



ORGANIZACIJA RAČUNALNIKOV

4. Paralelizem na nivoju ukazov

Namen in cilji 4. poglavja:

- Paralelnost poskušamo **izkoristiti** na več različnih nivojih
 - **Cevovod** – osnovni koncept paralelizma na nivoju ukazov
 - Prednosti in omejitve paralelizma na nivoju ukazov oz. **cevovoda**
 - osnovna ideja: več večperiodnih ukazov hkrati !
 - **+ transparentnost**
 - - cev. nevarnosti - medsebojna odvisnost ukazov (operandov)
 - - zaporednost razmišljanja, algoritmov, podatk. struktur
 - **Cevovodne nevarnosti** in načini njihove odprave
 - **Špekulativno** (?!) izvajanje ukazov
 - Večizstavitveni – **superskalarni** procesorji
 - Paralelnost na nivoju **niti** (npr. „Hyper-Threading“)
- Današnji zmogljivi „desktop“ procesorji (npr. **Intel, AMD**)

4. Paralelizem na nivoju ukazov

Def: Večina ukazov v zaporedju (programu) se lahko izvede vzporedno...

Število ukazov, ki jih CPE izvede v 1 sek : $MIPS = \frac{f_{cpe}}{CPI \cdot 10^6}$

MIPS..Million Instr. Per Second

Hitrost CPE (št.ukazov/sek) je možno povečati na dva načina:

- povečamo f_{cpe} (hitrejši elementi)
- zmanjšamo CPI (z uporabo večjega števila elementov)

MIPS..Million Instr. Per Second
Je zelo **RELATIVNA enota** – primer:

CISC	RISC
ADD (A),(B),(C)	LDR R1,(A)
	LDR R2,(B)
	ADD R3,R1,R2
	STR R3,(C)

CISC = **1MIPS** RISC=**4MIPS**
Sicer čas lahko enak !!!!

CEVOVOD: način realizacije CPE, pri katerem se naenkrat izvaja več ukazov.

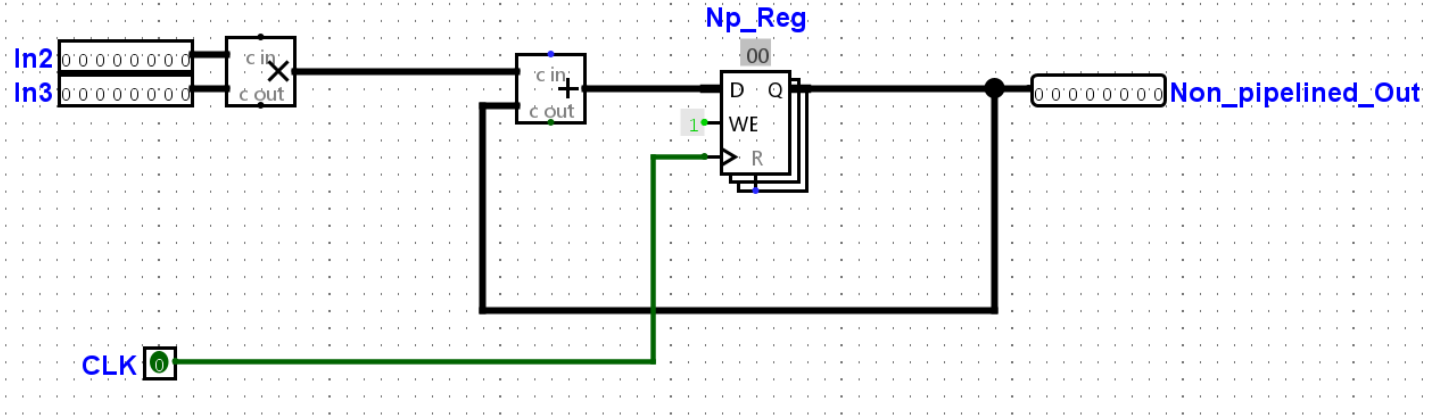
Glavna prednost cevovoda (poleg pohitritve) je

