekvenčný obvod v danom čase generovať rôzne hodnoty výstupných premenných. Chovanie sa sekvenčného logického systému (obvodu) teda vyjadruje jeho pamäťovú schopnosť





- Na vstupe logického systému pôsobia vstupné signály (veličiny)  $x_1, x_2, \dots, x_m$ , ktoré menia svoju hodnotu v čase nezávisle od systému. Systém má ďalej výstupné signály (veličiny)  $y_1, y_2, \dots, y_n$ , ktorých funkčne závisia od hodnôt vstupných veličín.
- V sekvenčnom systéme sú vzťahy medzi hodnotami výstupných a vstupných veličín sú vo všeobecnosti sprostredkované určitými vnútornými veličinami stavovými veličinami systému  $z_1, z_2, \dots, z_p$ .



Ak tieto časové okamihy zmien závisia len od okamihov zmien vstupných premenných, hovoríme o **asynchrónnom sekvenčnom logickom systéme**. Ak tieto časové okamihy zmien závisia nielen od okamihov zmien vstupných premenných ale aj od synchronizačnej alebo hodinovej premennej (CLK), hovoríme o **asynchrónnom sekvenčnom logickom systéme**.



## KONEČNÉ STAVOVÉ AUTOMATY

Konečný stavový stroj - automat (Finite State Machine = FSM) je algebrický systém

$$A = (X, S, Y, p, v), A = (X, S, Y, p, v, s_0),$$

kde

$X \Rightarrow$  množina vstupných symbolov, vstupov

$S \Rightarrow$  množina stavov

$Y \Rightarrow$  množina výstupných symbolov, výstupov

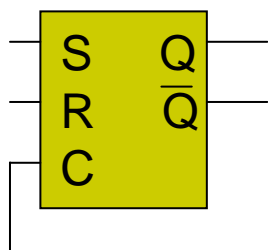
$p \Rightarrow$  prechodová funkcia  $p: S \times X \rightarrow S$

$v \Rightarrow$  výstupná funkcia  $v: S \times X \rightarrow Y$  (Mealy)

$v: S \rightarrow Y$  (Moore)



synchronný PO SR



	S		R		Q
0	0	1	X	0	0
1	1	1	X	0	1

	S		R
P	1	X	0

synchronný PO JK

	J		K		Q
0	0	1	1	0	0
1	1	1	0	0	1

univerzálny

všetky typy správania

	J		K
P	1	K	0

synchrónny PO-D

	<u>D</u>		Q
0	0	1	0
1	0	1	1

<u>D</u>	
0	1

synchrónny PO-T

	<u>T</u>		Q
0	0	1	0
1	1	0	1

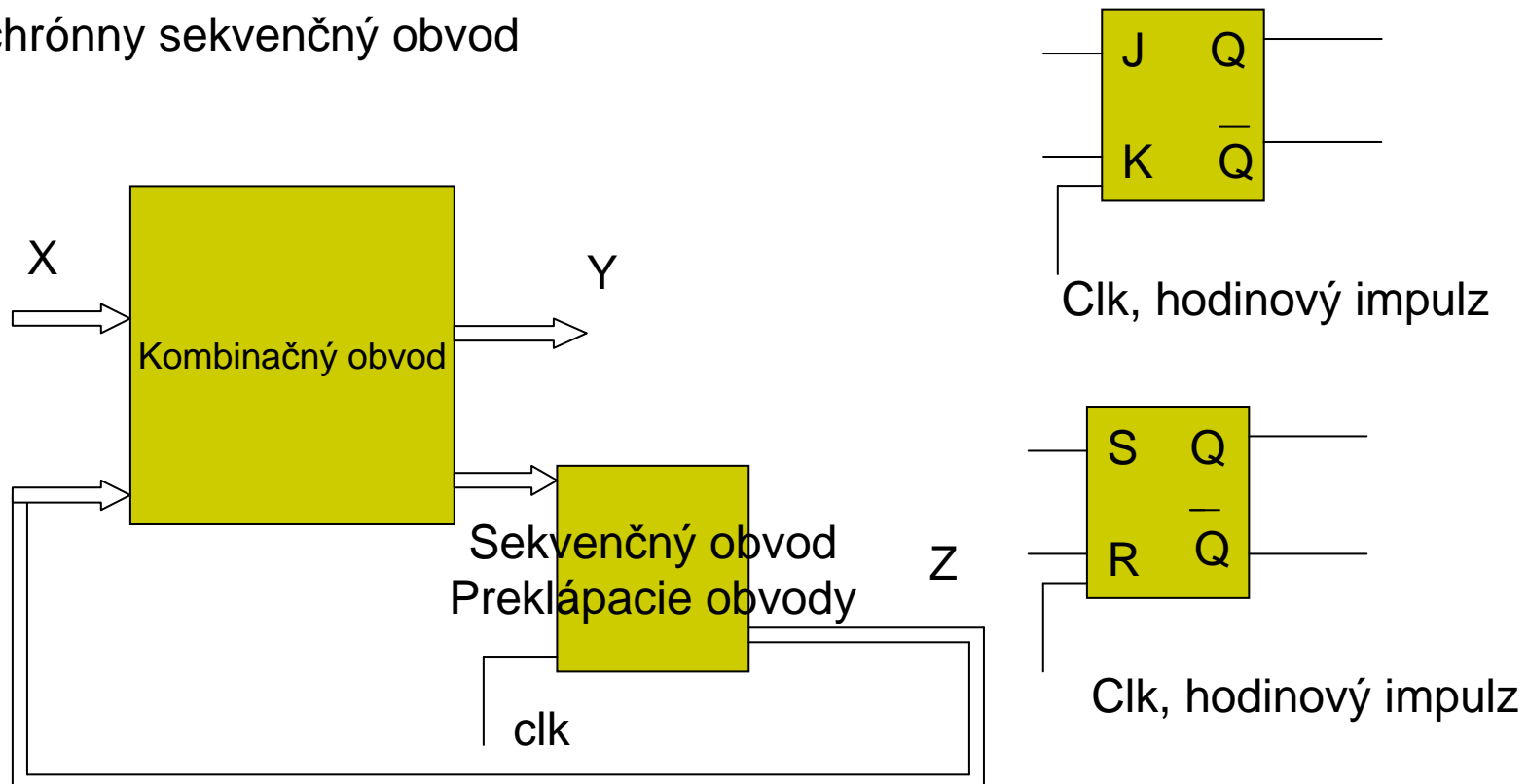
<u>T</u>	
P	K



**Podľa prednášky LOGICKÉ SYSTÉMY**, Návrh digitálnych systémov na úrovni logických obvodov, Norbert Frištacký, Katedra informatiky a výpočtovej techniky, FEI-STU, 2003



### Synchrónny sekvenčný obvod





Základným formálnym špecifikačným prostriedkom je tzv. **fundamentálny stavový stroj** (FSM, automat)

$$M = (A, Q, U, p, v),$$

ktorý má nasledujúcu vlastnosť: pri každom stave  $q \in Q$  a pri každom vstupnom symbole  $v \in A$  platí:

vstupné slovo:  $v \quad v \quad v \quad .. \quad v \quad v \quad v \quad v \quad v \quad ....$

postupnosť stavov:  $q \quad q_1 \quad q_2 \quad .. \quad q_k \quad q' \quad q' \quad q' \quad q' \quad ....$ ,

kde  $k \geq 0$  je pevné číslo pre danú dvojicu  $(q, v)$

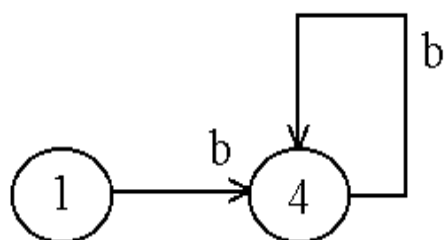
**Formálne:** Pri dostatočne veľkom čísle "n" pre každý stav  $q$  a vstupný symbol  $v$  platí:

$$p(q, vn) = p(q, vn+1), \quad n \geq k$$



Stav "q" v danom FA sa nazýva **stabilný pre daný vstupný symbol "v"** práve vtedy ak platí  $p(q, v) = q$ .

Vo fundamentálnom FSM pre každý stav q a vstupný symbol v platí, že jeho dostatočnom opakovaní sa automat **dostane do stabilného stavu** pre vstupný symbol v.



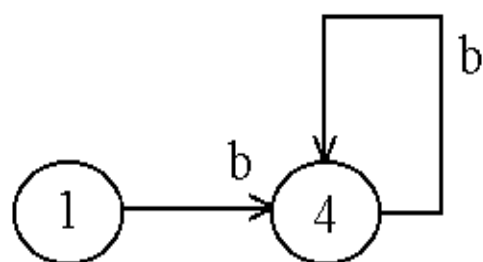
V prech.  
grafe

M1	a	b	c	d	
1	①	4	①	①	u
2	1	②	②	4	w
3	③	2	1	③	r
4	3	④	1	④	h
S	p				v

$p(1, a) = p(1, aa) = 1$   
 $p(1, b) = p(1, bb) = 4$   
 $p(2, d) = p(2, dd) = 4$   
 $p(2, a) = p(2, aa) = 1$   
 atd.

