(Mikro)Procesor

CPU – Central Process (Processing) Unit

Hlavná vykonávacia jednotka
(spravidla realizovaný mikroprocesorom)

CP – Central Processor

(označenie v superpočítačoch
Superpočítač =počítač s minimálne
1000-násobným výkonom ako PC)

Procesor - časť počítača určená na vykonávanie programu. Je to logický obvod, ktorý dokáže spracovávať sadu **inštrukcií**.

Inštrukcia – z hľadiska programátora úkon vykonateľný procesorom. Legitímna snaha je, aby bol čo najmenší t.j. elementárny Hlavne v multimédiách tá snaha nevedie k výsledkom a tak vykonávajú inštrukcie aj komplexné úkony

Kód inštrukcie – binárna reprezentácia inštrukcie uložená v pamäti.

Skladá sa obyčajne z :

- **kódu operácie**
- a **kódu operandu** (informácia, kde sa nachádza operand)

Operand – pamäťové miesto, ktorého obsah sa zúčastňuje operácie

(inštrukcie bez operandu, s jedným a dvomi operandmi)
V prípade operandu v registri býva taký operand zakódovaný v kóde inštrukcie

Program – postupnosť inštrukcií uložených v pamäti často v určitom poradí, ktoré nejako udáva požiadavky na poradie ich vykonania..

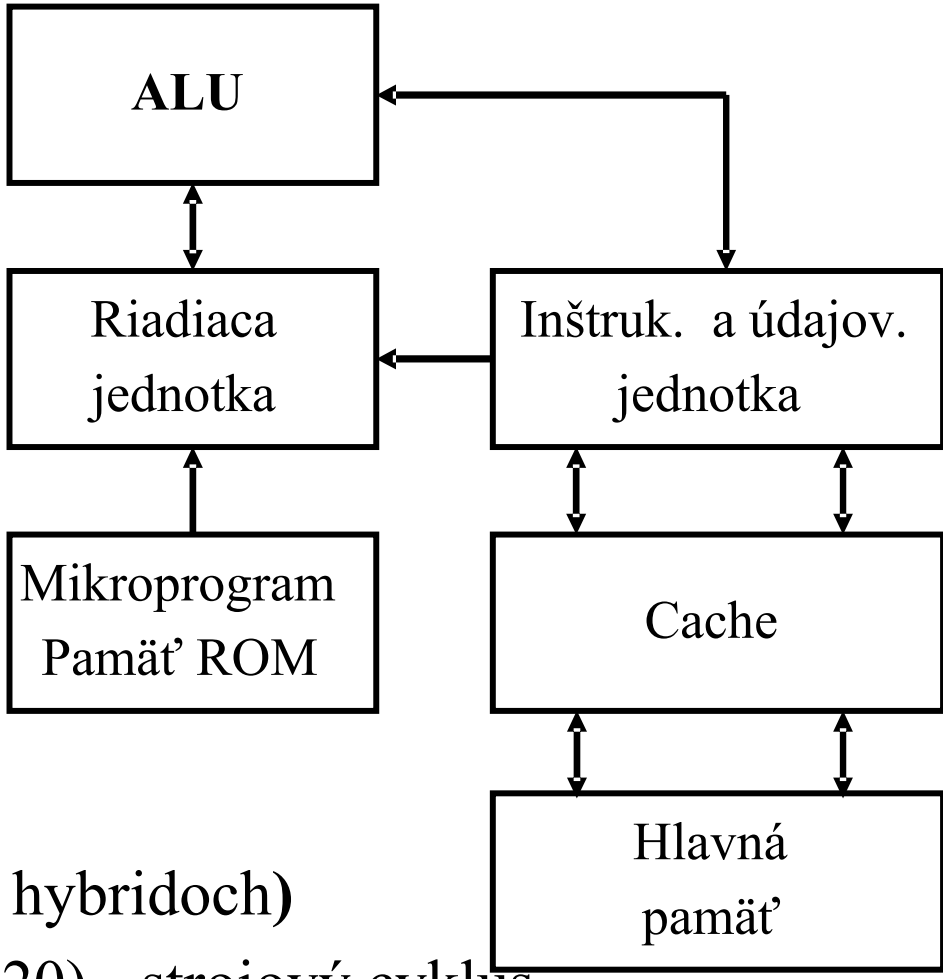
ZÁKLADNÉ KONCEPCIE MIKROPROCESOROV

Complex Instruction Set Computer (**CISC**) Procesor má „úplnú“ sadu inštrukcií. Cieľom vývoja týchto procesorov bol stav, kedy jazyk vyššej úrovne bude priamo strojovým jazykom daného procesora. Procesor CISC => jednoduchý návrh prekladačov. Jazykom je low level C, bez knižníc, hlavičkových súborov atď. „ C Is The New Assembly 14.2, 2007 I am reminded of the old chestnut that C is a language that combines all the elegance and power of assembly language with all the readability and maintainability of higer level languages.“ Tak bolo C navrhnuté http://www.red-sweater.com/blog/278/c-is-the-new-assembly

CISC procesory na konci 70 rokov boli už tak zložité, že sa objavili prvé pokusy o celkové zjednodušenie štruktúry procesorov. Vznikla tak celá nová kategória počítačov, dnes označovaná ako RISC.

Reduced Instruction Set Computer (**RISC**) Vychádza z predpokladu, že na vykonanie cca 80% operácií programu postačuje cca 20 % inštrukcií. Procesor má zabudované len základné (mikro)inštrukcie, ktoré sú jednoduchšie a ľahšie – rýchlejšie vykonateľné. Inštrukcie, ktoré nie sú v základnom inštrukčnom súbore sa jednoducho naprogramujú. Výhoda procesorov RISC, sú rýchlejšie ako CISC, paradoxne zvyšuje cenu počítača. RISC procesor predpokladá rýchlejšie zbernice, pamäte, atď.

Procesory CISC



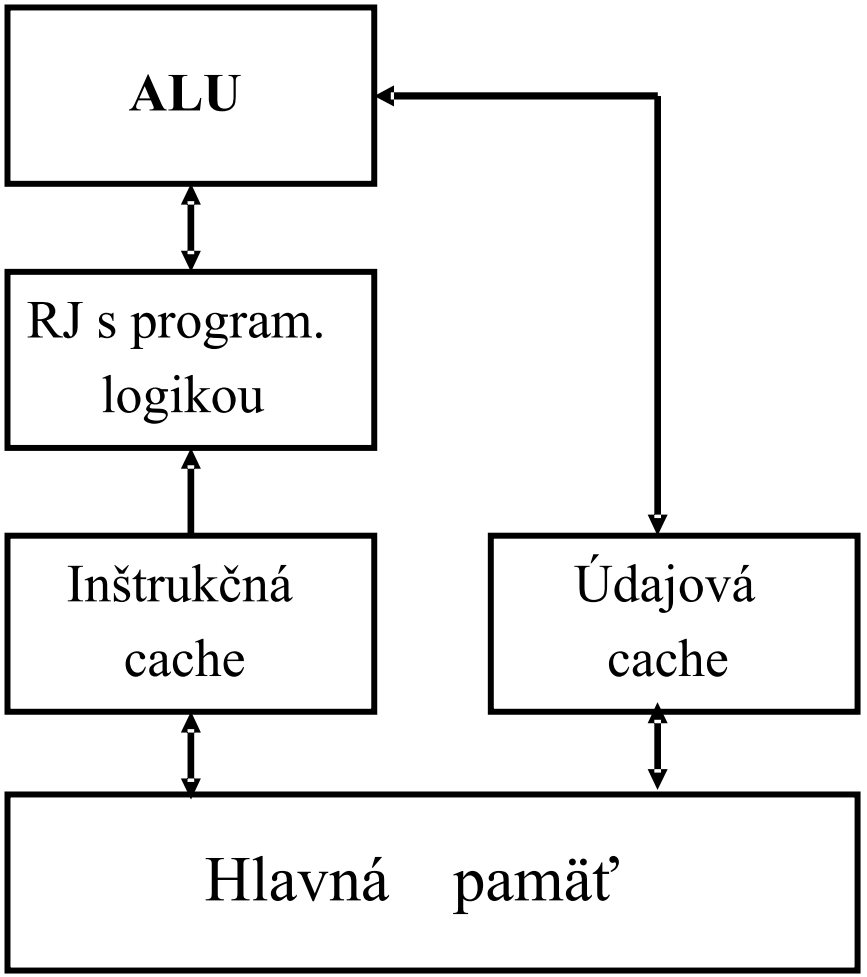
Charakteristika

- veľa zložitých inštrukcií (120 – 350+)
- malý počet vnútorných registrov
- inštrukcie s premenlivou dĺžkou
- veľký počet formátov inštrukcií (> 8)
- zložité, časovo náročné dekódovanie
- veľa adresovacích režimov (viac ako 12)
- **riadenie vykonávania inštrukcie pomocou mikroprogramu** (v RISC-CISC hybridoch)
- vykonanie inštrukcie na viacero SC (2 až 20) - strojový cyklus
- možnosť doplnenia používateľských inštrukcií v pamäti mikroprogramu - aj oprava chýb v obvode pomocou mikroprogramu

Možnosti zvyšovania výkonnosti

- prúdové spracovanie operandov
- prúdové spracovanie inštrukcií – s predvýberom inštrukcií a dát (IF)

Procesory RISC



Celkový čas vykonávania programu

T_c možno vyjadriť vzťahom:

$$T_c = T_{SC} \sum_{i=1}^{N} C_i$$

Kde:

N – počet inštrukcií programu

$\boldsymbol{T_{SC}}$ – čas trvania jedného SC

C_i – relatívny čas trvania i-tej inštrukcie

RISC svojou podstatou zvyšuje N

C_i a T_{SC} sú pre procesory RISC kratšie

Ako pri CISC. RISC môže mať vyššiu Frekvenciu a má vyššiu efektivitu

Požiadavka je

$$T_{SC} \leq 1$$

Charakteristika RISC

- redukovaný súbor inštrukcií (cca 100 a menej)
- inštrukcie s konštantnou dĺžkou
- malý počet formátov (menej ako 8) jednoduchšie dekódovanie
- malý počet adresovacích módov (3 až 5), ale pre čítanie a zápis do pamäte len dve inštrukcie (Load, Store), ostatné inštrukcie sú registrové
- **riadenie vykonávania inštrukcie pomocou pevnej logiky**
- vykonanie inštrukcie spravidla v **1 SC**
- zložitejší kompilátor z vyšších programovacích jazykov
- veľký počet vnútorných registrov (zmenšuje potrebu zapisovať a čítať medzivýsledky z pamäte), umožňuje odovzdávanie parametrov do podprogramov logickým usporiadaním registrov do „okien“

Argumenty za CISC

- komplexnejšie inštrukcie, zmenšujú počet čítaní inštrukcie z pamäte
- jednoduchší kompilátor z vyšších programovacích jazykov
- obchodný a reklamný faktor (procesor vykonávajúci viacej typov inštrukcií $\Rightarrow$ dokonalejší)
- vygenerovaný kód programu je kratší

Argumenty za RISC

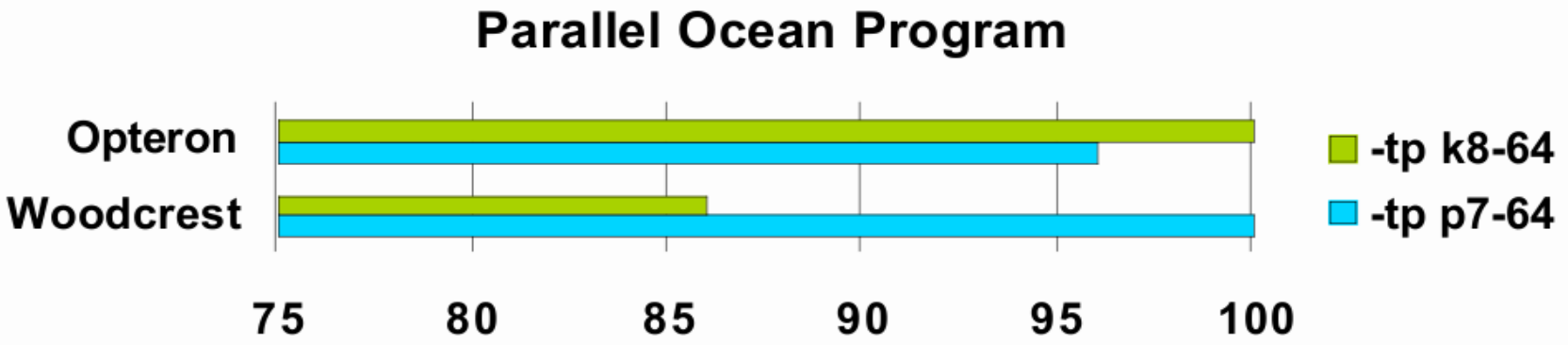
- na čipe procesora CISC zbytočne zaberá miesto pamäť mikroprogramu s príslušným zložitým riadením (pri hybride)
- vysoké percento inštrukcií CISC procesora nevie efektívne využiť kompilátor ani programátor – toolchain-y a výkony
- dynamické merania IBM ukázali, že
  - 50% času sa vykonávajú len 3 inštrukcie !!
  - 75% času 8 inštrukcií
  - inštrukcie, ktoré bežia 0,2% času zaberajú 60% pamäte mikroprogramu !!
- čas vývoja CISC až 5 rokov
- čas vývoja RISC 8- 20 mesiacov (uvedený RISC- prvý vyrobený čip fungoval)!!!!
- prúdové spracovanie inštrukcií (optimalizácia skokov, vyplňovanie bublín, odovzdávanie výsledkov ...)
- efektívny čas vykonania inštrukcie, pri špičkových RISC, menej ako 1 SC
- technologický a vývojový faktor

Kernelové programovanie- údaje z dnešného predmetu Operačné systémy reáalneho času

- **Vplyv nastavenia pri kompilácii (Portland Compiler, od 2013 patrí spol. nVidia)**