- **TRANSPARENTNOST:**
 - **ni posegov (v programe) !!!**

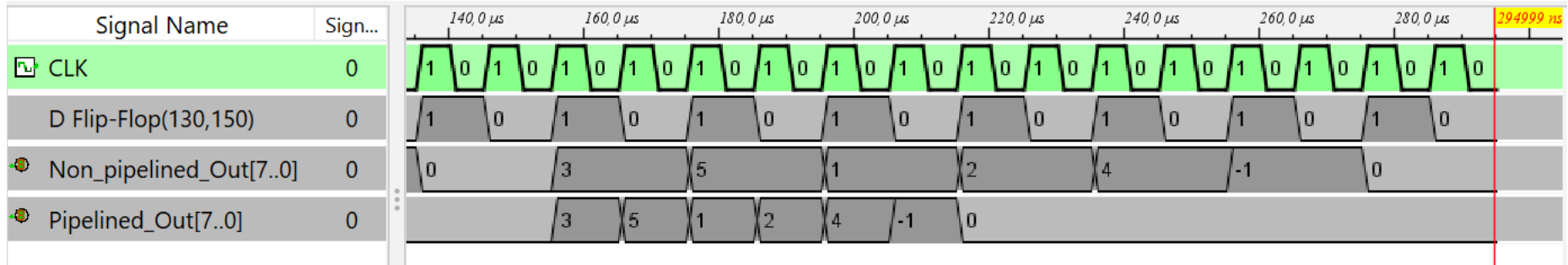
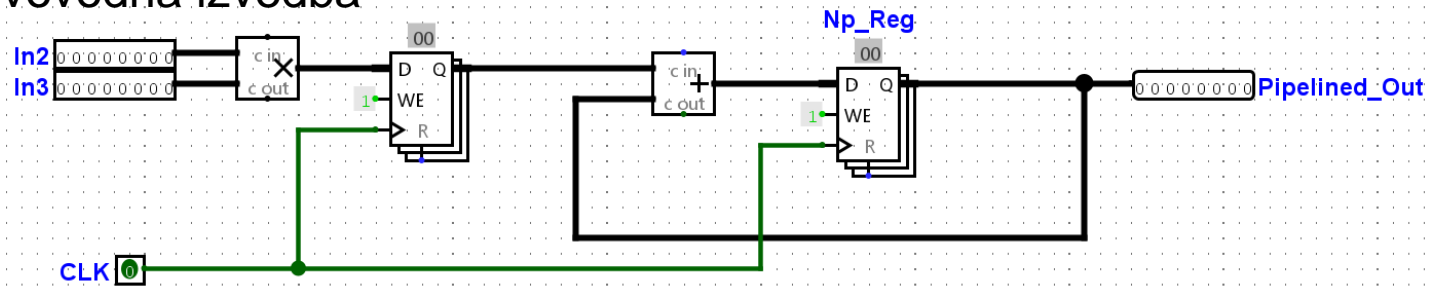
4. Paralelizem na nivoju ukazov

Ideja cevovodnega procesiranja – Logisim primer: $SUM = A \times B + SUM$

Necevovodna izvedba



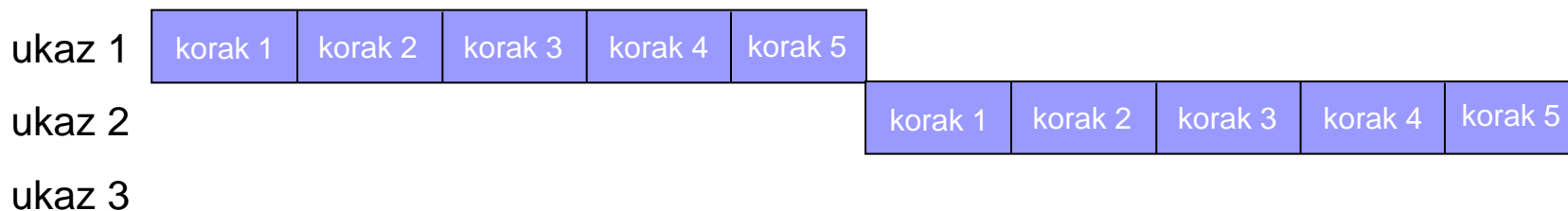
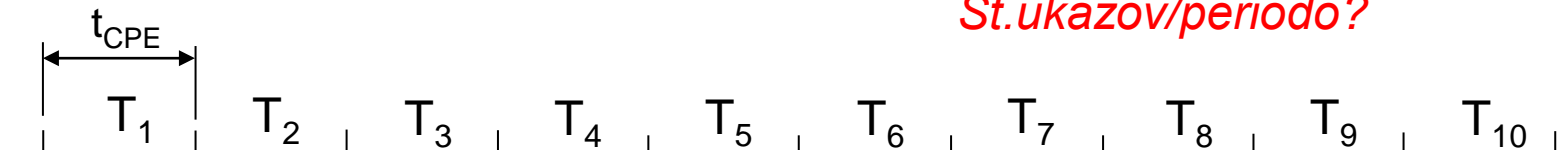
Cevovodna izvedba