fundamentálny automat prvého rádu



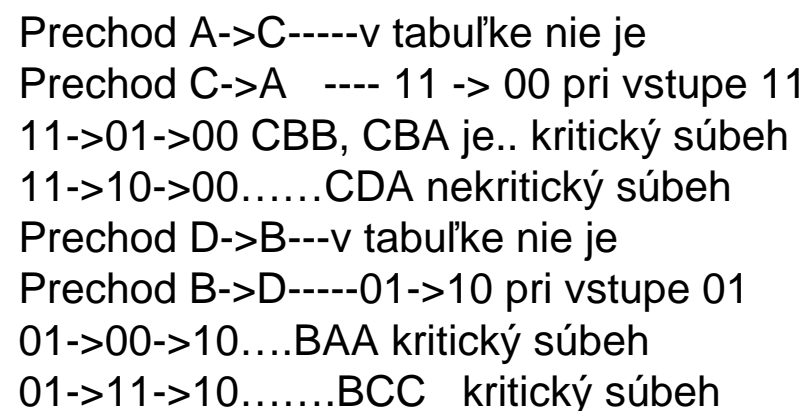
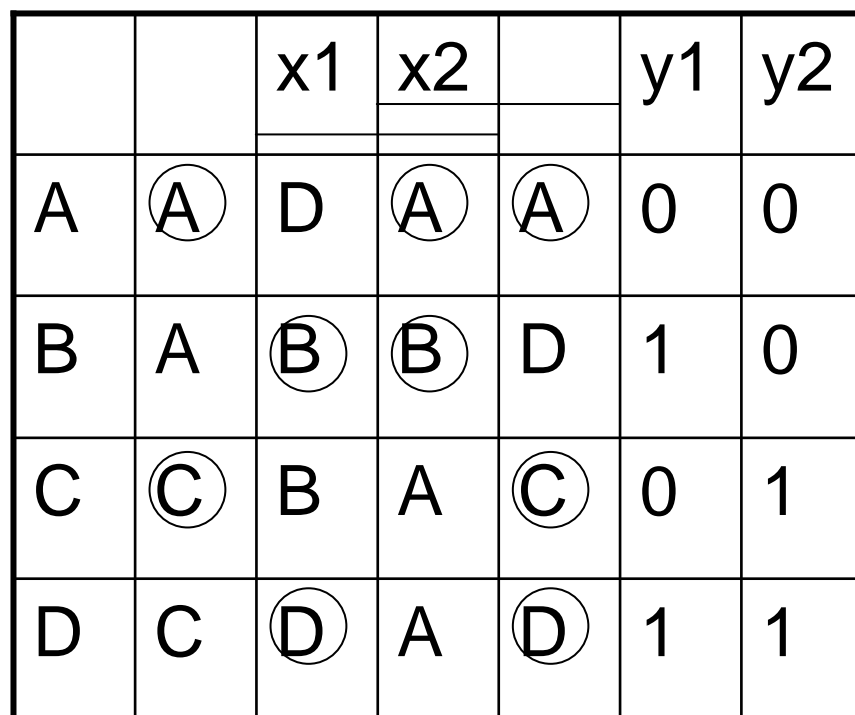


V prech.  
grafe

M1	a	b	c	d	
1	①	4	①	①	u
2	1	②	②	4	w
3	③	2	1	③	r
4	3	④	1	④	h
S	p				v

$p(1, a) = p(1, aa) = 1$   
 $p(1, b) = p(1, bb) = 4$   
 $p(2, d) = p(2, dd) = 4$   
 $p(2, a) = p(2, aa) = 1$   
 atd.

FFSM 1.rádu rozpoznať podľa toho, že prechodovej funkcií musí platiť: Ak je pri niektorom vstupnom symbole (vektore) t.j. v príslušnom stĺpci v prechodovej tabuľke niektorý stav q nezakrúžkovaný, potom musí byť pri tomto vstupnom symbole (v tomto stĺpci) aj stav q zakrúžkovaný.

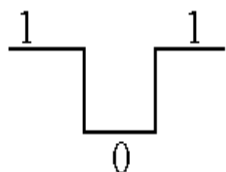




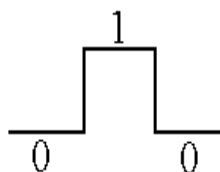
A	B
C	D

A	C	2	B
D	3		1

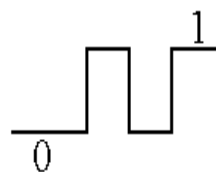
Univerzálny kód



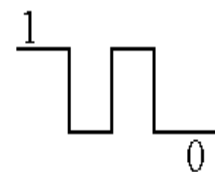
**statický  
hazard v 1**



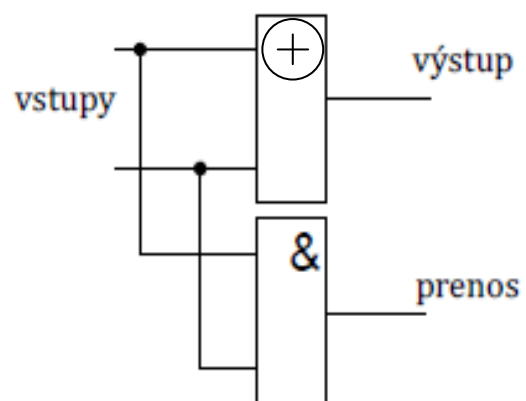
**statický  
hazard v 0**



**dynamický  
hazard pri 0→1**



**dynamický  
hazard pri 1→0**

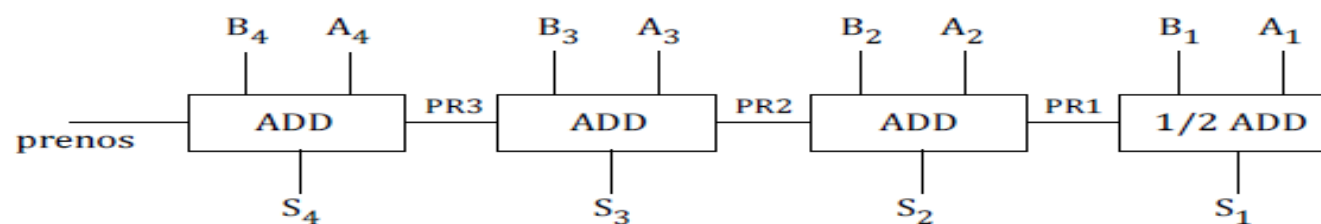
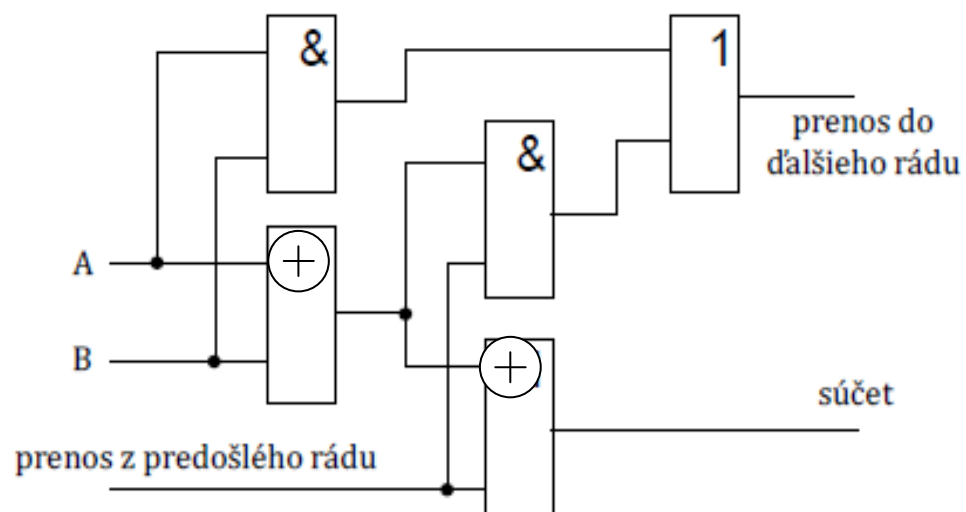


Polovičná sčítacia

$\oplus$  XOR

Úplná sčítacia

Sériová sčítacia



# Prepojovací podsystém počítača

**Zbernice** (prepojený každý podsystém počítača s každým)  
(v jednom okamihu len 1 vysielateľ)



## Rozdelenie zberníc

### 1. Podľa spôsobu riadenia

Single master iba 1 nadriadený podsystém-master

Multi-master – každé zariadenie môže riadiť zbernicu, ale v danom okamžiku iba jedno

### 2. Podľa synchronizácie prenosu

Synchrónne zbernice synchronizované synchronizačným impulzom

Asynchrónne zbernice-prenos synchronizovaný odpoveďou  
podriadeného, pomalšie

# Rozdelenie zberníc



## 3. Podľa časového multiplexu

Multiplexované zbernice-druh informácie sa mení s časom (adresa, inštrukcia, údaj)

Nemultiplexované zbernice – význam a druh informácie sa s časom nemení

## 4. Podľa tvaru prenášaných údajov

Paralelné zbernice

Sériové zbernice (prenos bit po bite)

Dnešné počítačové zbernice sú paralelné, asynchrónne, nemultiplexované a skladajú sa z nasledujúcich sekcií – adresová (adresa pamäte, v/v zariadenia), údajová (inštrukcie), riadiaca (povely, žiadosti)



## Processor

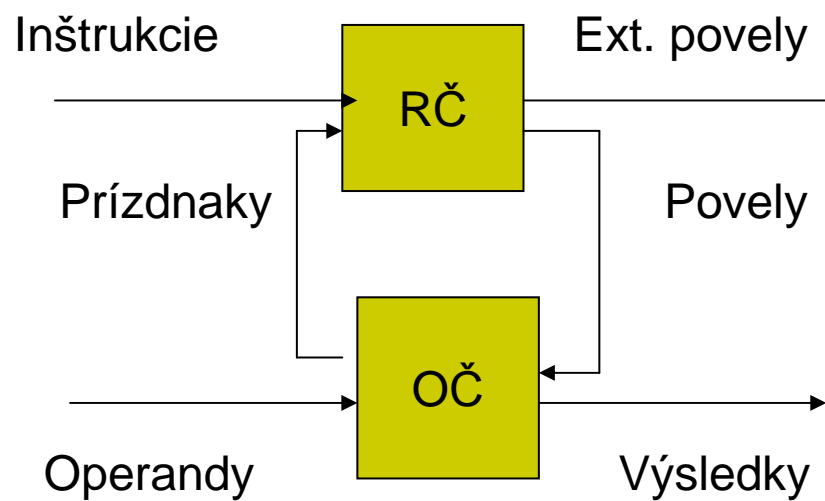
- Interpretuje inštrukcie programu
  - Výber inštrukcie z pamäte
  - Vykonanie operácie s operandami
  - Realizuje sa prenos informácií medzi časťami počítača

Univerzálne procesory (bohatý, úplný inštrukčný súbor)