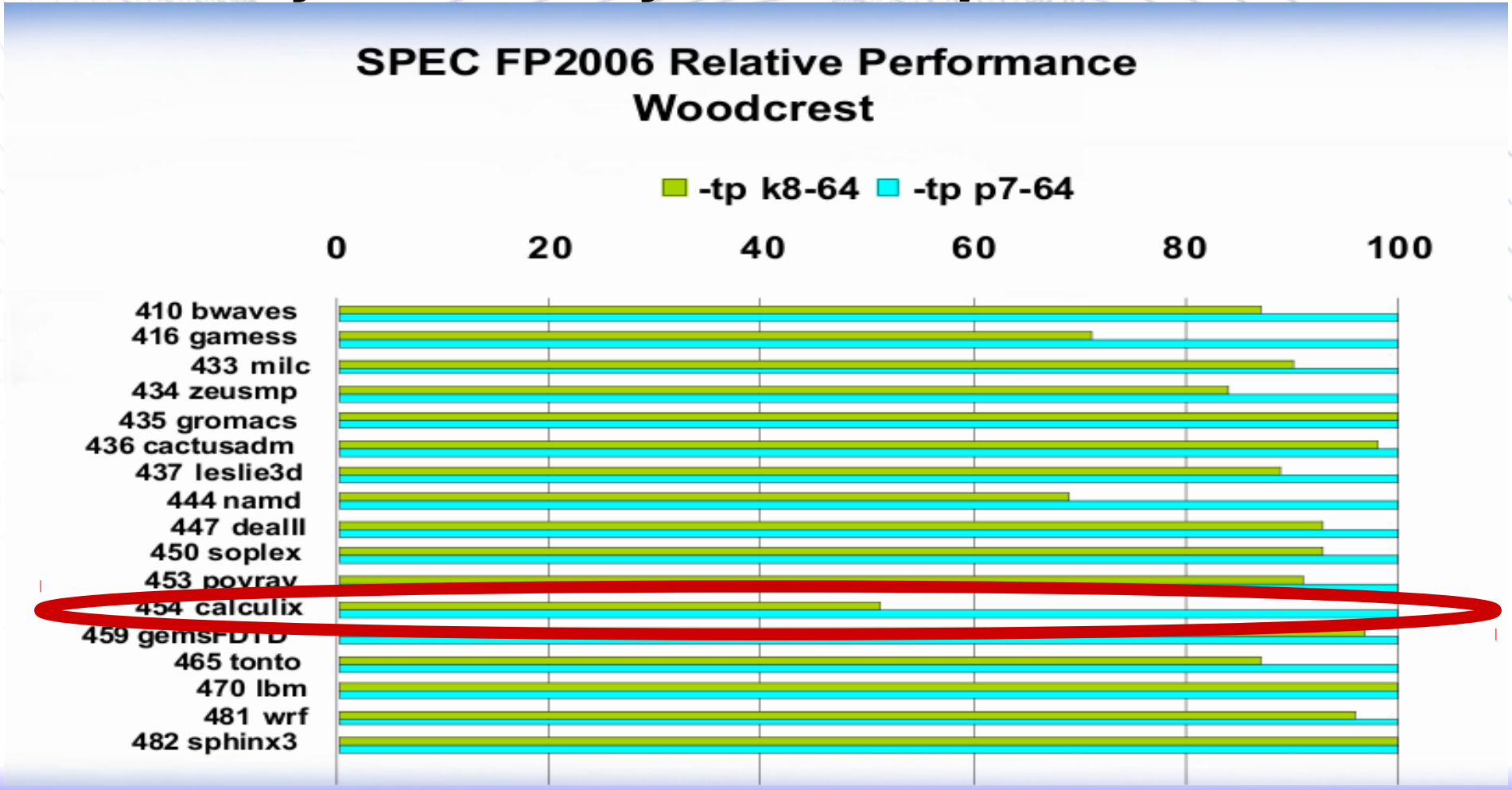
AMD vs. Intel
The Compiler as Referee

- `pgf95 -tp k8-64`
- `pgf95 -tp p7-64`



Kernelové programovanie

- **Vplyv nastavenia pri kompilácii (Portland Compiler) -Intel P4 -Woodcrest CPU**
  Nesprávne optimalizovaný kód dosahuje niekedy len 50% rýchlosti optimalizovaného



INŠTRUKČNÝ SÚBOR

Operácie, ktoré je schopné CPU vykonať (tvoria **inštrukčný súbor** procesora) môžeme rozdeliť na :

1. **operácie na presun údajov** (kopírovanie obsahu pamäťových miest registrov, pamäťových buniek a registrov I/O zariadení)

2. **operácie aritmeticko-logické** (zmena obsahu pamäťovej bunky)

3. **operácie riadenia toku programu** (skokové inštrukcie, inštrukcie vetvenia programu, volanie a návrat z podprogramu) – zmena obsahu programového počítadla

4. Výrobcovia procesorov doplňujú inštrukčnú sadu o inštrukcie pre prehrávanie videa, zvuku, grafiky.

5. Rôzne architektúry (ARM,x86,MIPS, SPARC, Power,…..) rôzne čísla pre tú istú inštrukciu, v rámci x86 rôzny výrobcovia zabúdajú rôzne staré Inštrukcie a ich kódy a nahrádzajú ich novými inštrukciami...

ZÁKLADNÉ BLOKY CPU

Na to aby CPU mohlo vykonávať program musí mať:

- **registre** na dočasné uchovávanie informácií (väčšina z nich
  Je prístupná programátorovi) (**akumulátor**, **registre pre všeobecné použitie, stavový register**)
- **programové počítadlo**,
- **inštrukčný register**,
- **smerník zásobníka**,
- **aritmeticko-logickú jednotku**,
- **riadiacu jednotku**,
- **jednotku na zápis a čítanie** informácie v pamäti a I/O priestore

Okrem týchto základných funkcií CPU by malo vedieť:

- dostať sa do definovaného stavu (reagovať na signál **RESET**)
- dostať sa do stavu, kedy nebude generovať signály na zbernicu (**HOLD**), tento stav je dôležitý pre realizácii **DMA** a pre súčinnosť s viacerými procesormi
- **vetviť program** (vykonávať inštrukcie mimo poradia určených programom) na základe **vonkajších signálov** (maskovateľné a nemaskovateľné prerušenia)
- **signalizovať** svoj **stav** na zbernicu
- v niektorých prípadoch vedieť prispôsobiť sa pomalším zariadeniam na zbernici a pod.

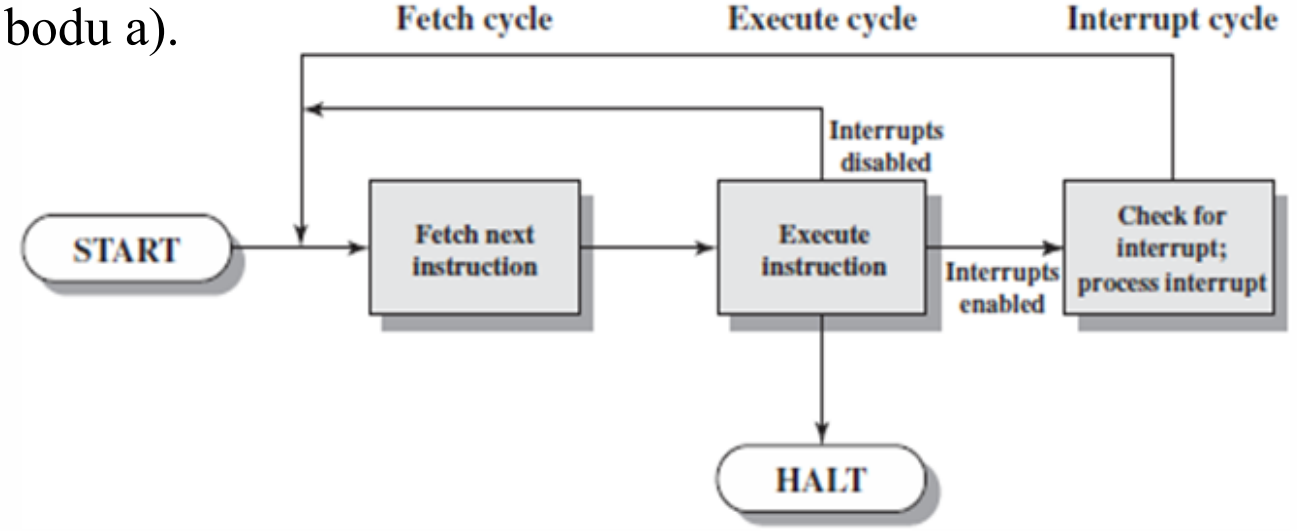
ZÁKLADNÁ FUNKCIA PROCESORA

je založená na **opakovaní základných činností**

a) **prečítanie inštrukcie** (adresa inštrukcie je uložená v registri CPU v **programovom počítadle PC**), ktoré pozostáva z:
   - z prečítania kódu operácie do **inštrukčného registra IR**
   - **dekódovania kódu operácie**, či k danému kódu netreba čítať ďalšie časti inštrukcie (**operand**), dokončiť čítanie inštrukcie

b) **riadiaca jednotka CPU** na základe kódu zabezpečí postupnosť **riadiacich signálov** tak, aby bola vykonaná požadovaná operácia. Výsledok operácie sa zvyčajne dočasne ukladá do vnútornej pamäte CPU – registrov CPU

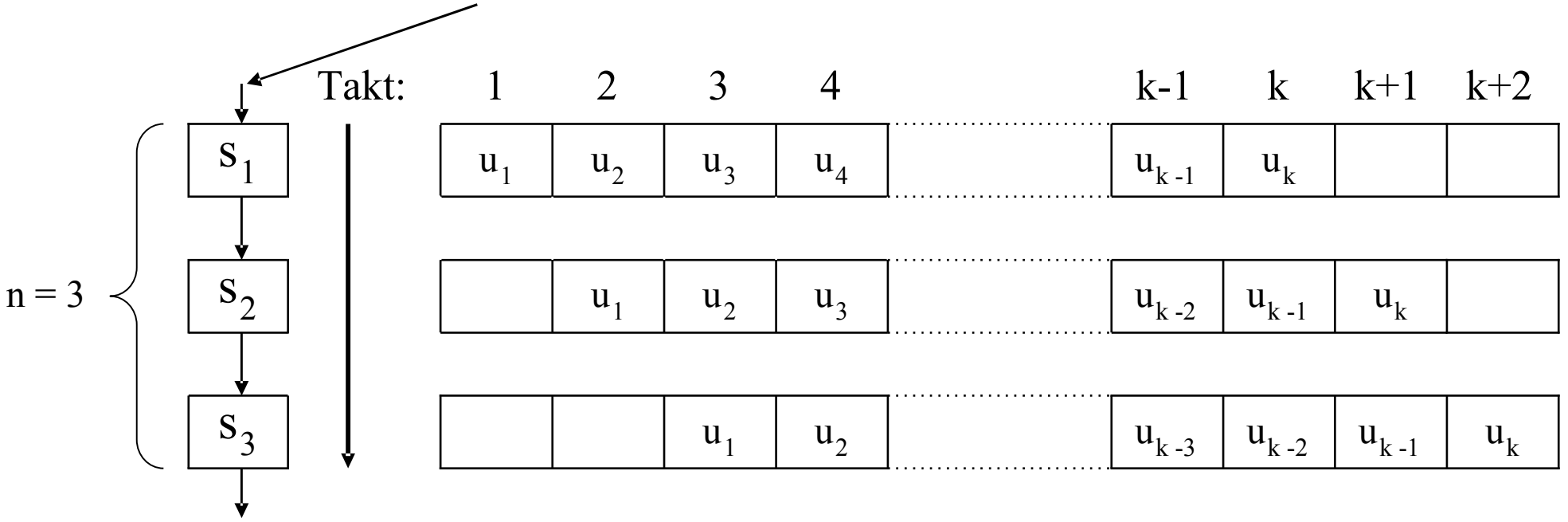
c) po vykonaní operácie resp. pred jej vykonaním riadiaca jednotka pripočíta k obsahu PC hodnotu, ktorá je rovná dĺžke inštrukcie v bytoch $\Rightarrow$ po vykonaní inštrukcie bude v PC **adresa nasledujúcej inštrukcie**, ktorá sa má vykonať, a činnosť prebieha znovu od bodu a).



Čas vykonania ***k*** úloh (nech fáza s_1 trvá T_1, s_2 trvá T_2 a s_3 trvá T_3), bude

$$T_C = k\left(T_1 + T_2 + T_3\right)$$

Prúdové spracovanie operandov (pipelining)



Čas vykonania K úloh bude v tomto prípade: $T_C = \left(k+2\right) T_{max}$

kde $T_{max} = max\left(T_1, T_2, T_3\right) + \Delta T$

ΔT je čas potrebný na zápis a čítanie výsledku mikroopeácie z vyrovnávacieho registra.

Napr.: $T_2 > T_1 = T_3$

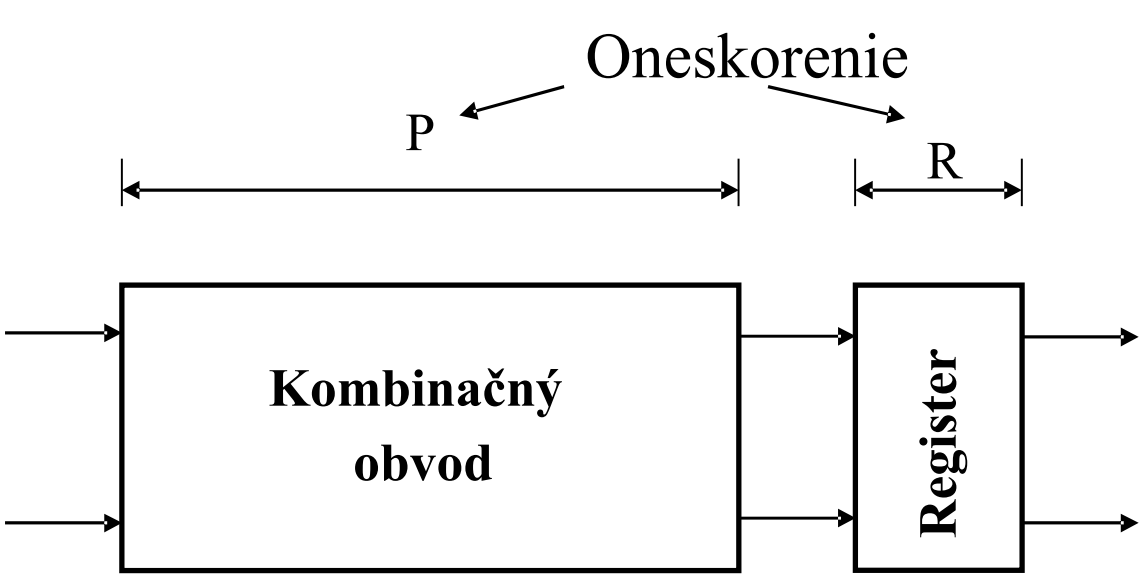
Pri určitom zjednodušení môže byť koeficient priepustnosti prúdovej jednotky $n+k-1$ oproti sekvenčne pracujúcej jednotke nk

$$q=\frac{nk}{n+k-1}\ pre\ k\gg n\ ,q\simeq n$$ kde ***n*** je počet sekcií

Podmienky realizácie prúdového spracovania :

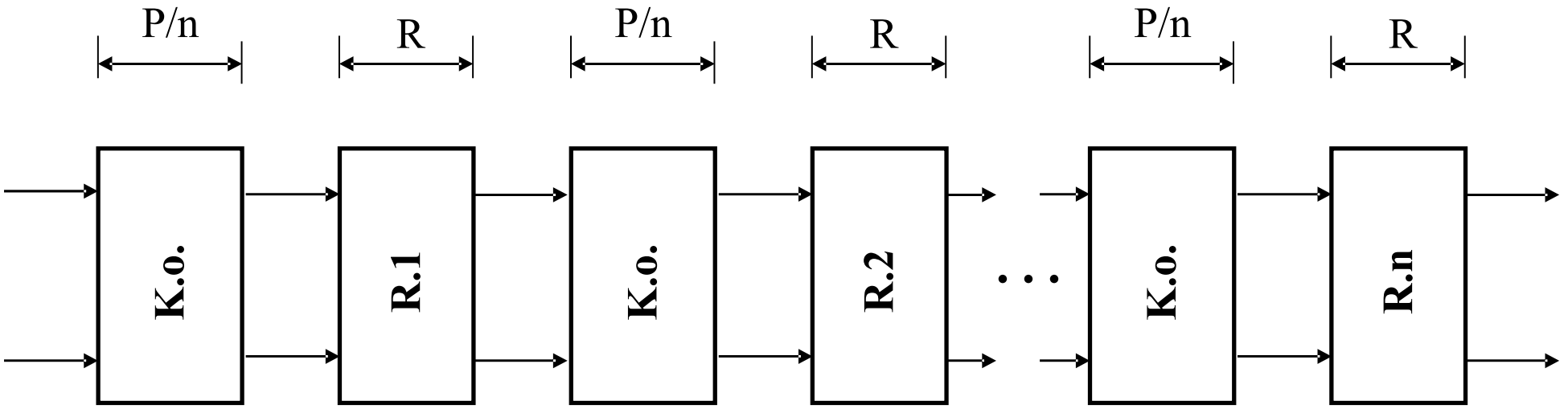
- činnosť sekcií musí byť **synchronizovaná**
- každá sekcia musí obsahovať **vyrovnávací register**, cez ktorý odovzdáva medzivýsledok ďalšej sekcii
- **čas vykonania** operácie v sekciách je **rovnaký** (čas vykonania operácie sa rovná času vykonania operácie v najpomalšej sekcii)

Realizácia jednej sekcie



Za čas P + R spracuje jednu úlohu.