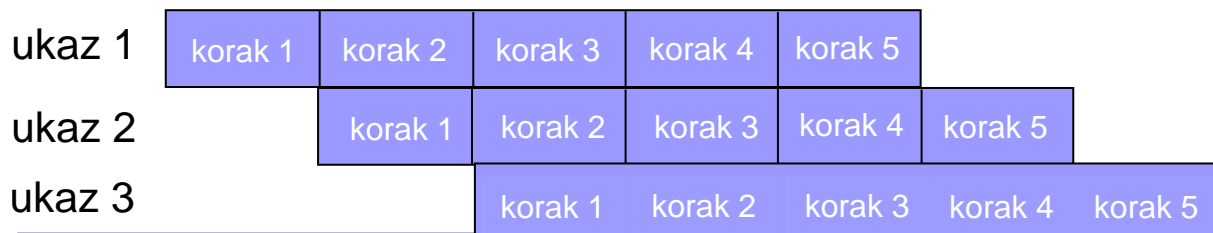
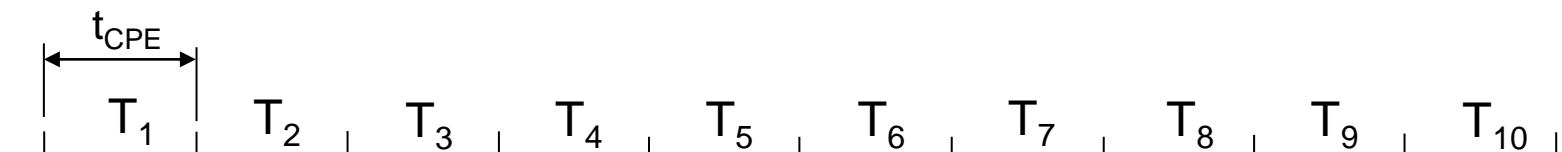
Izvrševanje treh ukazov pri necevovodni in cevovodni CPE s 5 stopnjami

Necevovodna CPE

Št. period za 3 ukaze ?
Št.ukazov/periodo?