Problémovo-orientované procesory špecializované funkcie

- **Hlavné časti procesora**

- Operačná časť
- Riadiaca časť







## Operačná časť

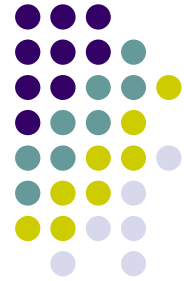
- Aritmeticko logická jednotka, ALU
- Registre
- Komunikačné obvody pre medziregistrové prenosy

### ALJ

- Paralelná dvojková sčítačka
- Sériové sčítačky
- Funkčné jednotky pre logické operácie
- Posúvacie obvody, logický posun, aritmetický posun, kruhový posun

# Prepojovací podsystém počítača

## Zbernice (prepojený každý podsystém počítača s každým)



### Rozdelenie zberníc

#### 1. Podľa spôsobu riadenia

Single master iba 1 nadriadený podsystém-master

Multi-master – každé zariadenie môže riadiť zbernicu, ale v danom okamžiku iba jedno

#### 2. Podľa synchronizácie prenosu

Synchrónne zbernice synchronizované synchronizačným impulzom

Asynchrónne zbernice-prenos synchronizovaný odpoveďou podriadeného, pomalšie čaká sa na potvrdenie prenosu, vhodné na spojenie zariadení s rôznou prenosovou rýchlosťou

#### 3. Podľa časového multiplexu

Multiplexované zbernice-druh informácie sa mení s časom (adresa, inštrukcia, údaj)

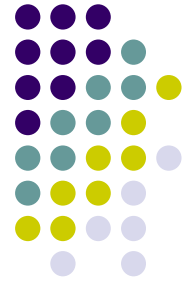
Nemultiplexované zbernice – význam a druh informácie sa s časom nemení

#### 4. Podľa tvaru prenášaných údajov

Paralelné zbernice

Sériové zbernice (prenos bit po bite)

# Rozdelenie zberníc



**Dnešné počítačové zbernice sú paralelné, asynchrónne, nemultiplexované a skladajú sa z nasledujúcich sekcií :**

adresová (adresa pamäte, v/v zariadenia),  
údajová (inštrukcie, údaje),  
riadiaca (povely, žiadosti)

Na zbernici sú definované signálové sledy-časové priebehy signálov, ktoré sa musia dodržať



# Processor

interpretuje inštrukcie programu

- Výber inštrukcie z pamäte
- Vykonanie operácie s operandami
- Realizuje sa prenos informácií medzi časťami počítača

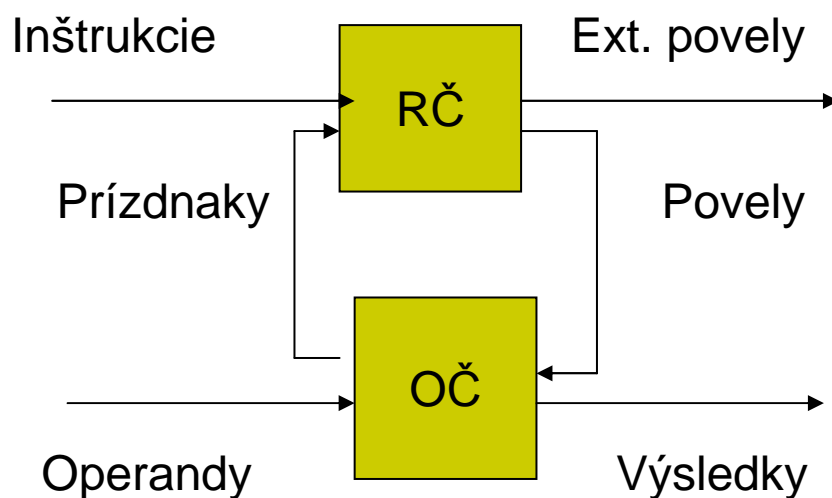
Univerzálne procesory (bohatý, úplný inštrukčný súbor)

Problémovo-orientované procesory špecializované funkcie napr. numerické koprocessory, grafický procesor, v/v procesory

# Hlavné časti procesora



- Riadiaca časť  
Výber inštrukcii, dekodovanie, zabezpečenie ich vykonania
- Operačná časť  
Operácie s operandami



# Operačná časť procesora



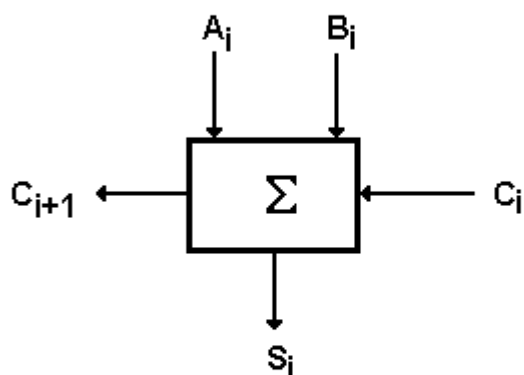
- Aritmeticko logická jednotka (ALU) určená na vykonanie inštrukcií
- Registre-prechodné uloženie operandov vstupujúcich do operácií a na uloženie výsledkov
- Komunikačné obvody na vykonávanie medziregistrových prenosov

## ALJ-ALU

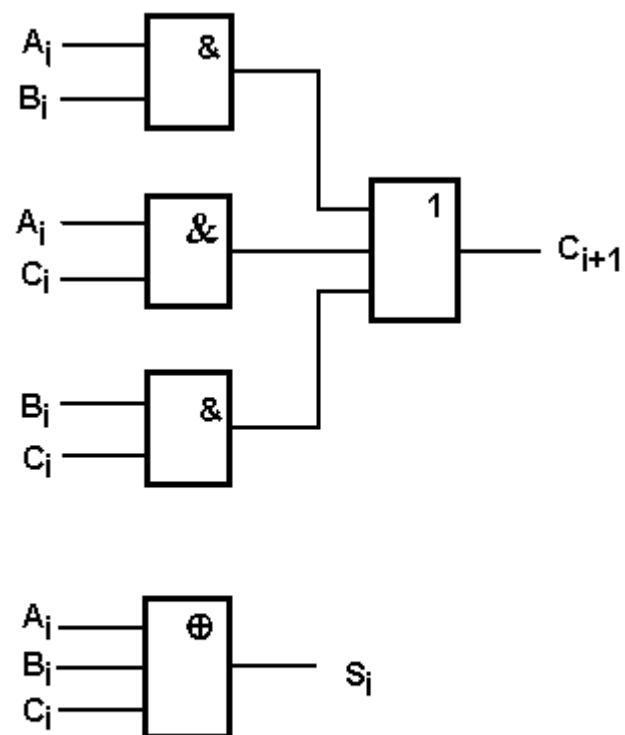
- Paralelná dvojková sčítačka dvoch n-bitových operandov sa používa na realizáciu základných aritmetických operácií, na výstupe n-bitový operand a prenos
- Sériové sčítačky, dvojková sčítačka a preklápací obvod  
Sčítačku možno realizovať i pomocou pevnej pamäte ROM
- Funkčné jednotky pre logické operácie
- Posúvacie obvody, logický posun, aritmetický posun, kruhový posun



## *i*-ty rád paralelnej dvojkovej sčítačky



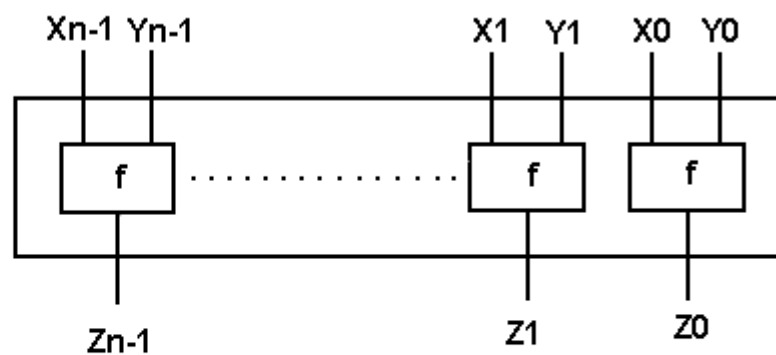
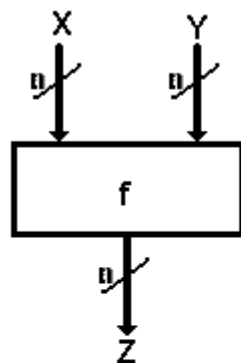
Bloková schéma