Realizácia obvodu s *n* sekciami



Existuje optimálna (rozumná) voľba počtu sekcií ?

Odpoveď: Áno, ale je to *zložitý vzťah.*

PRÚDOVÉ SPRACOVANIE INŠTRUKCIÍ

Inštrukcia (strojová inštrukcia) je **najmenší programový element** vykonateľný procesorom.

Inštrukcia sa delí na strojové cykly, samostatné elementárne úkony vykonateľné jednotkami procesora, z ktorých každá trvá násobky **hodinových cyklov**.

Každú inštrukciu je možné formálne rozdeliť na dielčie elementárne úkony.

Elementárne úkony pri vykonaní inštrukcie :

Typ inštrukcie: aritmetická alebo logická

Operandy dva: - 1. Register procesora

- 2. Hlavná pamäť

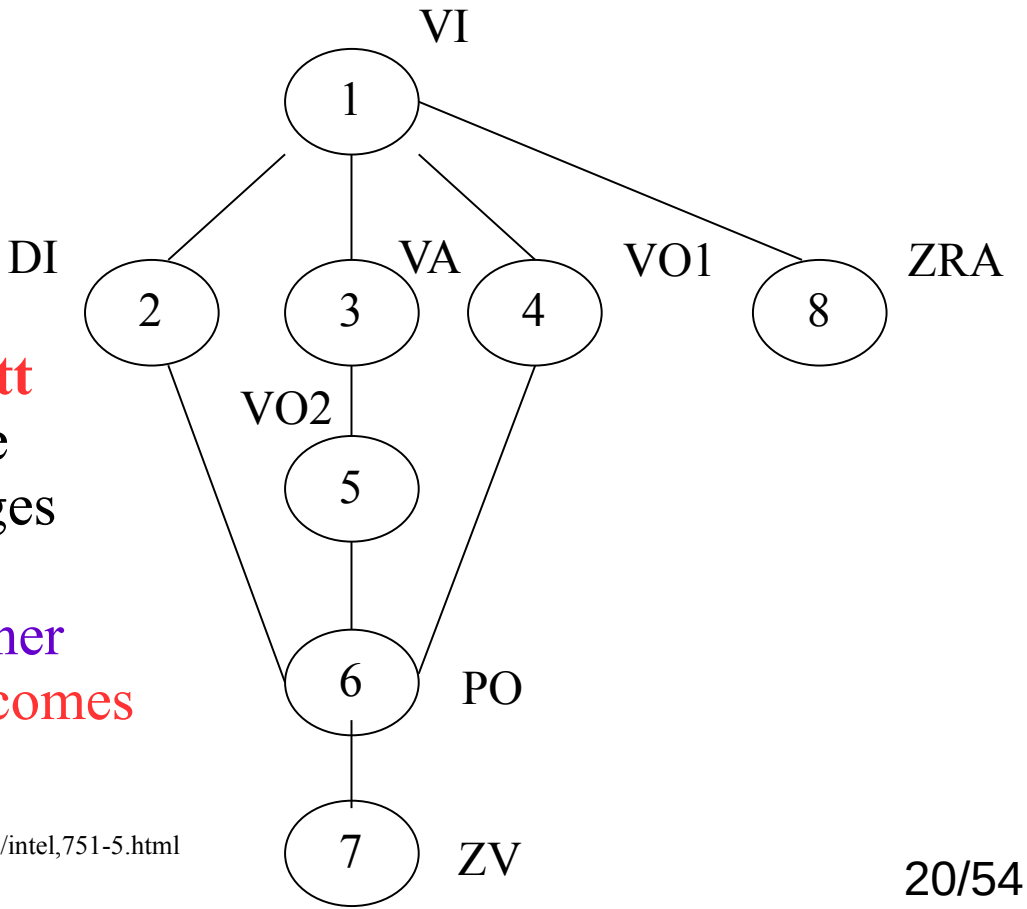
1. **V**ýber kódu **I**nštrukcie z pamäte (VI)
2. **D**ekódovanie **I**nštrukcie (DI)
3. **V**ýpočet reálnej **A**dresy operandu inštrukcie (operand 2) (VA)
4. **V**ýber **O**perandu z registra (**1**. operand) (VO1)
5. **V**ýber **O**perandu z pamäte (**2**. operand) (VO2)
6. **V**ykonanie **O**perácie (P**O**)
7. **Z**ápis **V**ýsledku (ZV)
8. **Z**väčšenie obsahu inštrukčného registra – **R**egistra **A**dries(ZRA)

Čas

Niektoré z úkonov sú navzájom nezávislé (vykonateľné paralelne), potom je možné usporiadať mikrooperácie nasledovne:

V novom usporiadaní bude 5 fáz

1. Výber kódu inštrukcie z pamäte (VI)
2. Dekódovanie inštrukcie (DI)
   (predtým DI, VA, VO1, ZRA)
3. Výber operandu z pamäte (2. operand) (VO) (predtým VO2)
4. Vykonanie operácie (PO)
5. Zápis výsledku (ZV)



Intel Pentium 4 vo verzii s jadrom Prescott

This has become even more important since the pipeline has been stretched from 20 stages to now 31 stages. Intel tries to reduce the complexity of each stage in order to run higher clock speeds. In exchange, the processor becomes more vulnerable to misprediction. **(zahodí sa všetko rozpracované)** http://www.tomshardware.com/reviews/intel,751-5.html

Postupné vykonávanie inštrukcií

1. instr. | VI | DI | VO | PO | ZV |
2. instr. | VI | DI | VO | PO | ZV |
3. instr. | VI | DI | VO | PO | ZV |

Vykonávanie inštrukcií s prekrytím (predvýber inštrukcie)

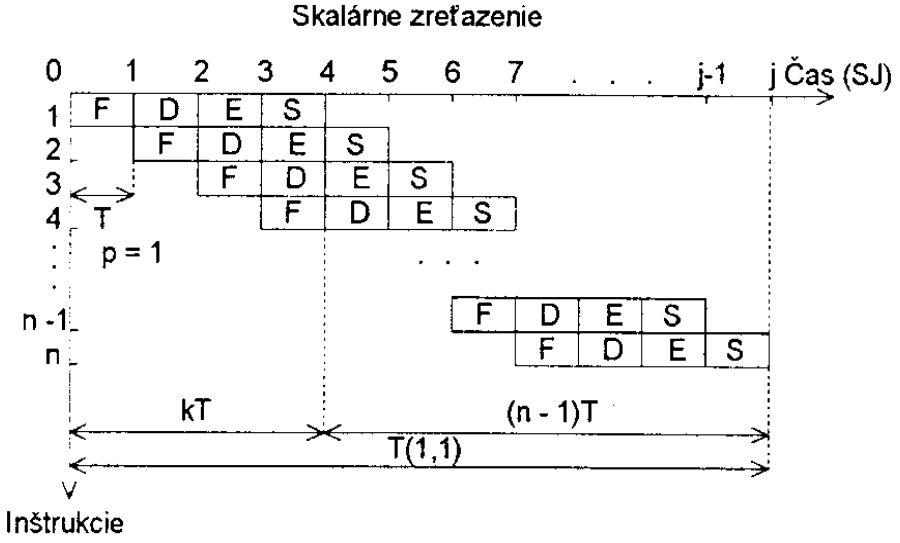
1. instr. | VI | DI | VO | PO | ZV |
2. instr. | VI | DI | VO | PO | ZV |
3. instr. | VI | DI | VO | PO | ZV |

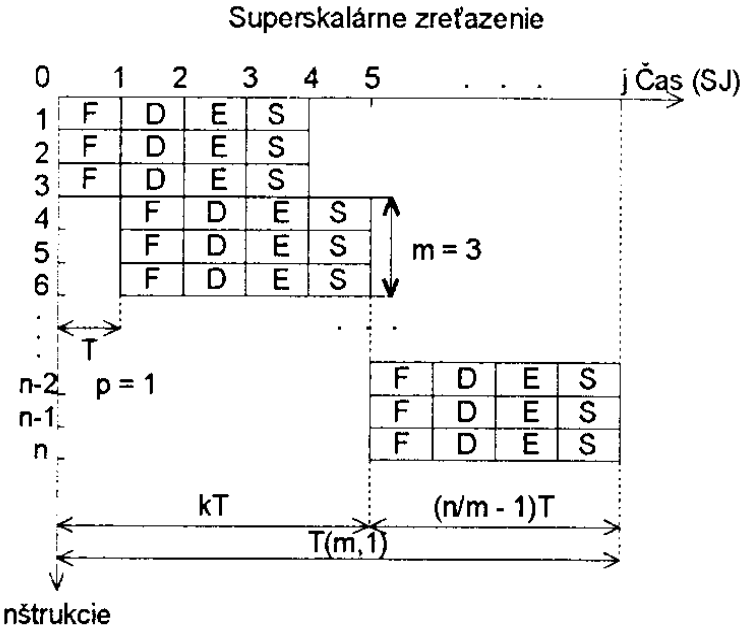
Formálne sa rozdelí vykonanie inštrukcie
- na predvýber (**FETCH**) (VI alebo VI+DI+VO) a
- vykonanie inštrukcie (**EXECUTE**) (DI+VO+PO+ZV alebo PO+ZV)

Zväčšenie počtu sekcií vedie k **prúdovému spracovaniu inštrukcií**
(všetky sekcie sú aktívne v každom takte)

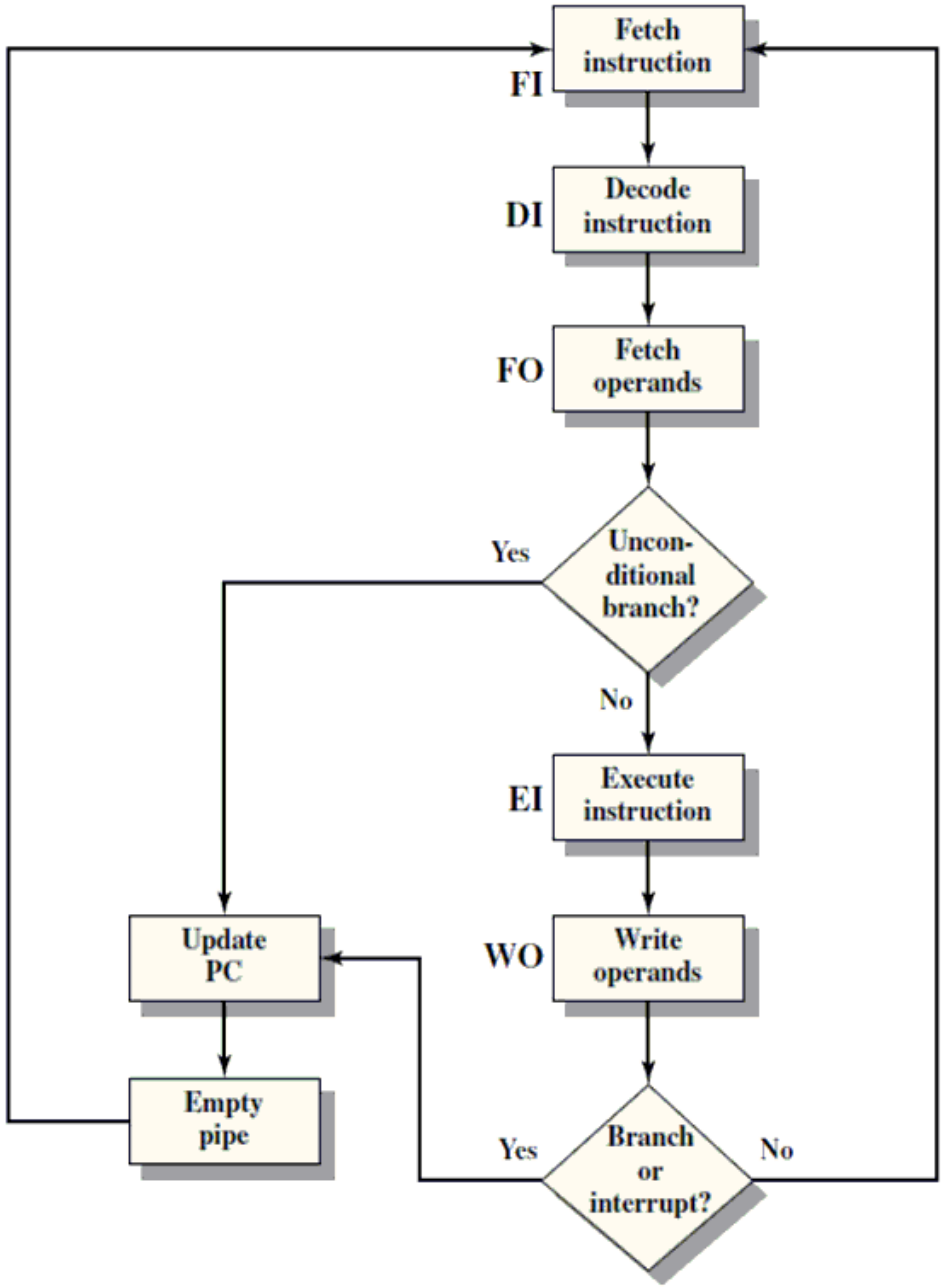
1.instr.	VI	DI	VO	PO	ZV							
2.instr.		VI	DI	VO	PO	ZV						
3.instr.			VI	DI	VO	PO	ZV					
4.instr.				VI	DI	VO	PO	ZV				
5.instr.					VI	DI	VO	PO	ZV			