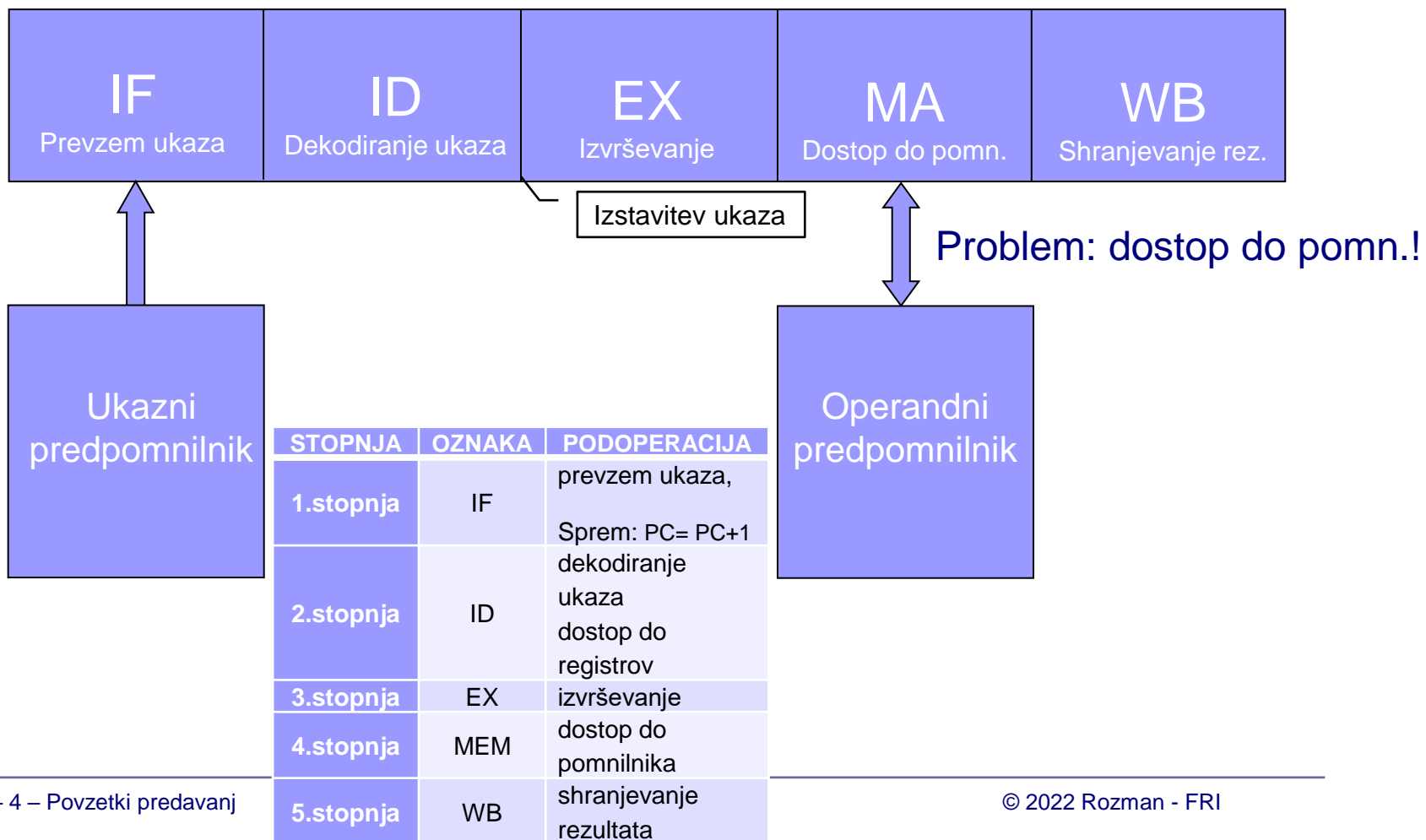


Cevovodna CPE (5 stopenj)



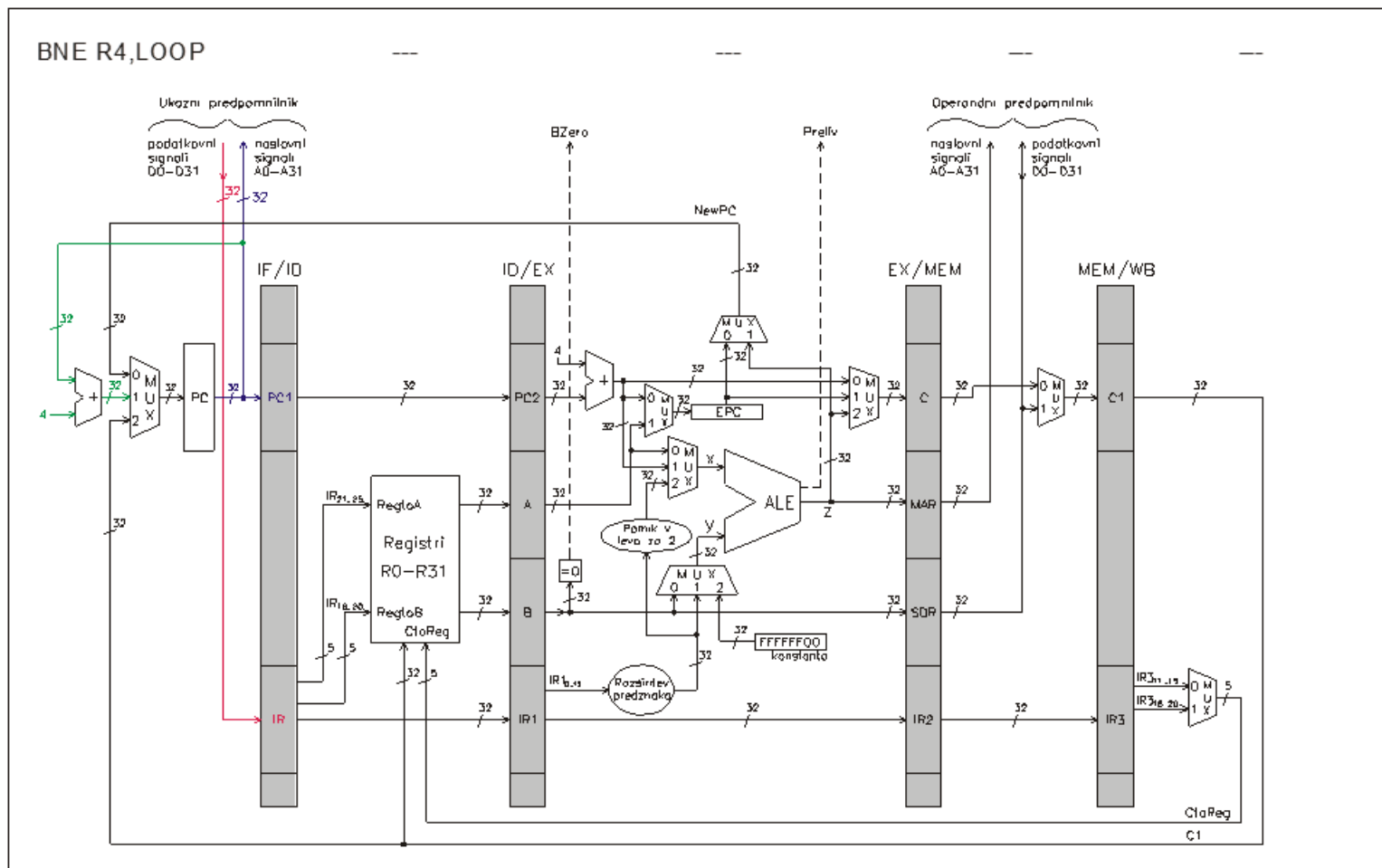
4.1 Zgradba cevovodne CPE

Primer cevovoda s 5 stopnjami (FRI-SMS,HiP):