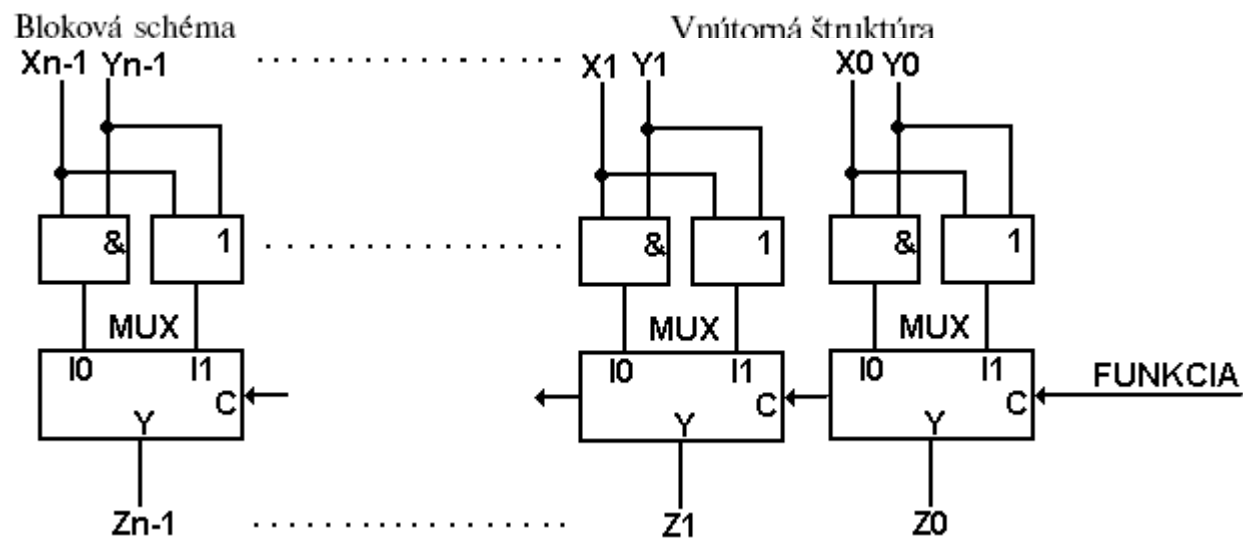
Vnútrotná štruktúra

$$S_i = (A_i \oplus B_i) \oplus C_i$$

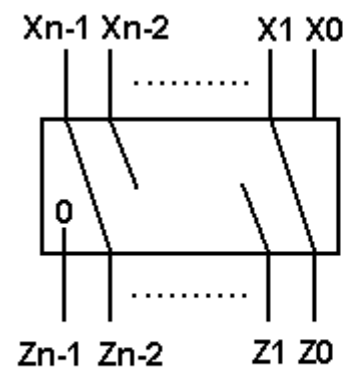
# Logické operácie



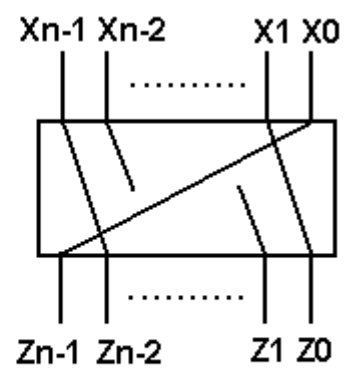
$f \in \{\text{OR, NOR, AND, ...}\}$



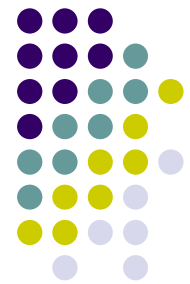




Logický posuv



Kruhový posuv



# Operačná časť procesora



- **Registre** (napr. ACC, príznakový register, ...)
- **Predikáty, príznaky** = dvojhodnotové funkcie nad hodnotami premenných, v procesore sa uchovávajú na ďalšie spracovanie a vyhodnotenie v špeciálnom **príznakovom registri**. (CF – príznak prenosu (carry flag) PF - príznak parity (parity flag) AF - príznak pomocného prenosu (auxiliary carry flag) ZF - príznak nuly (zero flag) SF - príznak znamienka (sign flag) OF - príznak preplnenia (overflow flag)...
- **Prepojovacie obvody**.
  - Spájajú jednotlivé prvky operačnej časti.
  - Realizujú medziregistrové prenosy
  - Dva spôsoby realizácie
    - multiplexory a demultiplexory - prepojenie registrov s ALJ.
    - zbernice.

# Riadiaca časť procesora



- Uskutočňuje výber inštrukcií
- Dekódovanie inštrukcií
- Vykonanie inštrukcií
- Riadi spoluprácu procesora s okolím
- Inštrukčný cyklus sa spravidla skladá zo 6 fáz
  - Výber inštrukcie z pamäte - instruction fetch IF
  - Dekódovanie inštrukcie - decode D
  - Výpočet adresy operandov – operand address OA
  - Výber operandov z pamäte alebo zo vstupného zariadenia operand fetch OF, odpadne pokiaľ sú operandy v registroch
  - Vykonanie požadovanej operácie s operandami – execute EX
  - Zápis výsledku do pamäte alebo výstupného zariadenia – S
- V počítačoch von Neumana (SISD) sa jednotlivé fázy vykonávajú postupne-sériovo **IF D OA OF EX S IF D OA OF EX S IF .....**, jediný paralelizmus , paralelná sčítacia

## Riadiaca časť procesora



### Registre

počítadlo inštrukcií (program counter - PC)  
inštrukčný register (instruction register - IR).

PC je register v ktorom je adresa inštrukcie v operačnej pamäti, ktorá sa má práve vykonať.

IR je register do ktorého sa načíta inštrukcia, ktorá sa bude vykonávať z operačnej pamäti a ktorej adresa je v PC. Po načítaní inštrukcie do IR sa obsah PC zväčší o jednu, čím sa určí adresa nasledujúcej inštrukcie. Pokiaľ je načítana inštrukcia inštrukcia skoku, obsah PC sa prepočíta podľa adresy v inštrukcii skoku.

# Riadiaca časť procesora

## FORMÁT INŠTRUKCIE

**KÓD ADRESA** (code address)

0,1, 2, 3 - adresové inštrukcie

## Typy inštrukcií

Presunové inštrukcie

Výpočtové inštrukcie

Skokové inštrukcie

Riadiace inštrukcie



# Riadiaca časť procesora

## Typy inštrukcií



- Presunové inštrukcie  
( $R \rightarrow R$ ,  $R \rightarrow M$ ,  $M \rightarrow R$ ,  $M \rightarrow M$ ,  $R \rightarrow VV$ ,  $VV \rightarrow R$ ,  $M \rightarrow VV$ ,  $VV \rightarrow M$ )
- Výpočtové inštrukcie (ADD, MUL, TEST, AND,...)
- Skokové inštrukcie  
(podmienené JNZ, nepodmienené skoky JMP, skok do podprogramu, návrat z podprogramu, prerušenia),  
zásobník, ...
- Riadiace inštrukcie (špeciálne operácie jako nastavovanie, nulovanie príznakov...)

# Riadiaca časť procesora



Adresovanie	V inštrukcii	V registri	V pamäti	Príklad
Implicitné : <i>Registrové</i> <i>Nepriame registrové</i> <i>Zásobníkové</i>		Operand Adresa Adresa	Operand Operand	SCASW MOVSB POPF
Bezprostredné	Operand			MOV CX, 10H
Registrové	Register	Operand		MOV BX, DI
Priame	Adresa		Operand	MOV ALFA, DX
Nepriame	Adresa 1		Adresa2	JMP WORD PTR DST
Nepriame registrové	Register	Adresa	Operand	JMP WORD PTR [BX]
Indexové	Indexový reg. *Posunutie	Index	Operand	MOV QQQ[DI], AX
Bázovo-indexové	Bázový reg. Indexový reg. *Posunutie	Bázová adresa Index	Operand	MOV AX, QQ[BX][SI]

Krajčovič, T. "Počítače, STU, 2000, 157 str.



## Riadiaca časť

- **Riadiaca časť s pevnou logikou (sekvenčný synchrónny obvod)**
  - ***Vstupný vektor*** tohto sekvenčného obvodu je tvorený *inštrukciami, signálmi z externého okolia* (napr. žiadosť o prerušenie, signál pripravenosti periférie atď.) a *príznakmi z operačnej časti*.
  - ***Výstupný vektor*** sa skladá z *povelov pre operačnú časť* (napr. signál zápisu do registra, nastavenie funkcie aritmeticko-logickej jednotky atď.) a *pre externé okolie* (napr. signál čítania z pamäte, zápisu do výstupného zariadenia, potvrdenia prerušenia atď.).





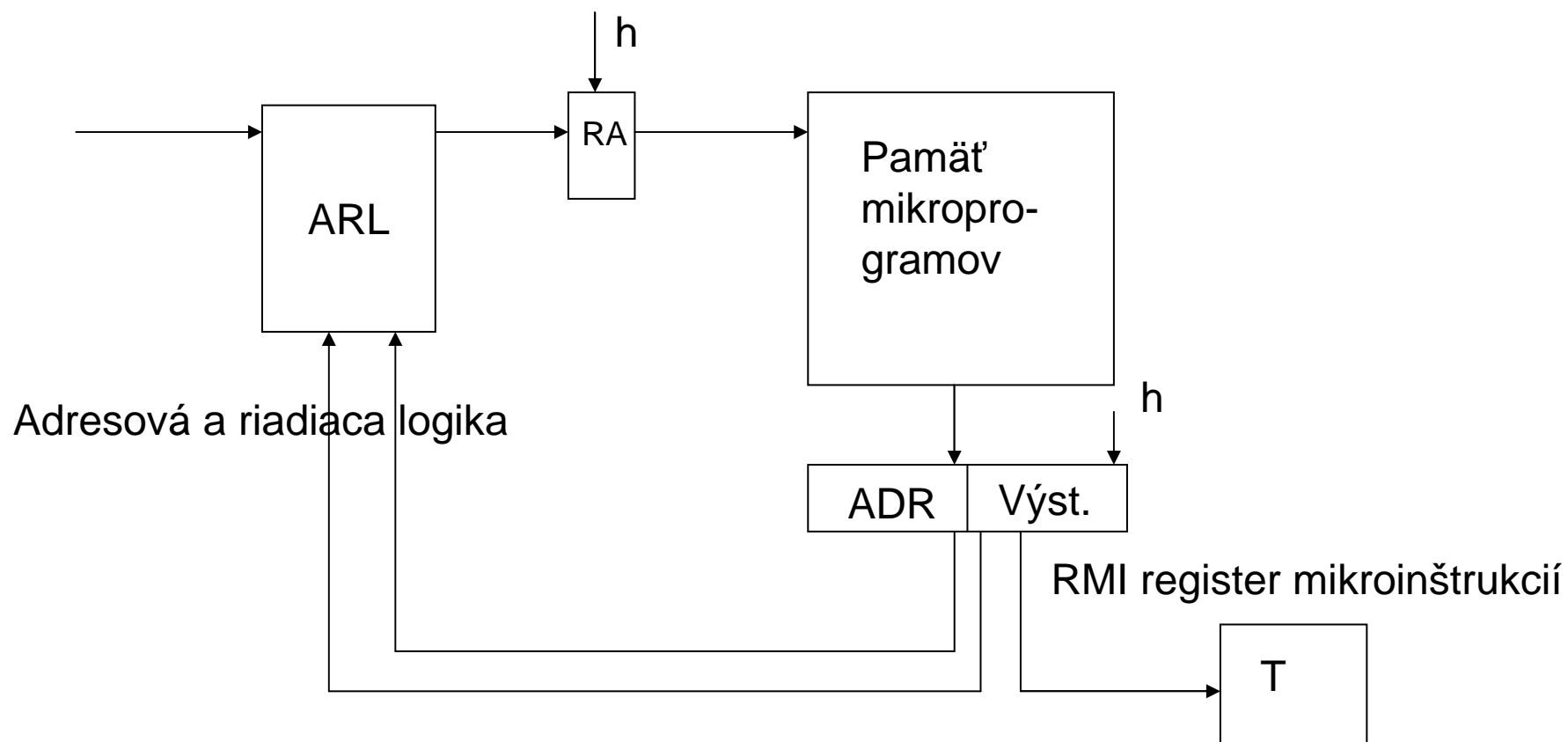
## Riadiaca časť

- **Riadiaca časť s programovateľnou logikou**
  - Synchronný systém, v ktorom je inštrukcia realizovaná vykonaním mikroprogramu
  - **Mikroprogram** je postupnosť *mikroinštrukcií*, ktoré sú uložené v *pamäti mikroinštrukcií*. Mikroprogramovú riadiacu jednotku môžeme teda charakterizovať ako špecializovaný procesor, ktorý priamo interpretuje mikroprogramy, uložené v svojej pamäti mikroprogramov a vykonáva ich na danej operačnej časti.
  - Zmenou obsahu pamäte mikroinštrukcií je možné dosiahnuť zmenu inštrukčného súboru procesora. Takýmto spôsobom je potom možné na jednom procesore vykonávať programy pre iný typ procesora. Vtedy hovoríme o **mikroprogramovej emulácii**.