Skalárne a **superskalárne** zreťazenie inštrukcií





Fázy vykonávania inštrukcie pri prúdovom spracovaní



Blokovanie sekcií – zabránenie vzniku konfliktov

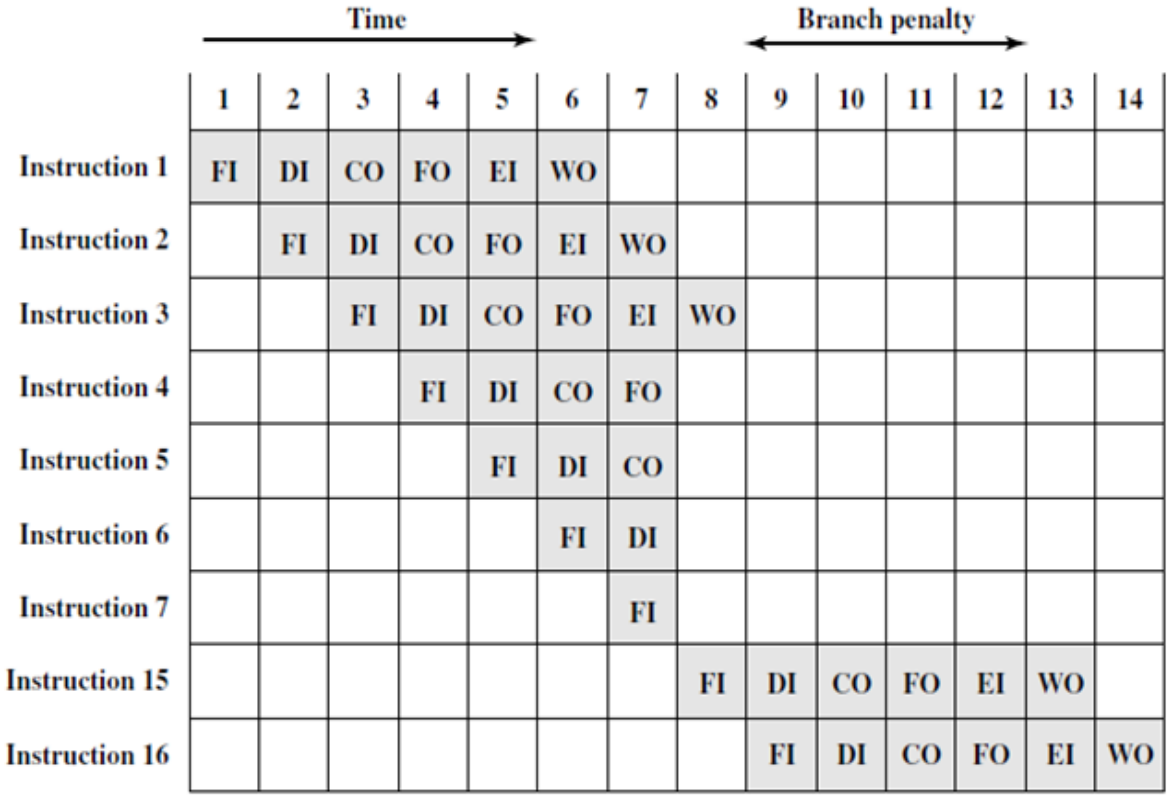
Konflikt (prostriedkov) medzi potrebou čítať operand z pamäte I1(FO) a čítaním Inštrukcie z pamäte I3(FI)

	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			Idle	FI	DI	FO	EI	WO	
I4					FI	DI	FO	EI	WO

Údajová závislosť I2(F0) musí čakať na zápis výsledku I1(W0)

	1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX	FI	DI	FO	EI	WO					
SUB ECX, EAX		FI	DI	Idle		FO	EI	WO		
I3			FI			DI	FO	EI	WO	
I4						FI	DI	FO	EI	WO

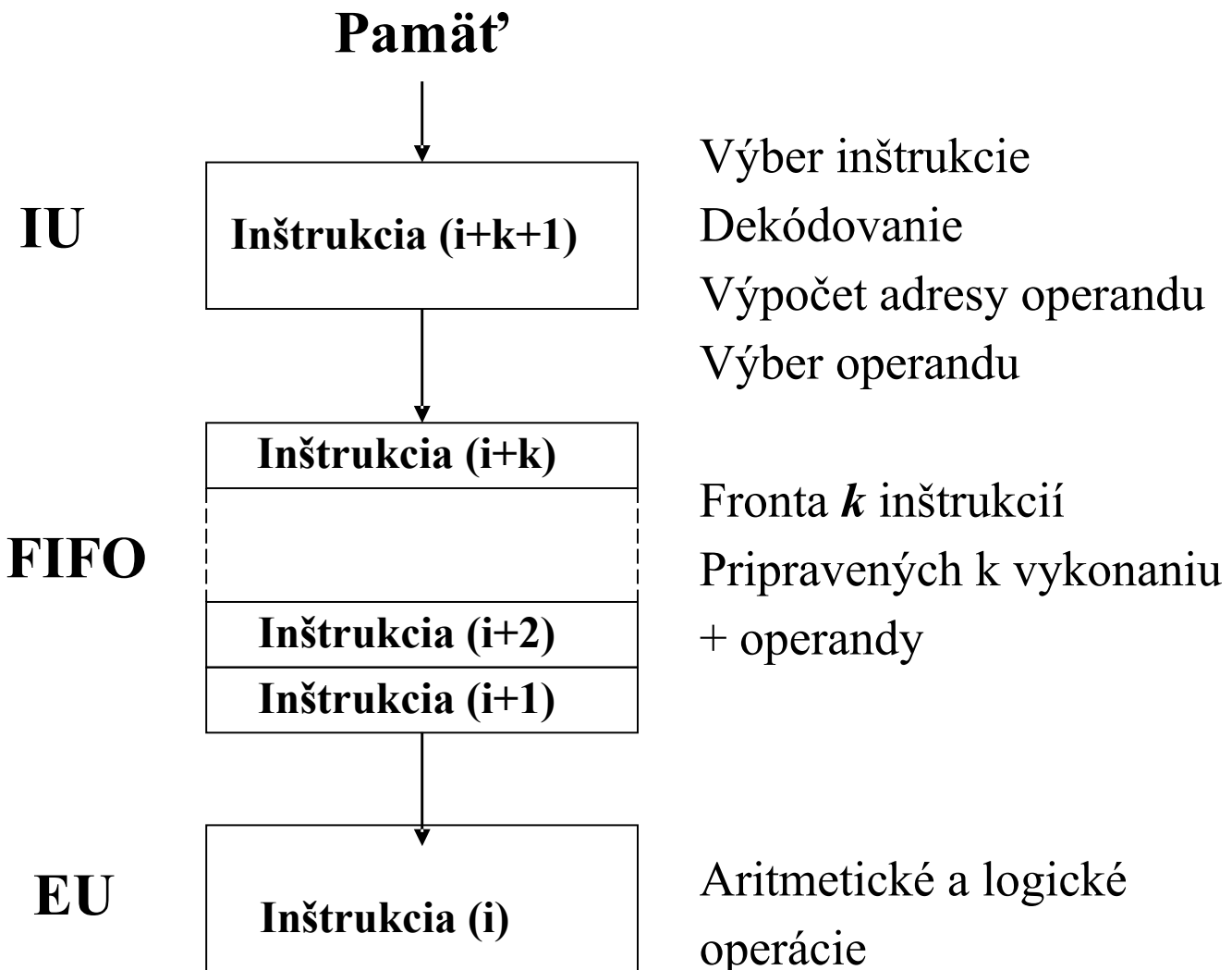
Blokovanie sekcií – pri podmienenom skoku I3(EI) je vyhodnotené, že program musí pokračovať inštrukciou I15



V **mnohých procesoroch s RISC** architektúrou sa preferuje riešenie konfliktov **blokovaním** – výhody programátor programuje ako v prípade sekvenčného vykonávania inštrukcií, podstatne jednoduchší hardvér procesora). Podobná situácia môže vzniknúť pri čítaní/zápise do pamäte (I/O)).

Inštrukčná fronta

- **predvýber** inštrukcie (výber inštrukcie, dekódovanie kódu inštrukcie, výpočet reálnej adresy operandu, výber operandu z pamäte)
- uloženie dekódovaných inštrukcií do **inštrukčnej fronty IF** (FIFO)
- vykonávacia jednotka **vykonáva** inštrukcie z inštrukčnej fronty



Problémom pri použití inštrukčnej fronty sú *skokové inštrukcie*.

-**nepodmienený skok** môže riešiť jednotka IU (začne plniť frontu od adresy skoku)

- v prípade **podmieneného skoku**, sú 3 riešenia
  - naplňovanie IF sa pozastaví po vyhodnotenie podmienky
  - v prípade, že podmienený skok vedie k zmene inštrukčného toku (vyprázdni sa inštrukčná fronta (je určitá pravdepodobnosť, že táto situácia nenastane), ale v prípade vyprázdnenia vykonávacia jednotka stojí (čaká na naplnenie FIFO)
  - v dokonalejších systémoch sa napĺňajú dve IF (hlavná a pomocná)

Literatúra:

[1] Strelec, J. Líška, M..: Architektúra procesorú RISC, Grada...,

[2] Jelšina, M.: Architektúry počítačových systémov, ...

[3] Hlavička, J.:Architektura počítačú. ČVUT 1998

[4] Krajčovič, T.: Počítače. STU FEI 2000

SUPERSKALÁRNE PROCESORY

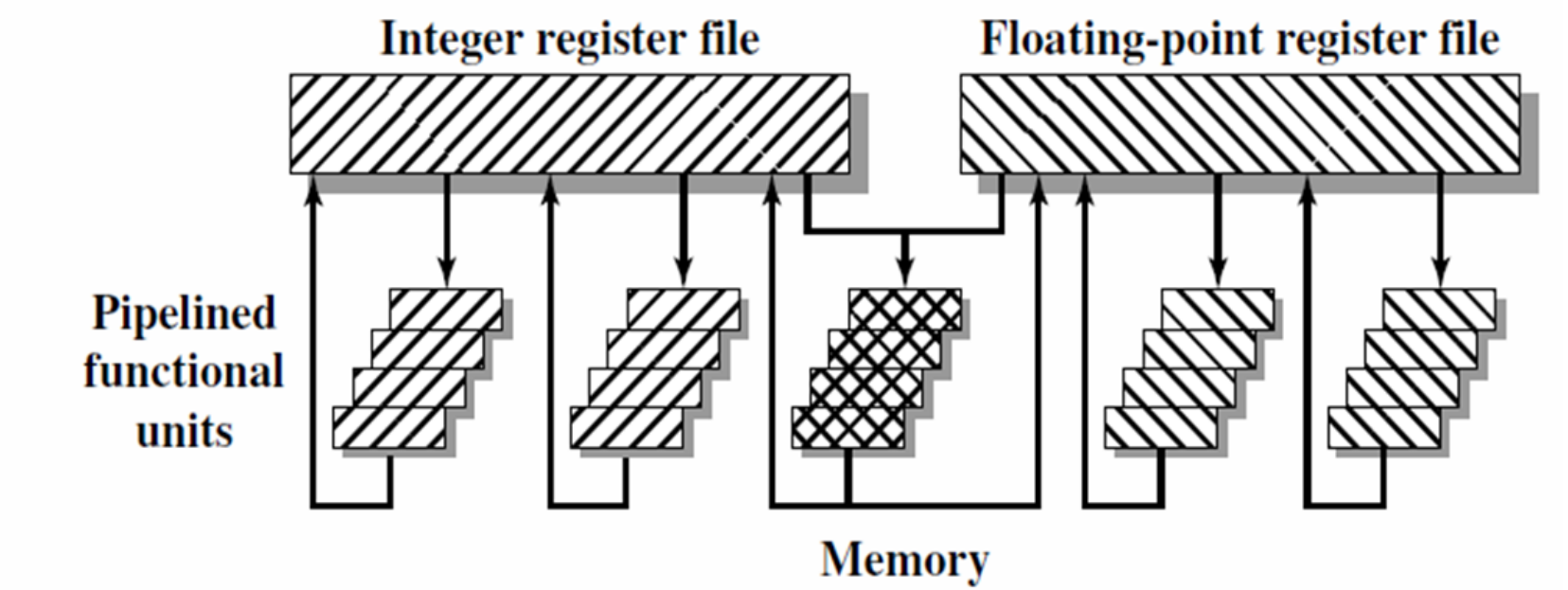
Používajú viacero **špecializovaných** jednotiek pre prúdové spracovanie inštrukcií

Výhody

– špecializovaná jednotka môže byť jednoduchšia

- zvýšenie priepustnosti procesora

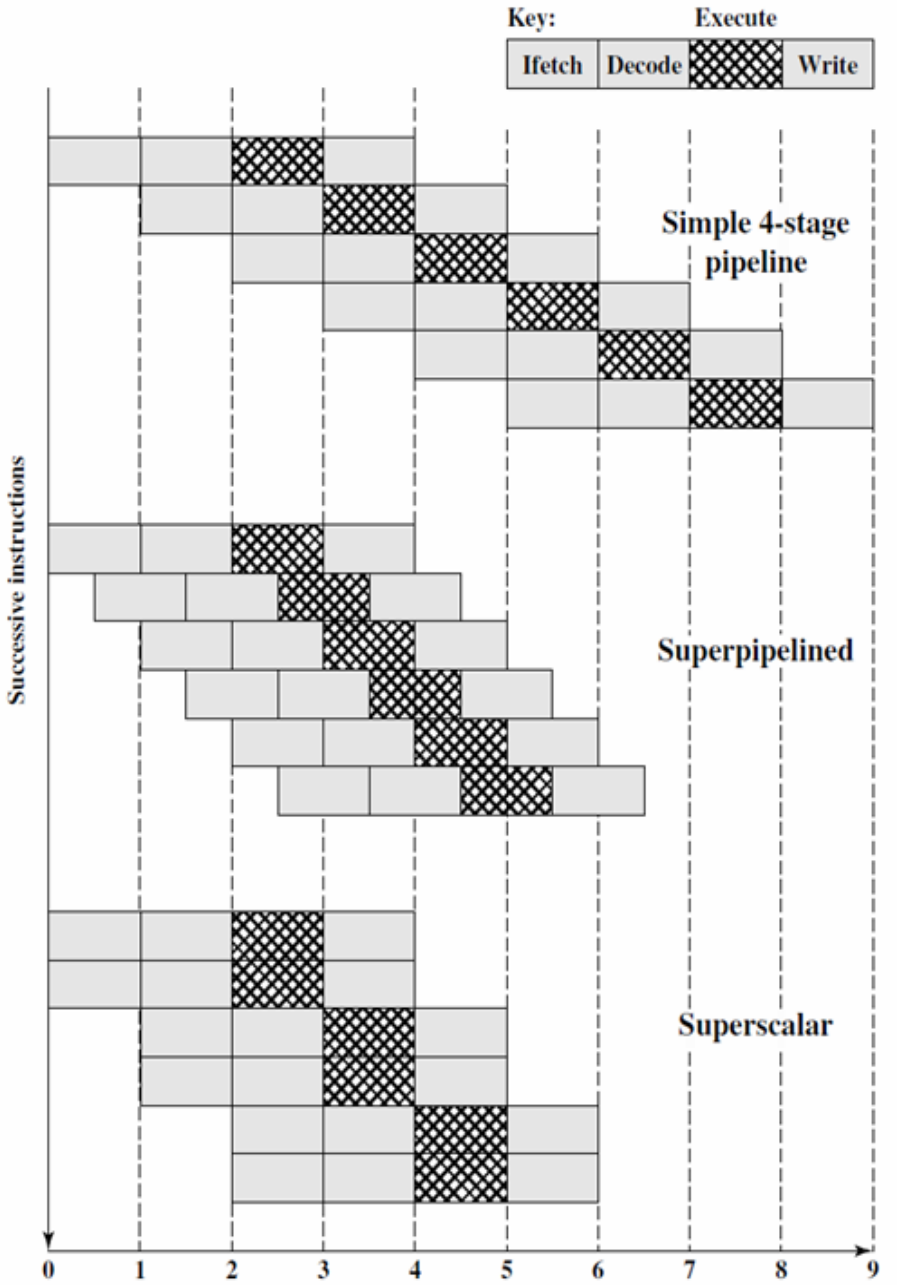
- schopnosť vykonávať inštrukcie nezávisle a paralelne



Princíp organizácie superskalárneho procesora s 5 prúdmi

2 pre FPA inštrukcie, 2 pre INT inštrukcie, 1 pre presun dát (čítanie, zápis)

Priebeh vykonávania inštrukcií bez blokovania sekcií pri **prúdovom**, **superprúdovom** a **superskalárnom** spracovaní inštrukcií so 4 sekciami



Superprúdové spracovanie je možné, ak sa dá úkon v každej sekcii rozdeliť na 2 nezávislé úkony (sekcie) synchronizované obidvomi hranami hodinového signálu (typické pre RISC procesory)

Obmedzujúce faktory pri paralelnom vykonávaní inštrukcií

- **dátová závislosť** (neaktuálnosť obsahu pamäťového miesta spôsobená nedokončením vykonávanej inštrukcie
- **procedurálna závislosť** – podmienené vetvenie programu
- závislosť daná konfliktom **pri prístupe k prostriedkom** (prístup k pamäti, obsadená špecializovaná jednotka, atď.)

Riešenia :

- **jednoduché**, **blokovanie sekcií** ako v prípade skalárnych procesorov
- použitie metód, ktoré vyžadujú **zvýšenie zložitosti procesora**

Dátová závislosť – 3 riziká

RAW (Read After Write - true depedency) – čítanie pred zápisom aktuálnej hodnoty (čakanie na zápis)

WAR (Write After Read – antidependency) – prepísanie aktuálnej hodnoty skôr, ako je prečítaná

WAW (Write After Write - output dependency) – prepísanie aktuálnej hodnoty, inštrukciou, ktorá mala byť dokončená pred zápisom

Začatie a dokončovanie inštrukcií v poradí

Príklad:

I1 vyžaduje 2 cykly na vykonanie

I3 a I4 vyžadujú rovnakú funkčnú jednotku (je len jedna)

I5 závisí od výsledku I4.

I5 a I6 vyžadujú rovnakú funkčnú jednotku

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

I3,I4 môžu byť spustené až po vykonaní I1 (dokončovanie v poradí)

I3,I4 nemôžu byť spustené naraz

I5 až po I4

I5, I6 nemôžu byť spustené naraz

Začatie inštrukcií v poradí a dokončovanie inštrukcií mimo poradia

(rovnaký príklad)

Decode		Execute			Write		Cycle
							1
I1	I2						2
I3	I4	I1	I2				3
	I4	I1		I3	I2		4
I5	I6			I4	I1	I3	5
	I6		I5		I4		6
			I6		I5		7
					I6		8

Zápis výsledku I2 nemusí čakať na zápis výsledku I1 (**dokončenie inštrukcie mimo poradia**)

V druhej fáze I1 môže začať I3 má voľnú funkčnú jednotku

(oproti predchádzajúcemu príkladu skrátime výkon inštrukcií o **1 SC**)

Začatie inštrukcií aj dokončovanie inštrukcií mimo poradia

(rovnaký príklad)

Decode		Window	Execute			Write		Cycle
								1
I1	I2							
I3	I4	*I1, I2*	I1	I2				2
I5	I6	*I3, I4*	I1		I3	I2		3
		I4, I5, I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

Na spúšťanie inštrukcií mimo poradia treba mať pomocnú tabuľku (window), kde v každej fáze sa vyhodnocuje bezkonfliktné spustenie inštrukcie.

I1, I2 – je bezkonfliktné

I3, I4 – nemôžu byť spustené naraz (spustenie I3, I4 zostáva)

I4, I5, I6 – I5 závisí od výsledku I4, spustenie I4, I6 (**I6 je spustené pred I5**) nakoniec spustenie I5

Výsledok spustenie aj dokončovanie mimo poradia. Ale zisk 1SC, oproti predchádzajúcemu prípadu.

Premenovanie registrov

I1: R3 ← R3 op R5
I2: R4 ← R3 + 1
I3: **R3 ← R5 + 1**
I4: R7 ← R3 op R4

I1: R3b ← R3a op R5a
I2: R4b ← R3b + 1
I3: R3c ← R5a + 1
I4: R7b ← R3c op R4b

op – ľubovoľná operácia

Riziko, ak nezačne čítanie z R3 (I2) skôr ako zápis do R3 (I3), potom chyba (WAR) !!!

Riešenie problému dátovej závislosti **WAR**, **WAW,** nie však **RAW!!!**

WAR, **WAW** je najčastejšie spôsobovaná práve **začatím** a **dokončovaním** inštrukcií **mimo poradia**. Ak je podporovaná táto možnosť, musí byť k dispozícii toľko sád, koľko je paralelných prúdov v procesore.

Riešenie vetvenia programu (podmienené skoky)

Snaha dosiahnuť **minimálny** počet konfliktov **s blokovaním** a **vyprázdňovaním** sekcií.

Dôležitý predpoklad, žiadna vetva nesmie byť dokončená zápisom výsledku!!! pred správnym rozhodnutím. **Nikdy nie je garantovaný odhad** správneho pokračovania toku inštrukcií.

Možné riešenia :

- **viacprúdové riešenie**, do výberu vstupujú inštrukcie z obidvoch (viacerých) smerov (spomaľuje čítanie z pamäte, jedna vetva (vetvy) po vyhodnotení sa zrušia nesprávne vetva (vetvy))
- **predvýber v smere skoku**, výber inštrukcií za príkazom skoku
- **tabuľka skokov** – cache pamäť pre inštrukcie (CAM), výhodné pri cykloch, okrem prvého cyklu, cyklus beží na základe sekvencie uloženej v cache (bez čítania za pamäte), profit je urýchlenie, ale keď vypadne z tabuľky nemáme žiadnu informáciu
- **predikcia skokov –** V prípade nesprávnej predpovede podelíte lokálnu efektívnu f počtom stages. **viď. jadro Prescoott na priesvitke č. 29** - úspešnosť predpovede „len“ 80%.K6-III mala 98,7% ale oveľa kratšiu pipeline t.j. jednoduchšiu logiku predpovedí. Zložitosť logiky predpovedí s rovnakou úspešnosťou rastie s druhou mocninou počtu pipeline stages.

PREDIKCIA SKOKOV

Statická predikcia

- **hypotéza** - podmienka skoku **nikdy nesplnená**
- **hypotéza** - podmienka skoku **vždy splnená** (často používaná, cykly),

analýzy zistili, že **viac ako 50%** skokov splní podmienku, je tu riziko, ak je skok **vzdialený** (ďaleký), môže vyvolať **page fault (výpadok cache pamäte)**

-rozhodovanie **podľa kódu inštrukcie** – podľa štatistických analýz niektoré z nich skáču s **pravdepodobnosťou** viac ako **75%**

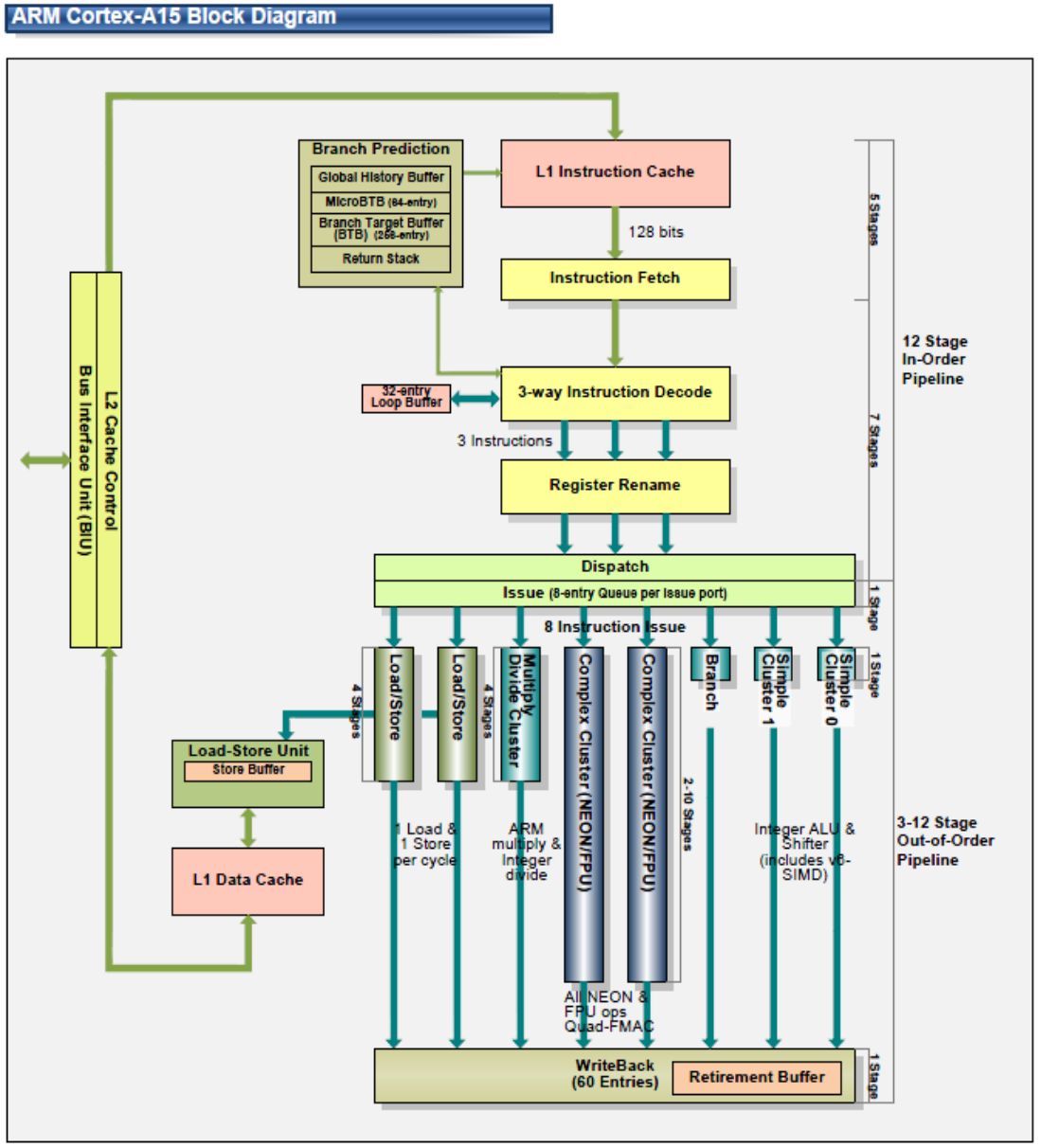
Dynamická predikcia

zvyšuje percento **správneho** rozhodnutia

- **história skokov (** cache pamäť zaznamenáva sa históriu výsledkov skoku v danom mieste programu, umožňuje prognózovať smer toku programu (cache histórie) po vyprázdnení strata informácie
- stačí **zoznam skokových** inštrukcií a pridať 1 alebo 2 bity, ktoré charakterizujú posledný výsledok skoku

KOMPILÁTOR ČASTO MANIPULUJE S VÝSLEDKOM PREDPOVEDE TAK, ŽE PRIDÁ ZBYTOČNÉ INŠTRUKCIE, KTORÝCH JE MENEJ AKO STRATA PRI NESPRÁVNEJ PREDPOVEDI, ALE ZARUČIA NIMI SPRÁVNU PREDPOVEĎ

Príklad superskalárneho procesora ARM Cortex-A15



Aplikačné oblasti:

Advanced Smartphones

Mobile Computing

High-end Digital Home Entertainment

Wireless Infrastructure

Low-power Servers

8- paralelných prúdov (nerovnakého a premenlivého počtu sekcií)

Podpora vektorových výpočtov (SIMD)

In-Order (12 sekcií), Out-of-Order (3-12 sekcií)

TRI ZÁKONY https://lionoftheblogosphere.wordpress.com/2014/12/01/moores-law-has-ended-but-no-one-noticed/

- **MOOREOV ZÁKON** (autor, zamestnanec Shockley, spoluzakladateľ Fairchild Semiconductor a Intelu, Gordon Moore) **zo 19.4.1965** (Happy Birthday k polstoročiu :)))
- http://www.extremetech.com/extreme/203031-moores-law-at-50-its-past-and-its-future

1965: Single component planar transistor invented in 1959, Moore noticed that the number of components was roughly doubling every year.

Revízia z 1975: He theorized that this meant computing power on a chip could double about every 18 months — slower than Moore's original 1965 prediction, but faster than the 1975 revision
(platí pri ideálnom využití tranzitorov programom)

- **GATESOV ZÁKON:** The speed of commercial software slows down by half every 18 months (Pre neefektívne optimalizácie prekladačmi=kompilátormi)

- **WIRTHOV ZÁKON-1995**: Software is getting slower more rapidly than hardware becomes faster due to bloat
http://antranik.org/using-moores-law-to-predict-future-memory-trends/

VÝKONNOSŤ POČÍTAČOV (procesorov) – zvyšovanie

Objektívne analýzy preukazujú, že relatívne porovnanie výkonnosti medzi rôznymi architektúrami a koncepciami je veľmi ťažké. Cesty zvyšovania výkonnosti:

- **zvyšovanie frekvencie** procesora
- **paralelizmus vykonávania inštrukcií** – počet **dokončených** inštrukcií za cyklus (prúdové spracovanie, superskalár)
- **paralelné vykonávanie programu** (paralelná štruktúra programu) a **programov**

Proces – inštancia programu spustená v počítači

Atribúty procesu

-**vlastníctvo prostriedkov** - virtuálny adresný priestor pre kód, údaje, zásobník, s možnosťou privlastnenia riadenia časti pamäte, I/O kanálov a súborov

-**vykonávanie (plánovanie)** – program beží pozdĺž jedného alebo viacerých programov (je **prekrývaný** a **prerušovaný** inými procesmi)

- **prepínanie procesov** – **prideľovanie procesora** viacerým procesom

Vlákno (thread)

– **má prístup k prostriedkom procesu** (hlavného vlákna),

- **vykonáva relatívne samostatnú** činnosť „**paralelne s procesom**“ (môže mať svoj priestor pre dáta, atď.)
- OS dokáže **prepínať vykonávanie vlákien procesu**, prepínanie vyžaduje menšiu réžiu ako prepínanie procesov.