## Riadiaca časť

Register adresy mikroinštrukcie



Transformačný obvod

## Zvyšovanie výkonnosti procesorov

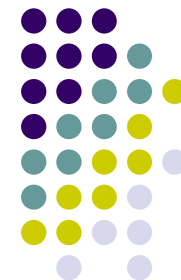
- **Zdokonalovanie technológie**

- Zmenšovanie fyzických rozmerov

- **Zdokonalovanie organizácie spracovania údajov**

- Pohyblivá rádová čiarka-numerický koprocessor
  - Predvýber a predspracovanie inštrukcií - rozdelenie výberovej a výkonnej fázy inštrukcií, jednotka predvýberu inštrukcií nevyužitie zbernice, rozdiel medzi rýchlosťou procesora a pamäte  $\Rightarrow$  predspracovanie inštrukcií, výber z pamäte, dekódovanie, výber operandov do Cache
    - Efektívne využitie zbernice
    - Problém skoky, dva prúdy inštrukcií





- Prúdové spracovanie inštrukcií – jednotlivé fázy (IF, D, OA ...S) sa realizujú nezávislou funkčnou jednotkou)

IF, D, OA, OF, EX, S

IF, D, OA, OF, EX, S

IF, D, OA, OF, EX, S

Údajová (data) a riadiaca nezávislosť inštrukcií, hazardy, konflikty

- Paralelné spracovanie inštrukcií
  - Aj v SISD
  - údajová a riadiaca nezávislosť inštrukcií
  - Párovanie inštrukcií (Pentium)

- **Procesory**

- CISC
- RISC
- NISC

## Procesor s architektúrou CISC (Complex Instruction Set Computer)



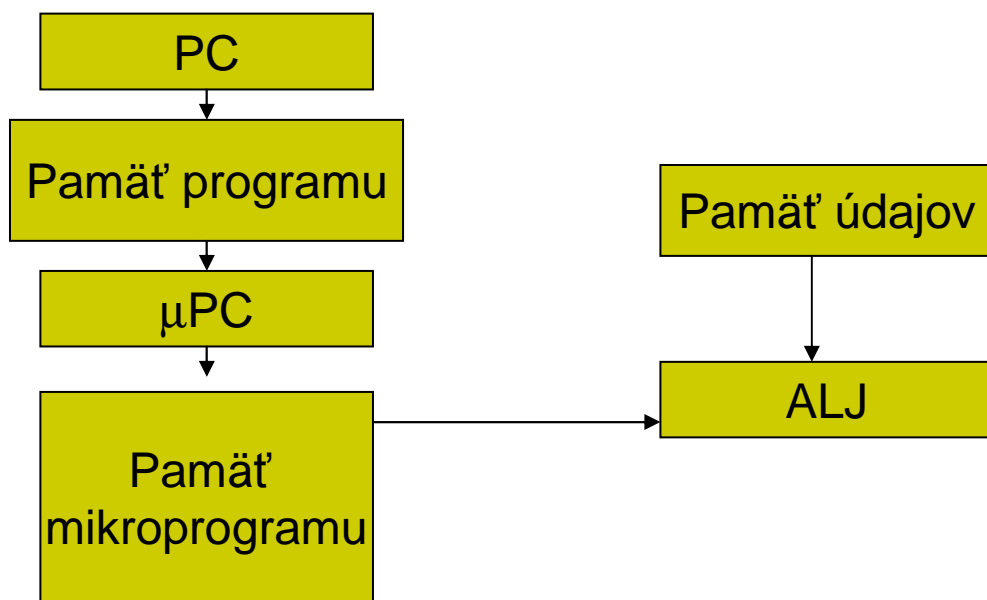
- *Zložitý inštrukčný súbor*-podporuje preklad z vyšších programovacích jazykov do strojového kódu procesora.
- *Veľa spôsobov adresácie operandov.*
- Inštrukcie realizujú aj *zložitejšie*, napr. reťazcové operácie,
- Implementovaná *priama podpora niektorých funkcií OS*
- Dodržiava sa *kompatibilita inštrukčného súboru v rodinách procesorov* zdola nahor atď.
- Zložitá *mikroprogramová riadiaca jednotka.*
- Zložité inštrukcie si vyžadujú pre svoju realizáciu veľký súbor *mikroinštrukcií* a podporných technických prostriedkov

## Procesor s architektúrou CISC (Complex Instruction Set Computer)

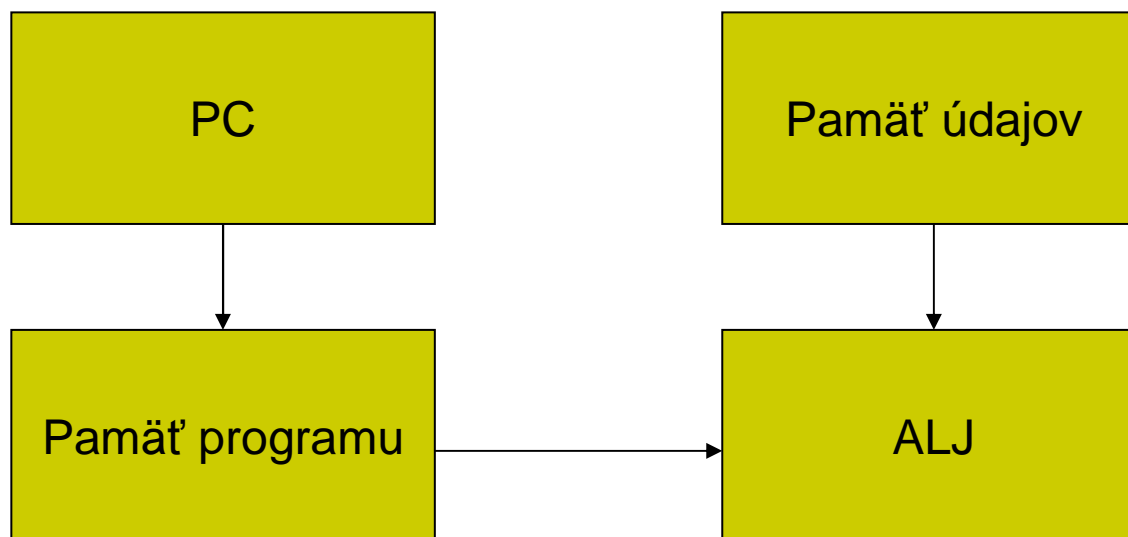


- Technické prostriedky na podporu mikroprogramovania zabierajú na čipe veľkú plochu, čím je daná aj vyššia cena
- Realizácia inštrukcií vzhľadom na použitie mikroprogramovej riadiacej jednotky trvá dlhšiu dobu.
- Iba malá podmnožina inštrukčného súboru tvorí väčšiu časť programov, pričom zložitejšie inštrukcie sa využívajú málo. Zložité inštrukcie dokonca môžu nepriaznivo ovplyvňovať optimalizáciu prekladu zložitejších jazykových konštrukcií vyšších programovacích jazykov.
- Predstaviteľmi procesorov CISC sú napr. procesory počítačov radu *IBM 43xx*, *VAX 11/780* alebo mikroprocesory rodiny *Intel 80x86*

# *Procesor s architektúrou RISC (Reduced Instruction Set Computer)*



# Procesor s architektúrou RISC vs. CISC







## *Procesor s architektúrou RISC (Reduced Instruction Set Computer)*

- Redukovaný inštrukčný súbor
- Inštrukcie sú jednoduché, ich vykonanie krátke (typicky v 1 strojovom cykle)
- Používa sa málo spôsobov adresácie operandov (obyčajne iba inštrukcie typu čítanie/zápis z/do hlavnej pamäte),
- Používa sa väčší počet univerzálnych registrov (desiatky až stovky).
- Riadiaca jednotka je jednoduchšia, obyčajne je to riadiaca jednotka s pevnou logikou.
- Na čipe sa uvoľnila značná časť plochy na implementáciu numerického koprocessora a vyrovnávacej pamäte Cache.
- Vykonanie inštrukcie, uloženej vo vyrovnávacej pamäti, je rýchlosťou porovnateľné s vykonaním mikroinštrukcie.
- Predstaviteľmi procesorov RISC sú napr. procesory SPARC, Motorola 88000, Transputer fy INMOS atď. (Apple)

# Procesor s architektúrou RISC vs. CISC



- Program, v ktorom sa nachádzajú aj zložitejšie inštrukcie, je síce pri procesore CISC kratší, ako pri procesore RISC (procesor RISC musí zložitejšie inštrukcie nahradiť postupnosťou svojich jednoduchých inštrukcií), ale vzhľadom na réžiu procesora CISC (musí inštrukciu dekomponovať na postupnosť mikroinštrukcií) je jeho vykonanie procesorom RISC rýchlejšie.