Multithreading

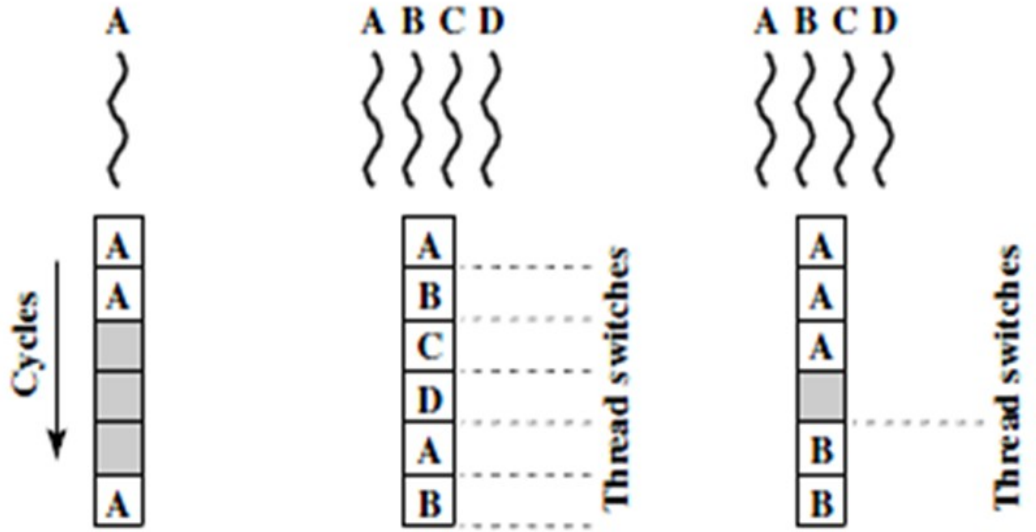
súčasné (alebo **„súčasné“**) vykonávanie viacerých vlákien / threadov

- **implicitný** (generuje kompilátor)
- **explicitný** – súčasné vykonávanie viacerých threadov a/alebo procesov

Explicitné :

- **Interleaved multithreading** (fine-grained multithreading) – (prekladaný)
  pri blokovaní sa prepína na iný thread (prepínanie – napr. page fault)
- **blocked multithreading** (coarse-grained multithreading) - blokujúci
- **S**imultaneous **M**ulti**T**hreading (**SMT**) – vybavený superskalárny procesor
- **Chip multiprocessing** – viacjadrové (multicore) procesory

Multithreading v skalárnych procesoroch



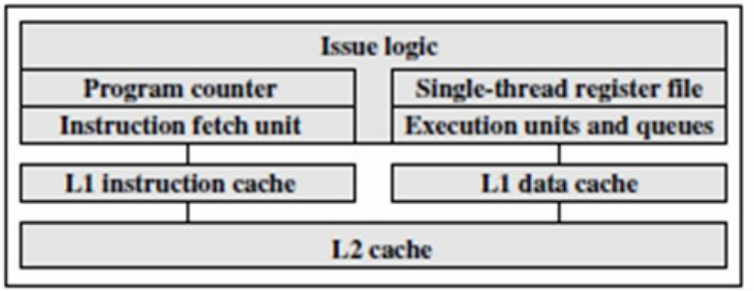
single-thread prekrývaný blokujúci multi-thread

SMT a VIACJADROVÉ PROCESORY

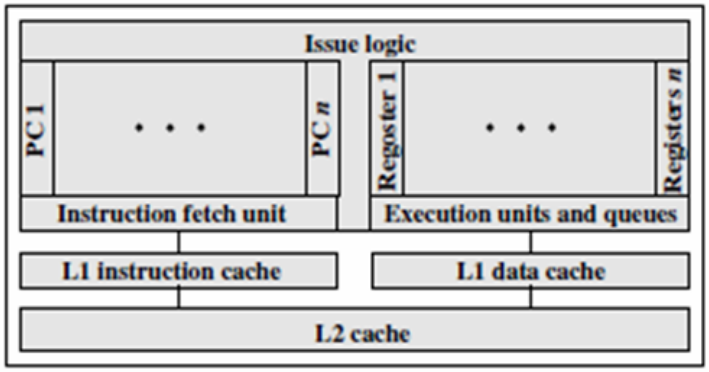
Zvyšovanie priepustnosti procesora (počet vykonaných inštrukcií za jednotku času)

SMT (**S**imultaneous **M**ulti**T**hreading) – súčasné vykonávanie inštrukcií viacerých vlákien – **doplnenie** superskalárnej architektúry (paralelné prúdovo pracujúce jednotky) **registrovými sadami** (PC,IR, SP, univerzálnymi registrami) (v Intel procesoroch **SMT=HyperThreading**)

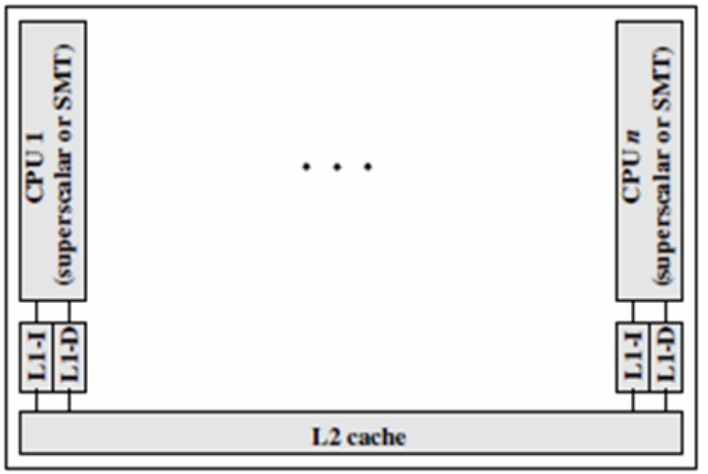
Viacjadrové procesory (multicore procesor) – **viacero jadier** (superskalárny CPU alebo SMT) na jednej podložke (v jednom čipe)



Superskalárna architektúra

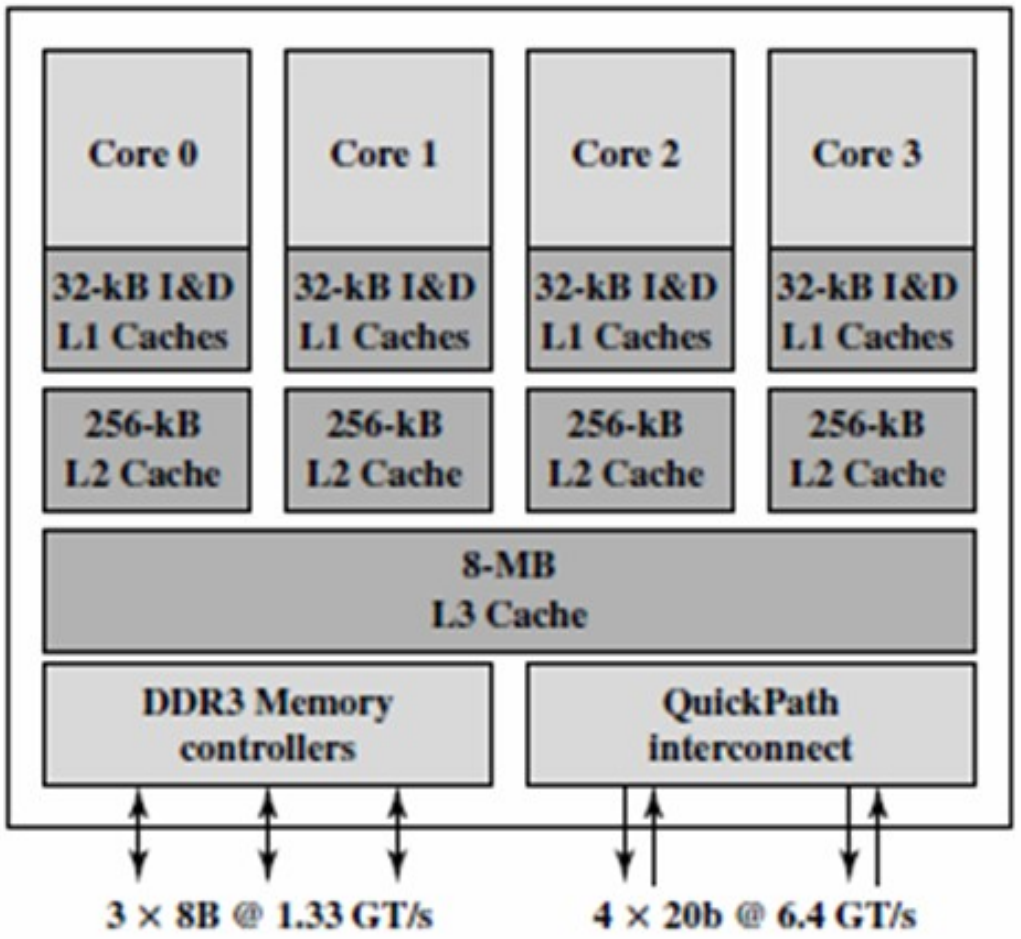


SMT architektúra



Viacjadrový procesor

Príklad organizácie viacjadrových procesorov

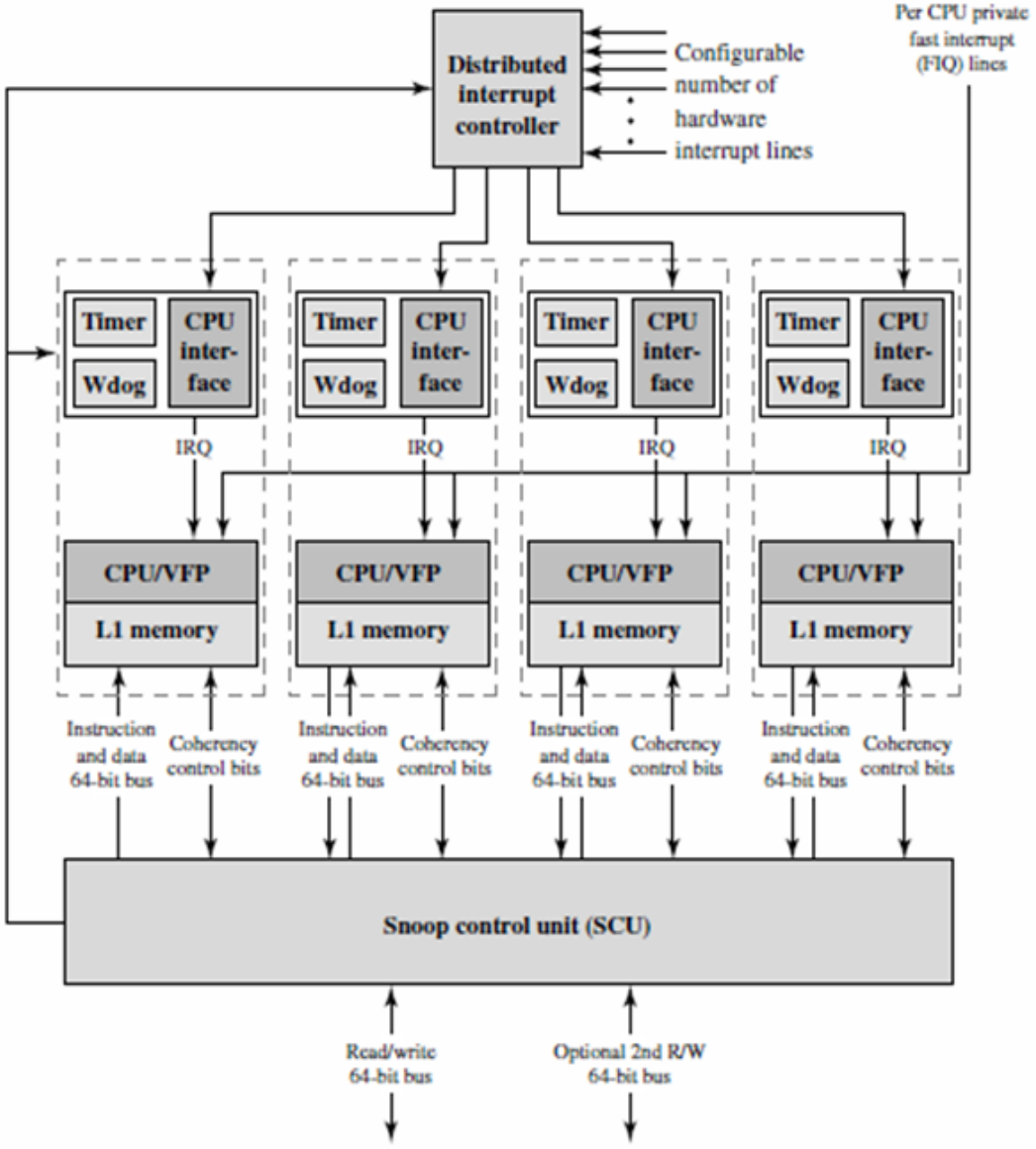


4-jadrový procesor Intel Core i7

Latencia cache pamätí: L1 – 4 CC, L2 – 11 CC , L3- 39 CC

L1 – oddelené cache pamäte L1I a L1D

Zvonka je procesor Princetonskej architektúry, Harvardskej architektúry
CISC – (do prúdového spracovania inštrukcií vstupuje prúd mikroinštrukcií)



ARM11 MPCore procesor – RISC architektúra

Wdog – WatchDog
zabezpečenie proti zablúdeniu
alebo pádu programu, vrátane OS

CPU – single procesor ARM11
VFP – Vector floating-point
(hardvérovo realizovaný)
L1 – zdieľaná L1I a L1D

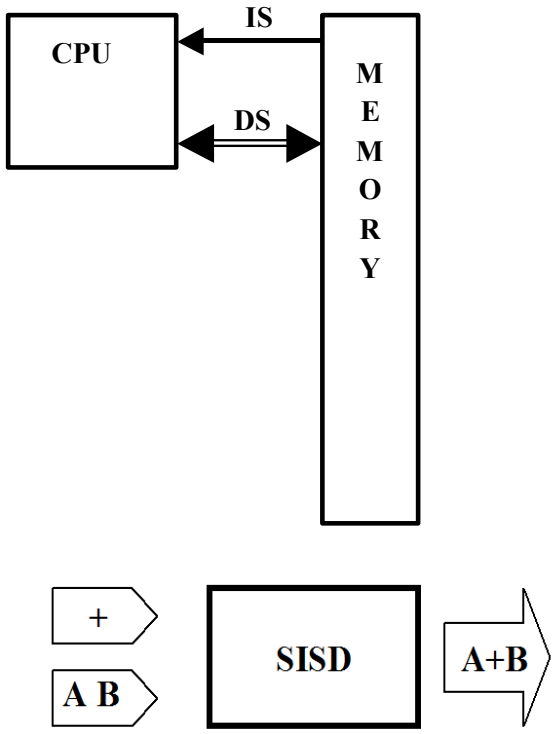
SCU – Snoopy Control Unit
sleduje a riadi koherenciu
všetkých L1D po zápise
CPU do cache

MULTIPROCESOROVÉ POČÍTAČE

Klasifikácia počítačov podľa Flynna (1966)

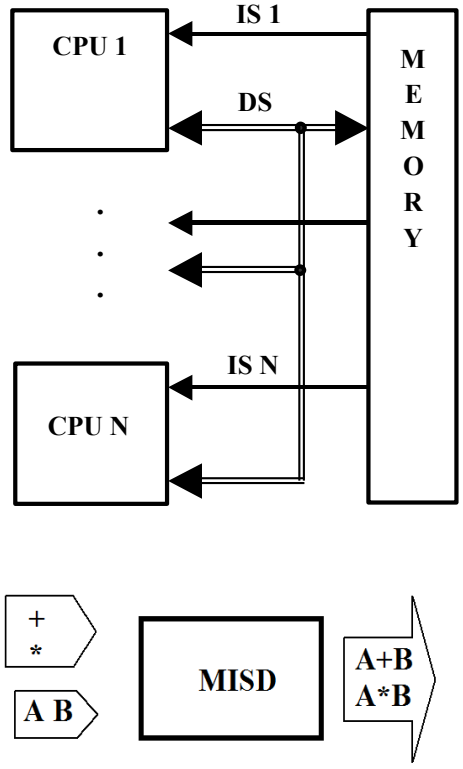
Pri vykonávaní výpočtového procesu sa medzi procesorom a pamäťou prenášajú:
prúd inštrukcií (**IS** - **I**nstruction **S**tream) a prúd údajov (**DS** - **D**ata **S**tream) .
Ich počet definuje odpovedajúcu architektúru.

SISD - Single IS, Single DS



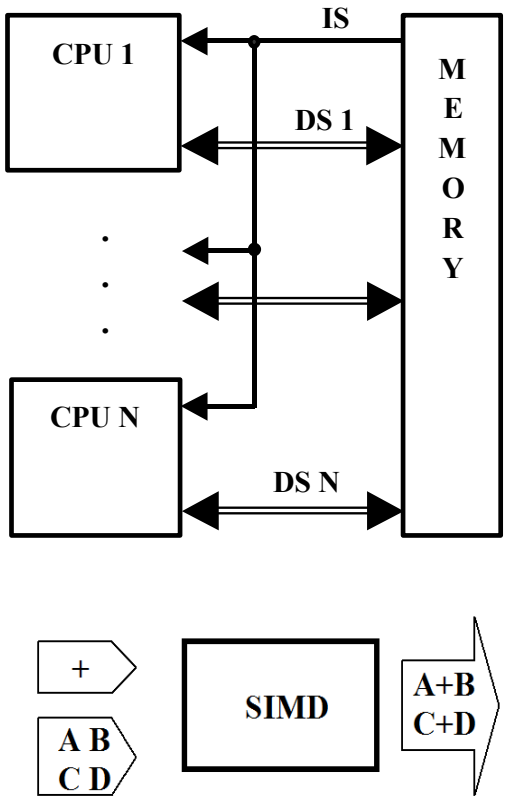
jednoprocesorové počítače

MISD - Multiple IS, Single DS



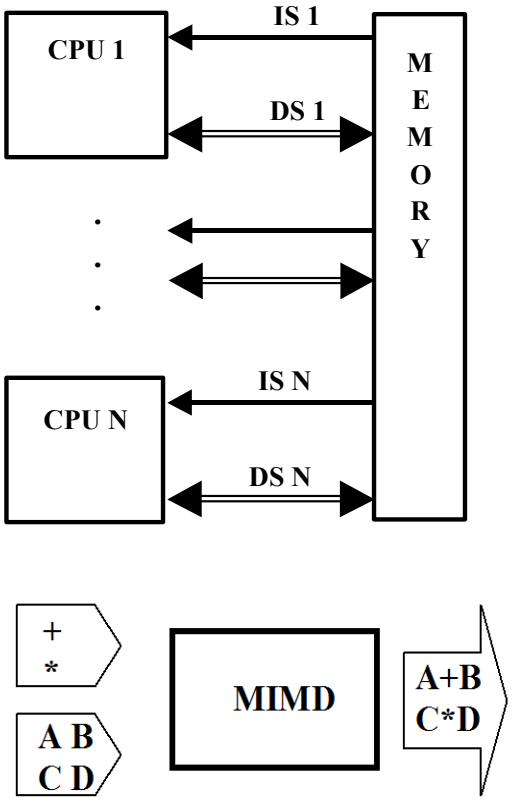
prakticky sa nepoužívajú

SIMD - Single IS, Multiple DS



Vektorové a
maticové procesory

MIMD - Multiple IS, Multiple DS



Multiprocesorové a
multipočítačové systémy

Klasifikácia MIMD

podľa prístupu k operačnej pamäti (OP)

Multiprocesorové počítače

jeden adresný priestor OP, zdieľaná OP

UMA (Uniform Memory Access) - centrálna OP

– PVP (Paralell Vector Processor)

- SMP (Symmetric MultiProcessors)

NUMA (Non-Uniform Memory Access) - distribuovaná OP

– COMA (Cache Only Memory Architecture)

- CC-NUMA (Cache-Coherent NUMA)

- NCC-NUMA (Non-Cache-Coherent NUMA)

Multipočítačové systémy

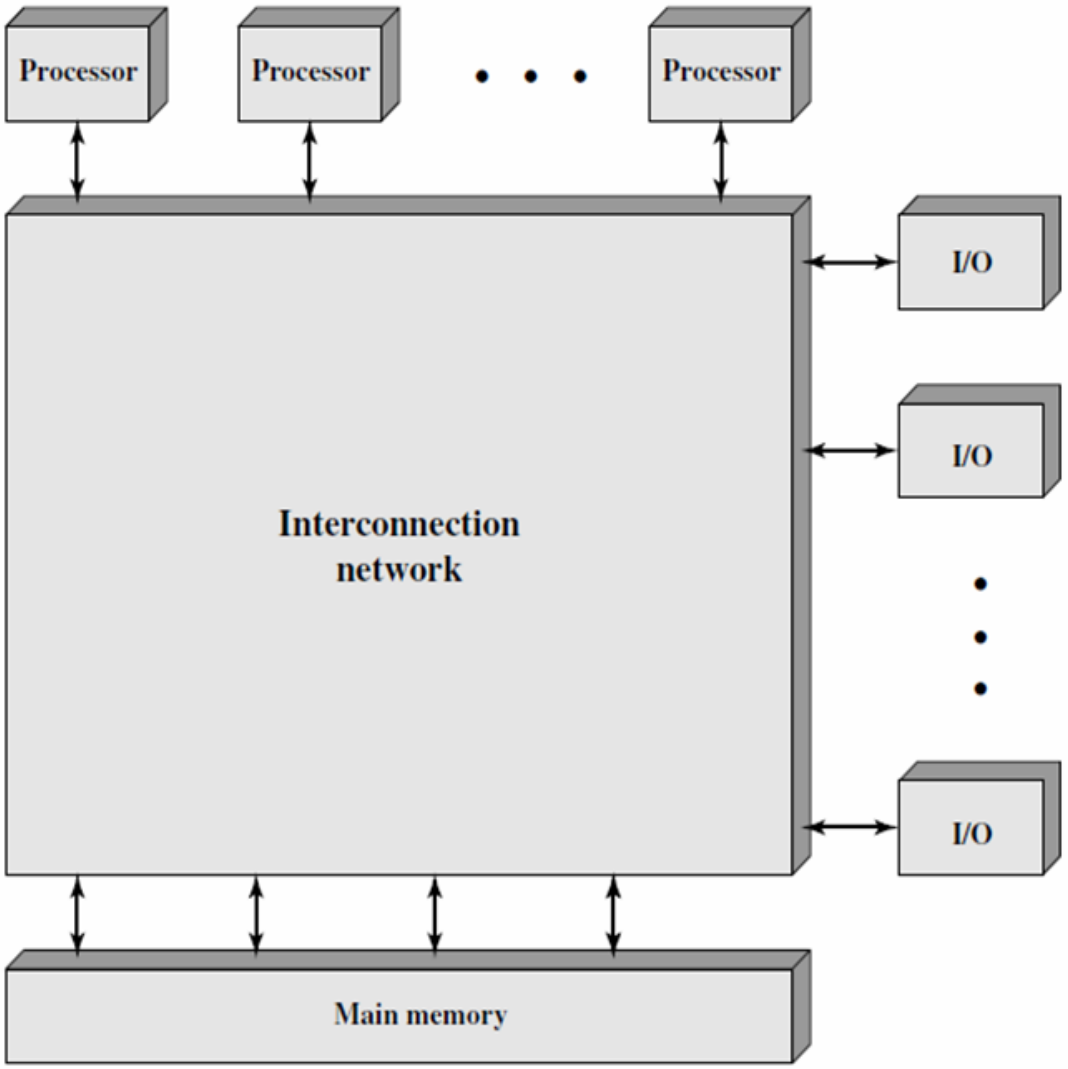
viacero adresných priestorov OP

NORMA (NO-Remote Memory Access) – oddelené OP

- Cluster – voľne viazané systémy, viacero operačných systémov

- MPP (Massively Parallel Processor) – tesne viazané systémy, jeden operačný systém

PEVNE VIAZANÉ MULTIPROCESOROVÉ SYSTÉMY

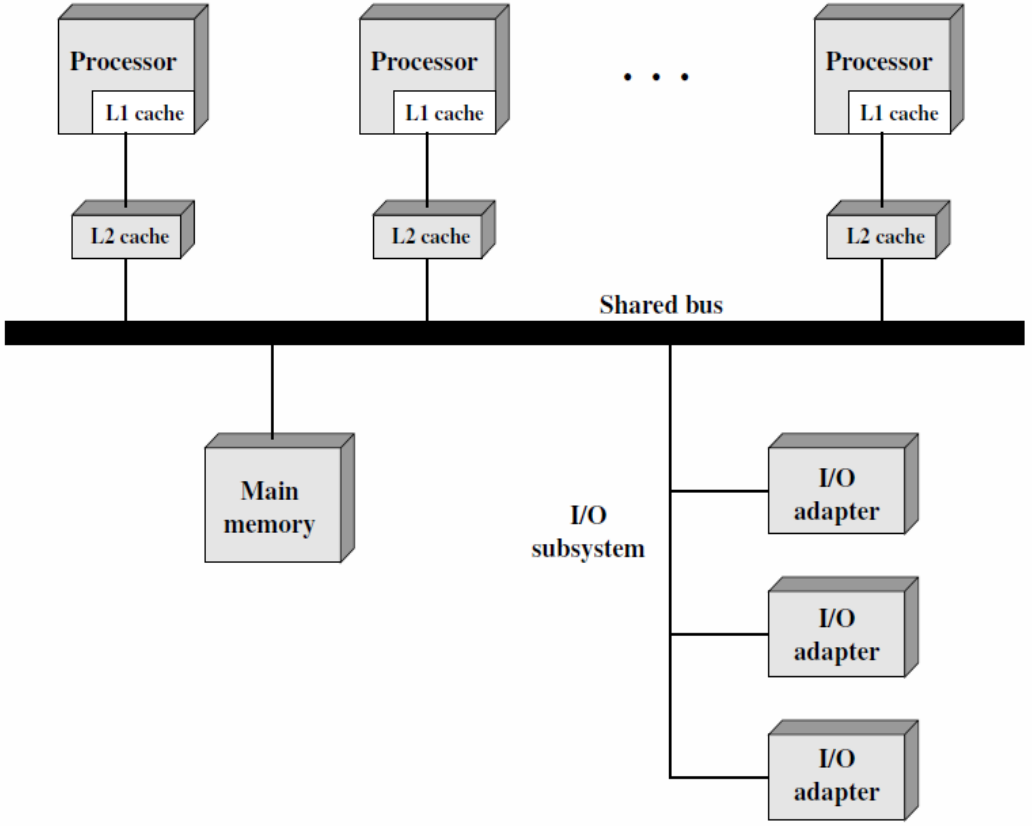


Všeobecná schéma pevne viazaných procesorov

SYMETRICKÝ MULTIPROCESOROVÝ SYSTÉM

(**SMP** - **S**ymmetric **M**ulti**P**rocessors)

- 2 a **viac** procesorov s **porovnateľnými vlastnosťami**
- **zdieľajú spoločnú** pamäť aj I/O podsystém
- všetky procesory môžu sú **schopné vykonávať rovnaké funkcie** (symetria)
- systém riadi **OS** (**O**peračný **S**ystém) zabezpečuje **spoluprácu procesorov**
- OS **plánuje procesy alebo vlákna** voči všetkým procesorom



Hlavné problémy

-zdieľanie spoločných prostriedkov

- koherencia L2 cache pamäte

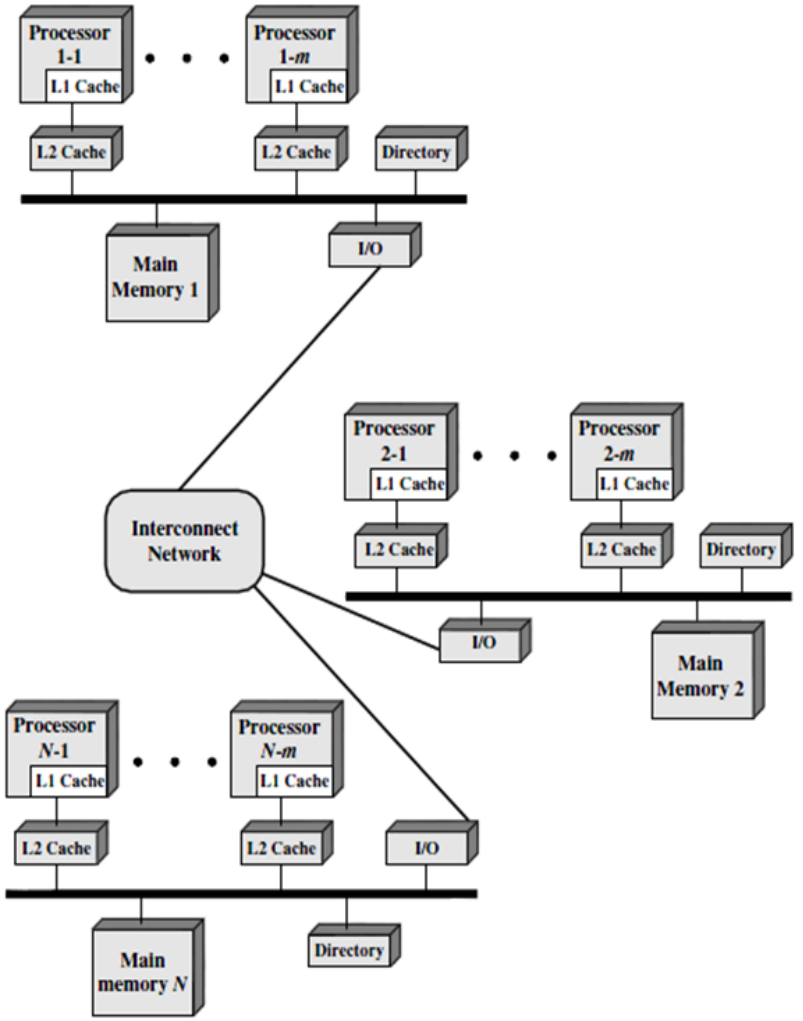
Koherencia cache pamäte

primárny problém spoločnej pamäte pre viaceré procesory.
Každý procesor musí mať v cache pamäti platné (aktuálne údaje). Riešenie koordinácia zápisu do cache pamäte. Cache pamäť musí udržiavať informáciu o neplatnosti načítaného obsahu položky.
Možné riešenia :
DIRECTORY PROTOCOLS - centrálne riadenie s vedením evidencie o obsahu a zmenách všetkých cache pamätí.
Pri požiadavke zápisu do cache pamäte je centrálne riadenie požiadané o pridelenie **exkluzivity,** riadenie zašle správu o neplatnosti položky, potom povolí zápis.
SNOOPY PROTOCOLS – každá zmena sa oznámi všetkým, všetci musia sledovať správu a vyznačiť zmenu
MESI PROTOCOLS – 2-bitová informácia pre každý TAG cache pamäte, obsah môže byť v 4 stavoch (**M**odified, **E**xclusive,**S**hared, **I**nvalid) – rozhoduje o tom, či je možné použiť obsah chche pamäte na čítanie informácie, alebo je potrebné aktualizovať obsah