# *Procesor s architektúrou RISC vs. CISC*



- **RISC**

- JA Jump if Above
- JAE Jump if Above or Equal
- JB Jump if Below ...
- JPO Jump if Parity Odd
- JS Jump if Sign
- JZ Jump if Zero (príznakovom registri)

- **CISC**

- **Branch and Branch with Link.**
- BLEQ Branch with Link if Equal

NISC,

```
;Dokaze previest binarne,  
;hexadecimalne a decimalne cisla.  
;  
;Vstup: reg. BX obsahuje offset adresy  
;cisla v pamati.  
;  
;Vystup: reg. DX obsahuje cisel. hodnotu  
;-----
```

```
readnum    proc near  
            mov al,byte ptr [cs:bx]  
            inc bx  
            mov dx,0  
            mov cl,16  
            xor ch,ch  
            cmp al,"#"   
            jz readnum3  
            mov cl,2  
            xor ch,ch  
            cmp al,"%"   
            jz readnum3  
            xor ch,ch  
            mov cl,10  
            dec bx  
readnum3:  mov al,byte ptr [cs:bx]  
            sub al,"0"  
            cmp al,10  
            jc readnum4  
            sub al,"A"-"9"-1  
readnum4:  cmp al,16  
            jc readnum6
```

# RISC



```
# Compute first twelve Fibonacci numbers and put in array, then print
.data
fibs: .word 0 : 12      # "array" of 12 words to contain fib values
size: .word 12          # size of "array"
.text
la    $t0, fibs         # load address of array
la    $t5, size         # load address of size variable
lw    $t5, 0($t5)       # load array size
li    $t2, 1            # 1 is first and second Fib. number
add.d $f0, $f2, $f4
sw    $t2, 0($t0)       # F[0] = 1
sw    $t2, 4($t0)       # F[1] = F[0] = 1
addi  $t1, $t5, -2      # Counter for loop, will execute (size-2) times
loop: lw    $t3, 0($t0)  # Get value from array F[n]
      lw    $t4, 4($t0)  # Get value from array F[n+1]
      add   $t2, $t3, $t4 # $t2 = F[n] + F[n+1]
```

## **NISC**

Bez inštrukcií, bez dokódovania  
Iba riadiace slová 2-3 krát dlhšie ale každé  
vykonáva 2-3 inštrukcie RISC





- Rýchla odozva na externé udalosti
- Prerušenie po udalosti, keď je nevyhnutné začať vykonávať nový program tzv. obslužný program, prerušenia



# Prerušovací systém procesora

- Kasifikácia prerušení

- A Externé (z okolia procesora)

- B Interné (súvisia s vykonávaním inštrukcií)

- A Asynchrónne (externé, nesúvisí s vykonávanými inštrukciami, možno ho zakázať)

- B Synchronné (súvisí s programom, nemožno ho zakázať, skok do operačného systému, chyba pri vykonaní inštrukcie)

- Softvérové prerušenie (číslo prerušenia)

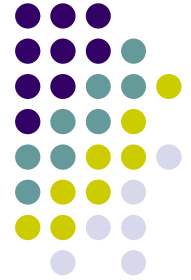
- Výnimka (exception, delenie nulou)

- A Maskovateľné (niektoré externé, spravidla asynchrónne)

- B Nemaskovateľné



# Prerušovací systém procesora



**Prerušenie** sa teda skladá z týchto krokov:

- prijatie požiadavky na prerušenie,  
(po vykonaní bežiacej inštrukcie)
- odloženie stavu procesora - PC, registre  
(zásobník)
- zistenie zdroja prerušenia, synchronne prerušenia majú pevné  
štartovacie adresy
- vykonanie zodpovedajúceho obslužného programu prerušenia,
- obnovenie pôvodného stavu procesora,
- pokračovanie v prerušenom programe.



*K bodu 1.*

Požiadavka na *externé* prerušenie môže prísť v ľubovoľnom okamihu, t.j. aj uprostred vykonávania inštrukcie. **S obsluhou prerušenia** (t.j. odloženie stavu procesora atď.) **sa však začne až po dokončení práve vykonávanej inštrukcie.**

*K bodu 2.*

**Okamžitý stav procesora** je charakterizovaný obsahom všetkých registrov procesora. *Stav procesora sa odkladá do zásobníka.* Použitie zásobníka umožňuje aj *hniezdenie prerušení*, to znamená, že počas obsluhy jedného prerušenia môže prísť k akceptovaniu ďalšieho prerušenia s *vyššou prioritou*.



**Asynchrónne prerušenie** je prerušenie, ktoré priamo nesúvisí s vykonávanými inštrukciami a môže nastať kedykoľvek. Je to tzv. *externé (hardvérové) prerušenie* a typicky je požadované niektorým vstupno/výstupným zariadením, keď je toto *pripravené na prenos*.

**Procesor má zvyčajne dva prerušovacie vstupy pre externé prerušenie:**

- **Vstup maskovateľného prerušenia.** Inštrukčný súbor procesora obsahuje v tomto prípade špeciálne inštrukcie, ktoré umožňujú *povoliť* resp. *zakázať* prijatie požiadavky z tohto vstupu.
- **Vstup nemaskovateľného prerušenia.** Toto prerušenie nie je možné zakázať a typicky sa používa pri obsluhu katastrofických situácií (napr. *výpadok napájacieho napätia*).

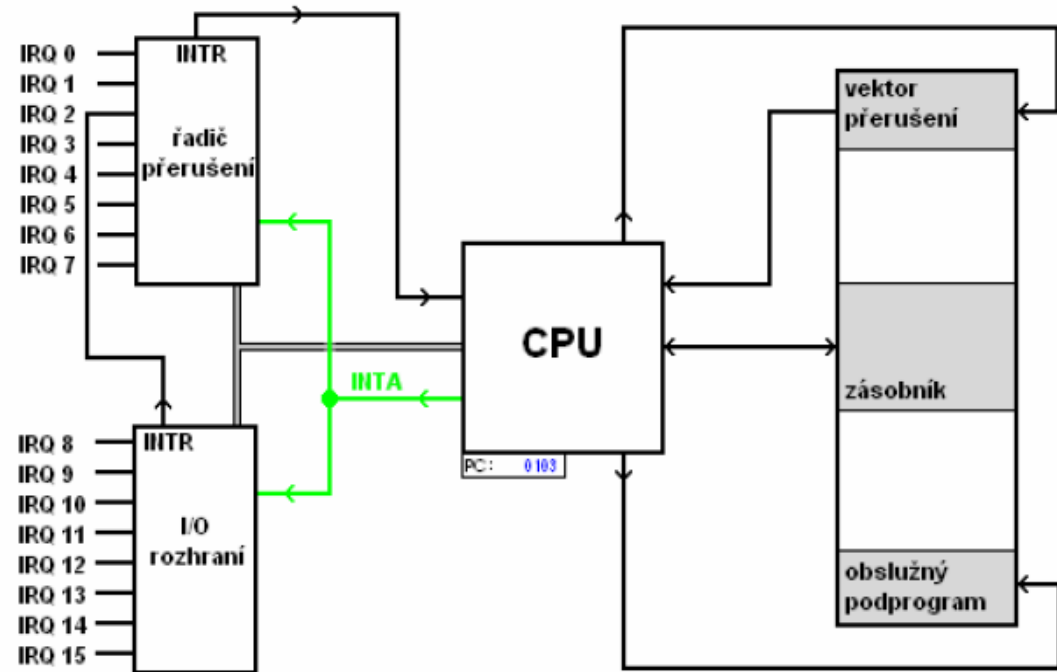


## Synchrónne prerušenie

**Synchrónne (interné) prerušenie** priamo súvisí s vykonávanými inštrukciami a nie je ho možné zakázať. Synchrónne prerušenie je dvojaké:

- **Softvérové prerušenie.** Softvérové prerušenie je generované po vykonaní špeciálnej *riadiacej inštrukcie*. Parametrom tejto inštrukcie je *číslo prerušenia*, ktoré sa má obslúžiť. Toto prerušenie sa typicky používa pri *volaní funkcií operačného systému*.
- **Výnimka (Exception).** Výnimka sa generuje automaticky, ak nastane nejaká chyba pri vykonaní inštrukcie (napr. *nedefinovaný operačný kód, delenie nulou, pokus o zápis do oblasti, kde sú uložené inštrukcie, sprístupňovaný segment sa nenachádza v hlavnej pamäti* atď.).

1. **Průběh hardwarového přerušení**
2. Vnější zařízení vyvolá požadavek o přerušení
3. I/O rozhraní vyšle signál IRQ na řadič přerušení (na port IRQ 2)
4. Řadič přerušení vygeneruje signál INTR – „někdo“ žádá o přerušení a vyšle ho k procesoru.
5. Procesor se na základě maskování rozhodne obsloužit přerušení a signálem INTA se zeptá, jaké zařízení žádá o přerušení.
6. Řadič přerušení identifikuje zařízení, které žádá o přerušení a odešle číslo typu přerušení k procesoru
7. Procesor uloží stavové informace o právě zpracovávaném programu do zásobníku.
8. Podle čísla typu příchozího přerušení nalezne ve vektoru přerušení adresu příslušného obslužného podprogramu.
9. Vyhledá obslužný podprogram obsluhy přerušení v paměti a vykoná ho.
10. Po provedení obslužného programu opět obnoví uložené stavové informace ze zásobníku a přerušený program pokračuje dál.



[http://www.dnp.fmph.uniba.sk/~kollar/pc\\_hw\\_sw/pc7.htm](http://www.dnp.fmph.uniba.sk/~kollar/pc_hw_sw/pc7.htm)

<http://cs.wikipedia.org/wiki/P%C5%99eru%C5%A1en%C3%AD>

# Virtuálna pamäť



- je taká správa pamäti, pri ktorej sa každému procesu dáva k dispozícii vlastná pamäť, ktorá má inú veľkosť alebo iný spôsob adresovania ako je fyzická pamäť
- Historicky vznikla hlavne z potreby vykonávať programy, ktoré sú väčšie než fyzická pamäť počítača
- Niektoré stránky virtuálnej pamäti sú vo fyzickej pamäti, iné sú odložené na disku



# Správa a ochrana hlavnej pamäte

- **MMU – memory management unit jednotka správy pamäte**
- **FAP fyzický adresový priestor-priradený pamäti**
- **LAP priradený programu**
- Ideálne  $LAP = FAP$ , nie je realita  $\Rightarrow$  správa pamäte
  - Program potrebuje  $LAP > FAP \Rightarrow$  vytvorenie virtuálnej pamäte na pevnom disku
  - Rozdelenie pamäte na viacero programov dynamická relokácia premiestňovanie LA na rôzne FA
  - Ochrana FAP – prístupové práva

# Zavedenie *správy pamäte* má tri hlavné príčiny:



- Program môže vyžadovať pre svoje vykonanie väčší logický adresový priestor, ako je fyzický adresový priestor, ktorý má počítač k dispozícii.
- Vyžaduje sa priradenie a rozdelenie fyzického adresového priestoru viacerým používateľským programom súčasne.
- Požaduje sa ochrana fyzického adresového priestoru, priradeného jednému používateľskému programu, pred ovplyvňovaním iným používateľskými programom.



## Segmentovanie

**Deskriptor segmentu** je záznam, ktorý tieto informácie o segmente:



- **Bázová adresa segmentu (Base).** Je to adresa, od ktorej je segment uložený v hlavnej pamäti
- **Veľkosť segmentu (Limit).** Pre každú inštrukciu alebo údaj v danom segmente musí platiť, že  $Offset < Limit$ .
- **Atribúty segmentu (Attributes).** Patrí sem:
  - Informácia o prítomnosti segmentu v hlavnej pamäti. Ak sa sprístupňovaný segment nenachádza v hlavnej pamäti, použije sa položka 4. a segment sa načíta z vonkajšej pamäte.
  - Informácia o type segmentu (vykonateľný segment, vykonateľný segment s možnosťou čítania, údajový segment len pre čítanie, údajový segment pre čítanie i zápis, zásobníkový segment atď.).
  - Informácia o privilegovanej úrovni segmentu (má význam v prípade, ak viacero programov môže používať ten istý segment)
  - **Adresa segmentu vo vonkajšej pamäti (External).**



- **Logická adresa** sa v prípade *segmentovania* skladá z dvoch častí, ktoré sa spracúvajú samostatne:

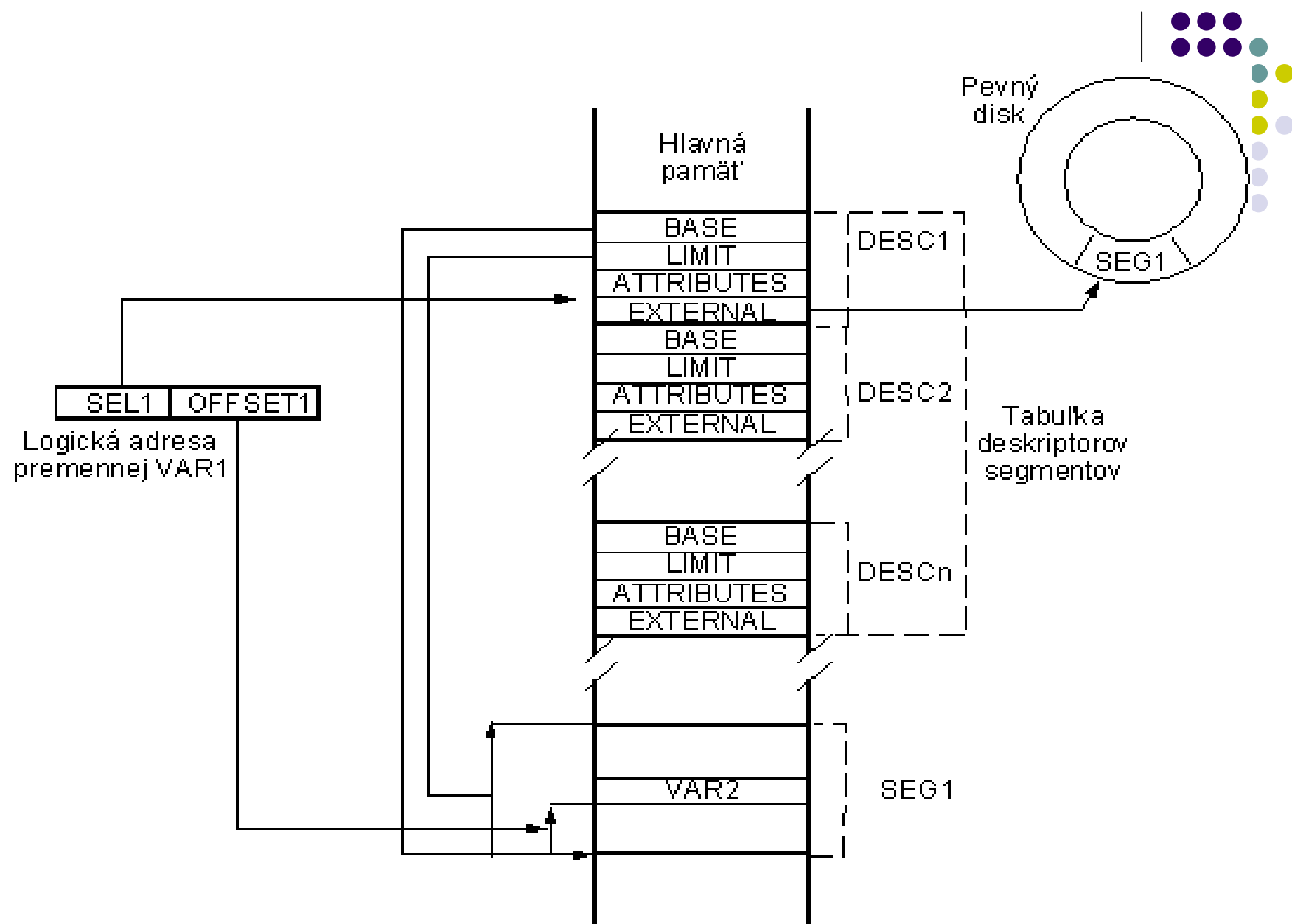
**Selektor: Posunutie**

- **Selektor** je ukazovateľ do *tabuľky deskriptorov segmentov*. Selektor sa nachádza v špeciálnom registri a manipuluje sa s ním nezávisle od posunutia. **Tabuľka deskriptorov segmentov** sa nachádza v hlavnej pamäti a sú v nej *deskriptory* všetkých segmentov.
- **Posunutie** (*offset*) reprezentuje *vzdialenosť* inštrukcie alebo údajov od začiatku segmentu. Iba *posunutie* vystupuje ako parameter v *adresovej časti* inštrukcií.

- Premenná *VAR1* patrí do segmentu *SEG1* a má logickú adresu *SEL1:OFFSET1*. Segment *SEG1* je opísaný deskriptorom *DESC1*. **Fyzická adresa** sa vypočíta tak, že k hodnote *bázy* sa pripočíta hodnota *posunutia*.



- Výhoda-segment možno presunúť na od ľub. adresy v pamäti, stačí zmeniť v deskriptore hodnotu bázy
- Externá fragmentácia



## Stránkovanie

- Pri **stránkovaní** je hlavná pamäť rozdelená na úseky rovnakej dĺžky, ktoré sa nazývajú **stránkové rámy** (*page frames*). Stránkové rámy sú očíslované. Program i údaje sú takisto rozdelené na úseky *rovnakej dĺžky* - **stránky**. Veľkosť stránky je rovnaká, ako veľkosť stránkového rámu.
- **Logická adresa** sa v prípade *stránkovania* skladá z dvoch častí, ktoré sú však na rozdiel od segmentovania spracovávané spoločne - *stránka* a *posunutie*.
- **Posunutie** (*offset*) reprezentuje *vzdialenosť* inštrukcie alebo údajov od *začiatku* stránky.
- **Stránka** (*page*) je ukazovateľ do *tabuľky deskriptorov stránok*.