NUMA multiprocesorový systém

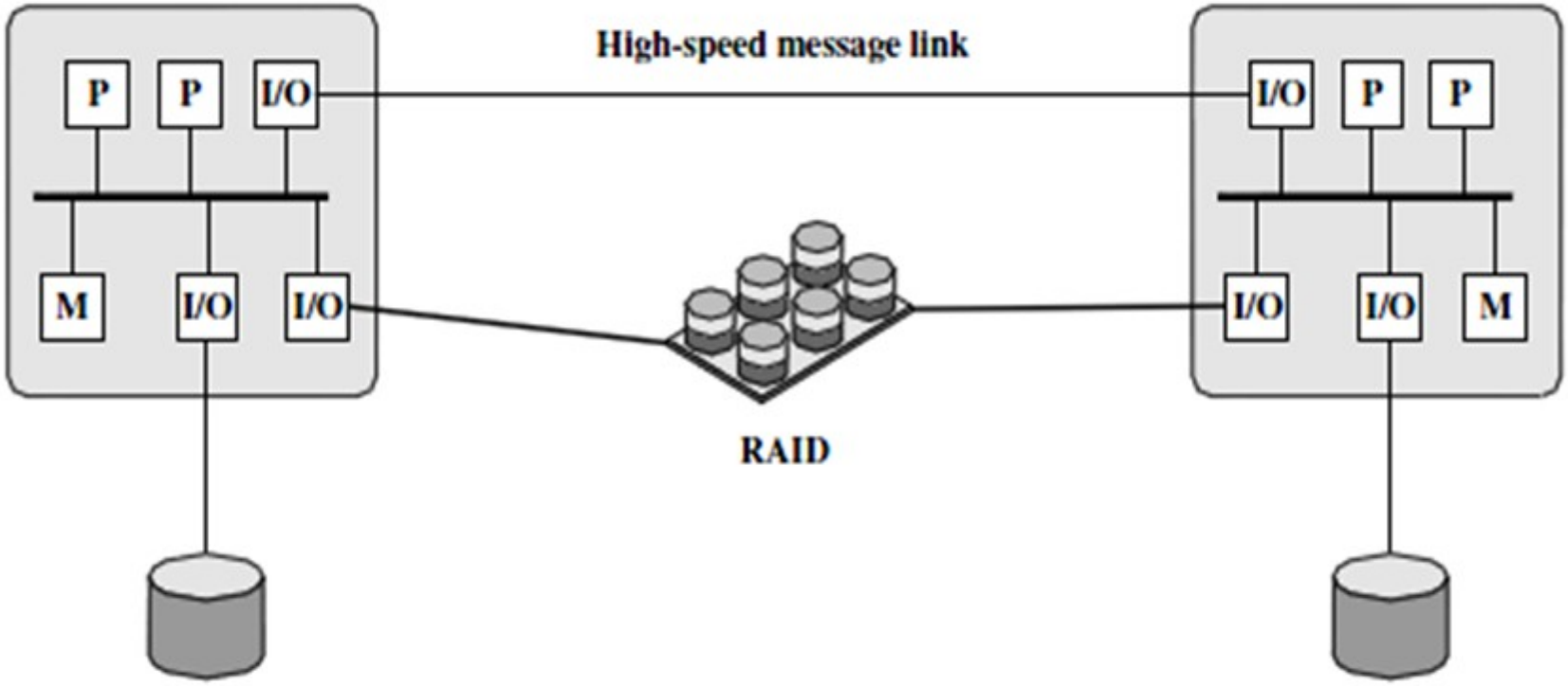
(CC-NUMA – Cache Coherent NUMA)

- prepojenie uzlov
- všetky pamäte sú prístupné všetkým uzlom (jedinečná adresa v rámci celého systému)
- prístup k pamäti v uzle je najrýchlejší
- čítanie z pamäte zabezpečuje riadenie cache uzla (pomocou tabuliek)
- čítanie z iného uzla ide cez prepojovaciu sieť
- správa koherencie obsahu cache
- priepustnosť nižšia ako SMP
- rieši priestorová a časová lokalita spúšťaných procesov (úloha OS)



CLUSTER

- voľne viazané MIMD s distribuovanou pamäťou
- je alternatívou SMP
- vyznačuje sa vysokým výkonom (serverové aplikácie)
- uzly tvoria plnohodnotné multiprocesorové počítače, prepojené vysokorýchlostými (LAN, WAN)
- integrujúcim faktrom je programové vybavenie a spoločné diskové polia, ktoré vytvára dojem jedného výpočtového zdroja (celku)

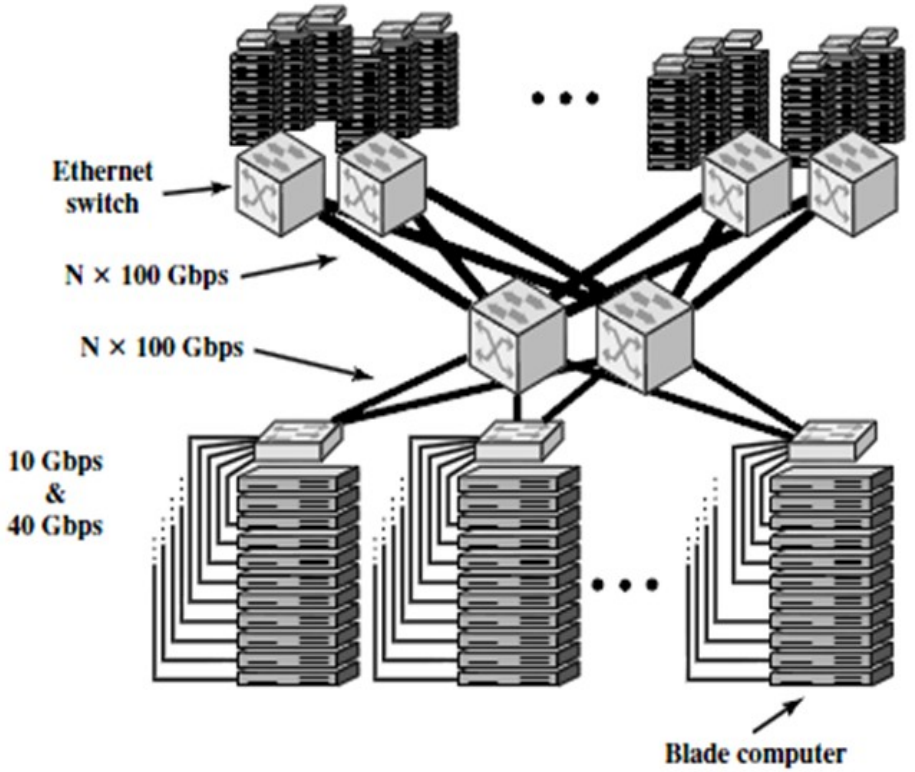


Najčastejšie uvádzané výhody

-absolútna škáľovateľnosť (dajú sa zostaviť systémy od desiatok do tisícok multiprocesorových uzlov)

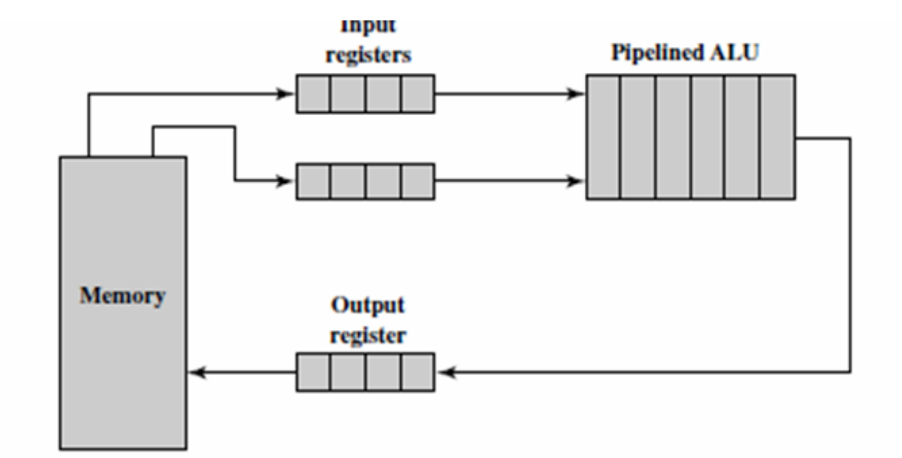
- **prírastková rozširiteľnosť** – cluster je nastavený tak, že je možné pridávať ďalšie uzly
- **vysoká dostupnosť** – porucha uzla sa rieši automaticky softvérom
- **vynikajúci pomer cena/výkon**

Použitie v dátových centrách – mohutné servery zostavené z modulov

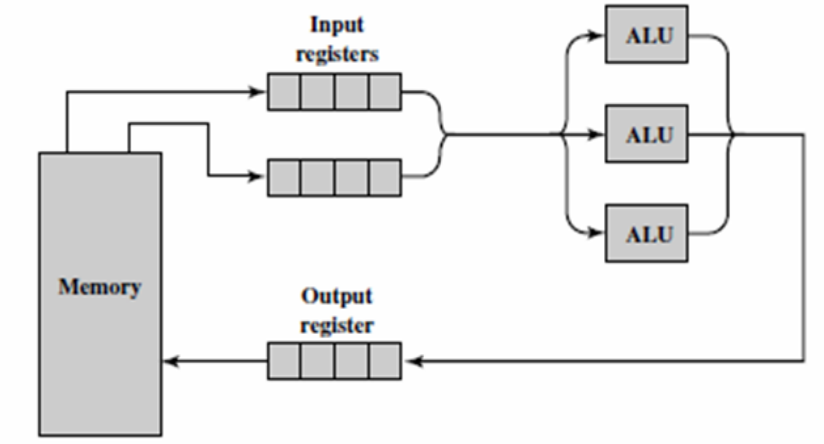


VEKTOROVÉ A MATICOVÉ PROCESORY (SIMD)

- riešené vektorovými procesormi založenými na (**FPA ALU**)
  - **skalárnej** (**superskalárnej**) ALU
  - **paralelnej** ALU

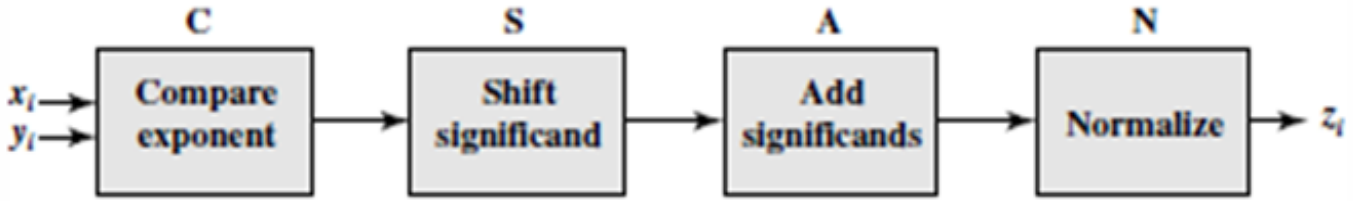


Použitie prúdovej

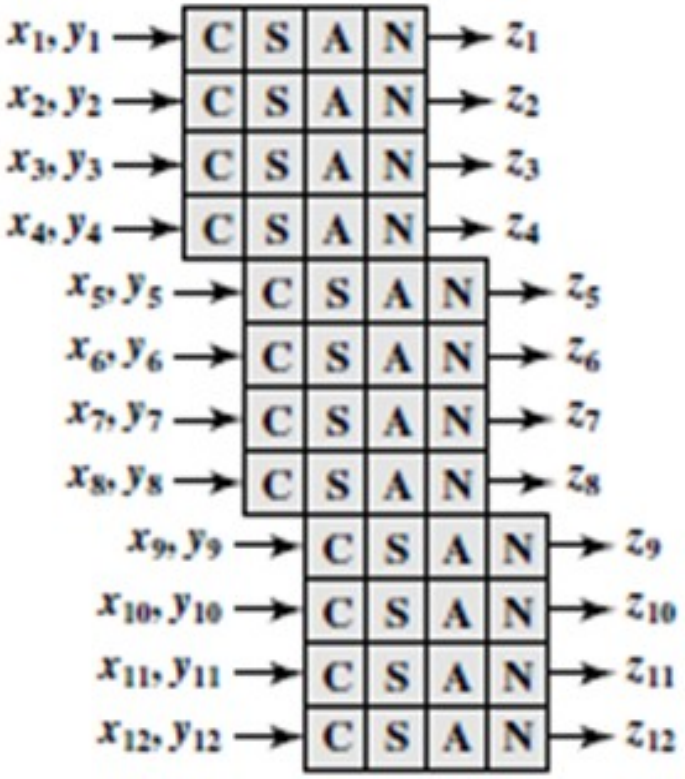


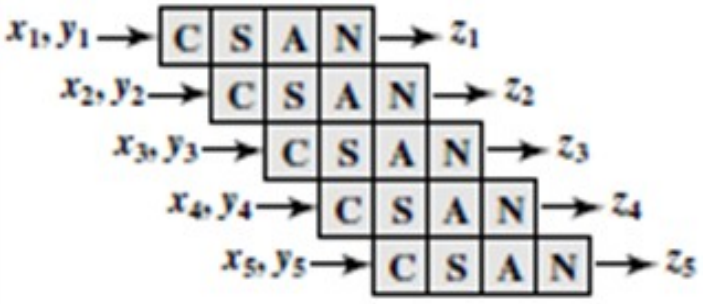
a paralelnej FPA ALU

Sekcie prúdovej FPA sčítačky



Paralená FPA sčítačka





Skalárna FPA sčítačka

Superskalárna FPA sčítačka