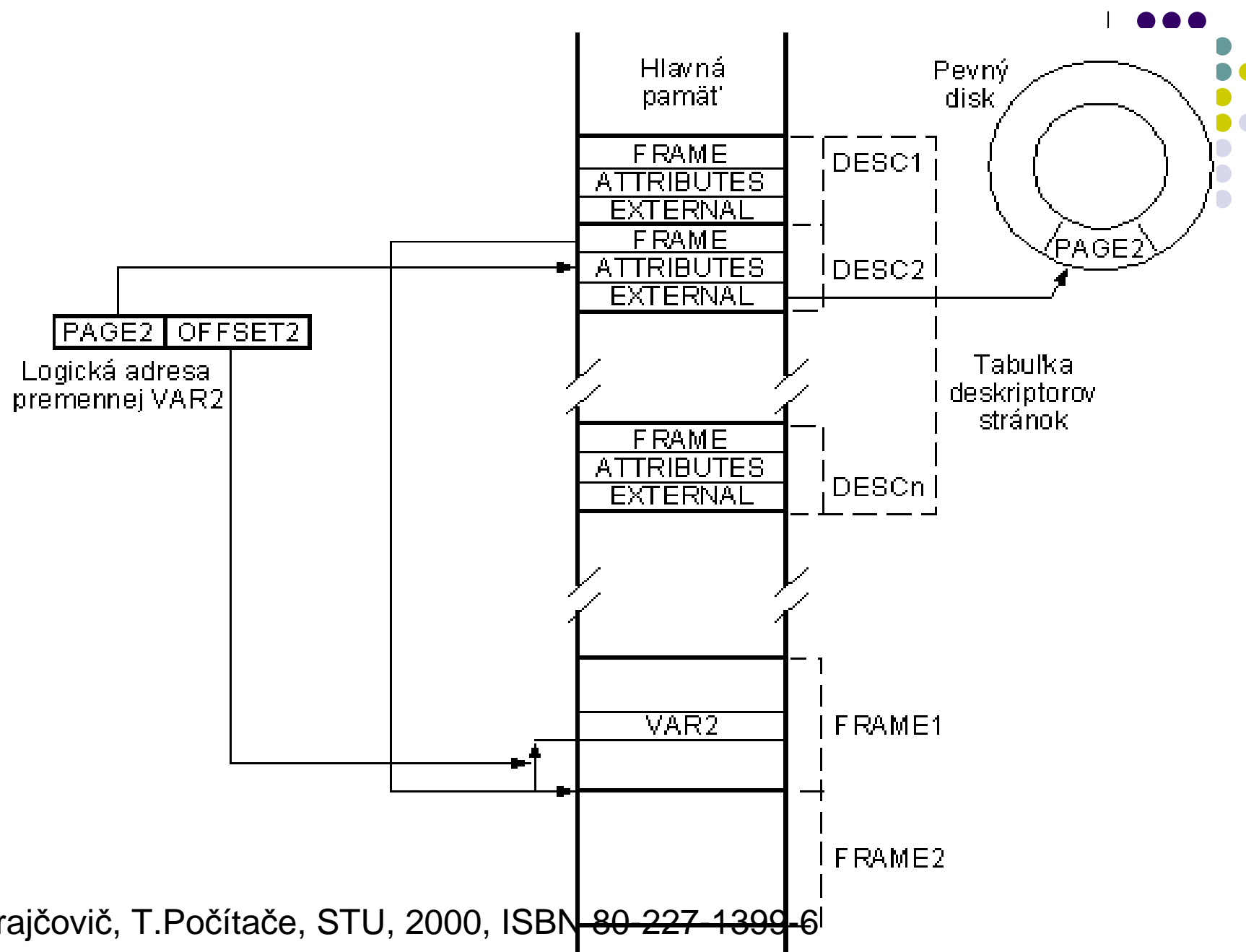


**Tabuľka deskriptorov stránok** sa nachádza v hlavnej pamäti a sú v nej deskriptory všetkých stránok.

**Deskriptor stránky** je záznam, ktorý obsahuje tieto informácie o stránke:

- **Číslo stránkového rámu (Frame)** - všetky stránkové rámy majú rovnakú veľkosť, číslo stránkového rámu jednoznačne určuje adresu, od ktorej je stránka v hlavnej pamäti uložená (*začiatok stránky*).
- **Atribúty stránky (Attributes)**. Tieto sú rovnaké, ako atribúty segmentu.
- **Adresa stránky vo vonkajšej pamäti (External)**.

! Pretože všetky stránky majú rovnakú veľkosť, deskriptor stránky informáciu o veľkosti stránky *nemusí obsahovať*.

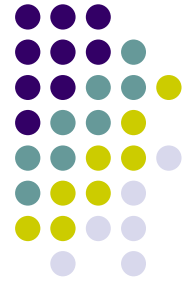




- Veľkosť programov alebo údajov nie je vo všeobecnosti celočíselným násobkom veľkosti stránky  $\Rightarrow$  **interná fragmentácia**
  - Posledná stránka potom nie je plne využitá. Čím viac programov alebo údajových štruktúr sa v pamäti nachádza, tým viac sa prejavuje nevyužitie hlavnej pamäte.
  - Čím je väčšia veľkosť stránky, tým je problém vypuklejší. Na druhej strane, ak zmenšíme veľkosť stránky, výrazne rastie tabuľka deskriptorov.  $\Rightarrow$  kompromis.
- 
- Okrem jednoduchého stránkovania sa používa aj *viacúrovňové stránkovanie*, prípadne sa kombinuje *segmentovanie a stránkovanie*. Vtedy hovoríme o **stránkovaných segmentoch**.



Komplexné riešenie ochrany pamäte si vyžaduje spoluprácu technických a programových prostriedkov počítača.



### !!! Kontrola

- či sa sprístupňovaný segment *nachádza v hlavnej pamäti*,
- či *adresa* sprístupňovanej inštrukcie alebo údajov *nepresiahla limit segmentu*,
- či je *použitie daného typu segmentu korektné* (napr. či *nedochádza k výberu inštrukcie z údajového segmentu*)
- či je *oprávnené použitie daného segmentu* (žiadateľ má *dostatočnú privilegovanú úroveň*).

$$A + B = B + A$$

$$A \cdot B = B \cdot A$$

$$A + (B + C) = (A + B) + C$$

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

$$A \cdot (B + C) = A \cdot B + A \cdot C$$

$$A + B \cdot C = (A + B) \cdot (A + C)$$

$$A + A = A$$

$$A \cdot A = A$$

$$A + 0 = A$$

$$A \cdot 0 = 0$$

$$A + 1 = 1$$

$$A \cdot 1 = A$$

$$A + \bar{A} = 1$$

$$A \cdot \bar{A} = 0$$

$$A = \bar{\bar{A}}$$

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

$$A \cdot (A + B) = A$$

$$A + A \cdot B = A$$

$$A + \bar{A} \cdot B = A + B$$

$$A \cdot (\bar{A} + B) = A \cdot B$$

$$\bar{A} + A \cdot B = \bar{A} + B$$

$$\bar{A} \cdot (A + B) = \bar{A} \cdot B$$



1. $x + y = y + x$	$x \cdot y = y \cdot x$	komutatívne zákony
2. $(x + y) + z = x + (y + z)$	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	asociatívne zákony
3. $(x + y) \cdot z = (x \cdot z) + (y \cdot z)$	$(x \cdot y) + z = (x + z) \cdot (y + z)$	distributívne zákony
4. $\overline{x + y} = \overline{x} \cdot \overline{y}$	$\overline{x \cdot y} = \overline{x} + \overline{y}$	de Morganove zákony
5. $x + x = x$	$x \cdot x = x$	zákony idempotentnosti
6. $x + \overline{x} = 1$	$x \cdot \overline{x} = 0$	zákony komplementárnosti
7. $\overline{\overline{x}} = x$		zákon involúcie
8. $x + (x \cdot y) = x$	$x \cdot (x + y) = x$	zákony absorpcie
9. $x + 0 = x$	$x \cdot 1 = x$	zákony identity
10. $x + 1 = 1$	$x \cdot 0 = 0$	zákon jednotkového sčítovania a nulového násobenia

## Opakovanie



Hovoríme, že množina logických spojok  $S$  je úplný systém logických spojok (skrátene USLS), ak pre každú formulu  $a$  existuje formula  $b$ , ktorá obsahuje iba logické spojky z množiny  $S$  a platí  $a \Leftrightarrow b$ . Vtedy tiež hovoríme, že formula  $a$  sa dá vyjadriť pomocou  $S$ .

napr.

1.  $+$   $.$   $^-$
2.  $\uparrow$
3.  $\downarrow$

Množina  $\{f_1, f_2, \dots, f_k\}$  booleovských funkcií sa nazýva úplný systém booleovských funkcií (stručne USBF), ak každá booleovská funkcia  $f$  sa dá vyjadriť ako zložená funkcia z funkcií  $f_1, f_2, \dots, f_k$ .

## Priamy kód

V priamom kóde sa najvyšší bit používa ako *znamienkový bit*. Kladné čísla sa zobrazujú rovnako, ako prirodzené čísla, záporné číslo sa líši od kladného tým, že znamienkový bit má jednotkový.

Nevýhodou je, že číslo 0 má dva obrazy.

Rozsah zobrazenia je pre n-bitový register  $\langle -2^{n-1}, 2^{n-1}-1 \rangle$ .

Napr. Pre 16 bitovové číslo (register)

+6      0000 0000 0000 0110

- 6      1000 0000 0000 0110

-----  
+6      0000 0000 0000 0110

- 6      1111 1111 1111 1001

## Inverzný kód

V inverznom kóde má najvyšší bit opäť význam znamienka, ale na rozdiel od priameho kódu vstupuje do operácie. Kladné Čísla sa zobrazujú rovnako, ako prirodzené čísla. Záporné číslo sa získa takým spôsobom, že kladné číslo s rovnakou absolútnou hodnotou sa *invertuje* bit po bite.

Rozsah zobrazenia je rovnaký, ako v priamom kóde. Číslo 0 má opäť dva obrazy.

## Doplnkový kód

V *doplnkovom kóde*, tak ako v inverznom kóde, má najvyšší bit význam znamienka a tiež vstupuje do operácie. Kladné čísla sa zobrazujú rovnako, ako **prírodné** čísla. Záporné číslo v doplnkovom kóde získame zo záporného čísla v inverznom kóde tak, že k nemu pripočítame jedničku v najnižšom ráde. Výsledkom je vlastne doplnok absolútnej hodnoty záporného čísla do čísla 2".

+6	0000 0000 0000 0110
- 6	1111 1111 1111 1010

Rozsah zobrazenia pre  $n$ -bitový register:  $\langle -2^{n-1}, 2^{n-1}-1 \rangle$

1111 1111 1111 1001
1
1111 1111 1111 1010

Číslo v pohyblivej rádovej čiarke je zobrazené v tvare:  $M \cdot z^E$

kde M je mantisa (obyčajne pravý zlomok),

z - základ použitej číselnej sústavy (obyčajne 2),

E - exponent (celé číslo).



Desiatkové čísla predstavujú špeciálnu triedu celých čísel a často sa spracúvajú priamo, bez prevodu do dvojkovej číselnej sústavy. Z tohto dôvodu sa používajú aj špeciálne kódy na ich zobrazenie, a to *BCD (Binary Coded Decimal) kód* a *zhustený BCD kód*.

Pri zobrazení v *štandardnom BCD kóde* sa do registra dĺžky jednej slabiky (8 bitov) zobrazí jedna desiatková číslica. Horné štyri bity sú nulové, v dolných štyroch bitoch je zakódovaná jedna číslica desiatkového čísla v prirodzenom dvojkovom kóde.

**Príklad:** Zobrazme číslo 1463 v BCD kóde.

Je nutné použiť štyri slabiky:

00000001 00000100 00000110 00000011

Pri zobrazení v *zhustenom BCD kóde* sú v jednej slabike zakódované dve desiatkové číslice.

**Příklad:** Zobrazme číslo *1463* v zhustenom BCD kóde.

Stačia nám dve slabiky:

00010100 01100011

Ak sa používajú aj záporné čísla, na znamienko je vyhradená obyčajne celá jedna slabika.