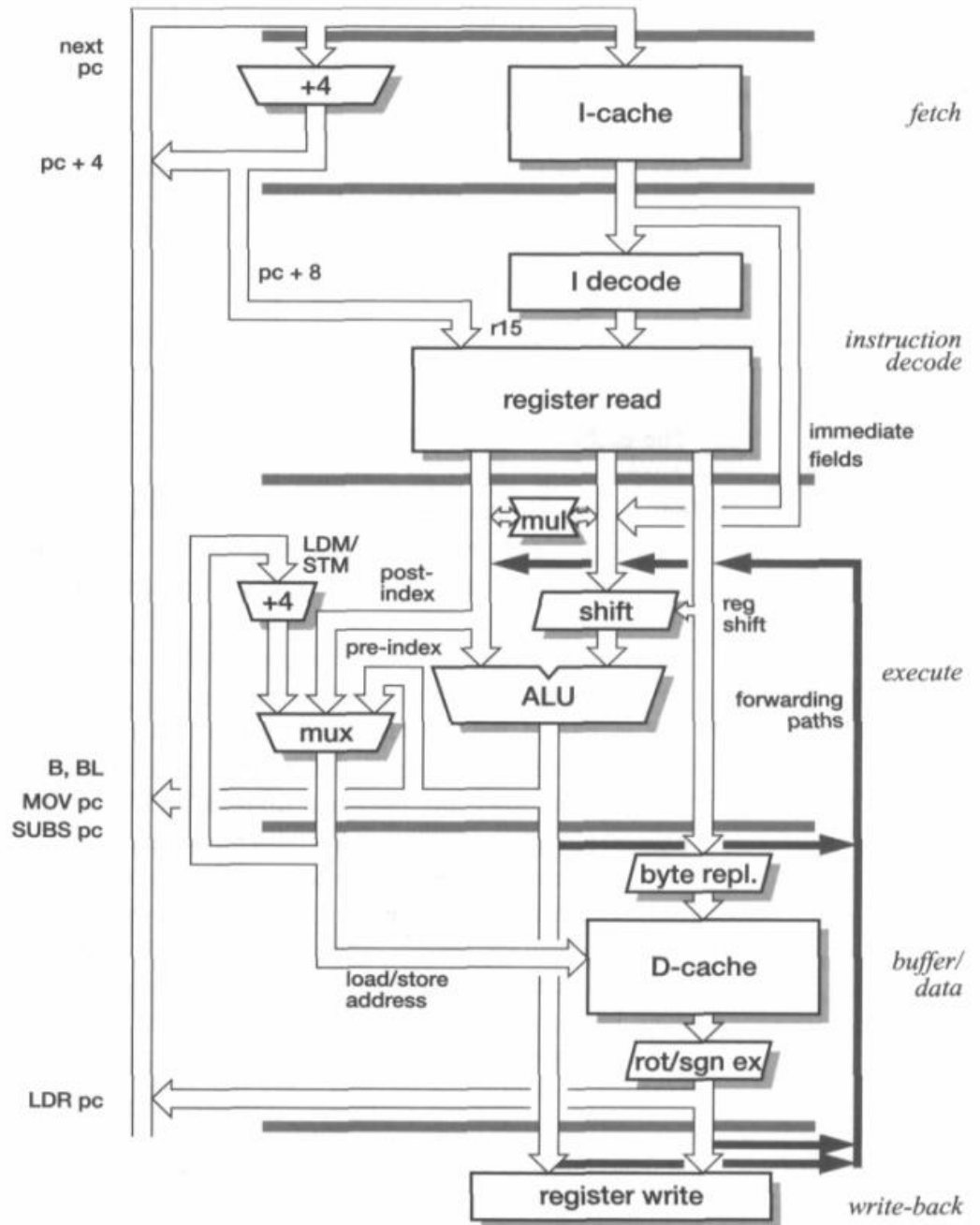
4.1 Zgradba cevovodne CPE

Primer cevovoda s 5 stopnjami (HiP):



4.1 Zgradba cevovodne CPE

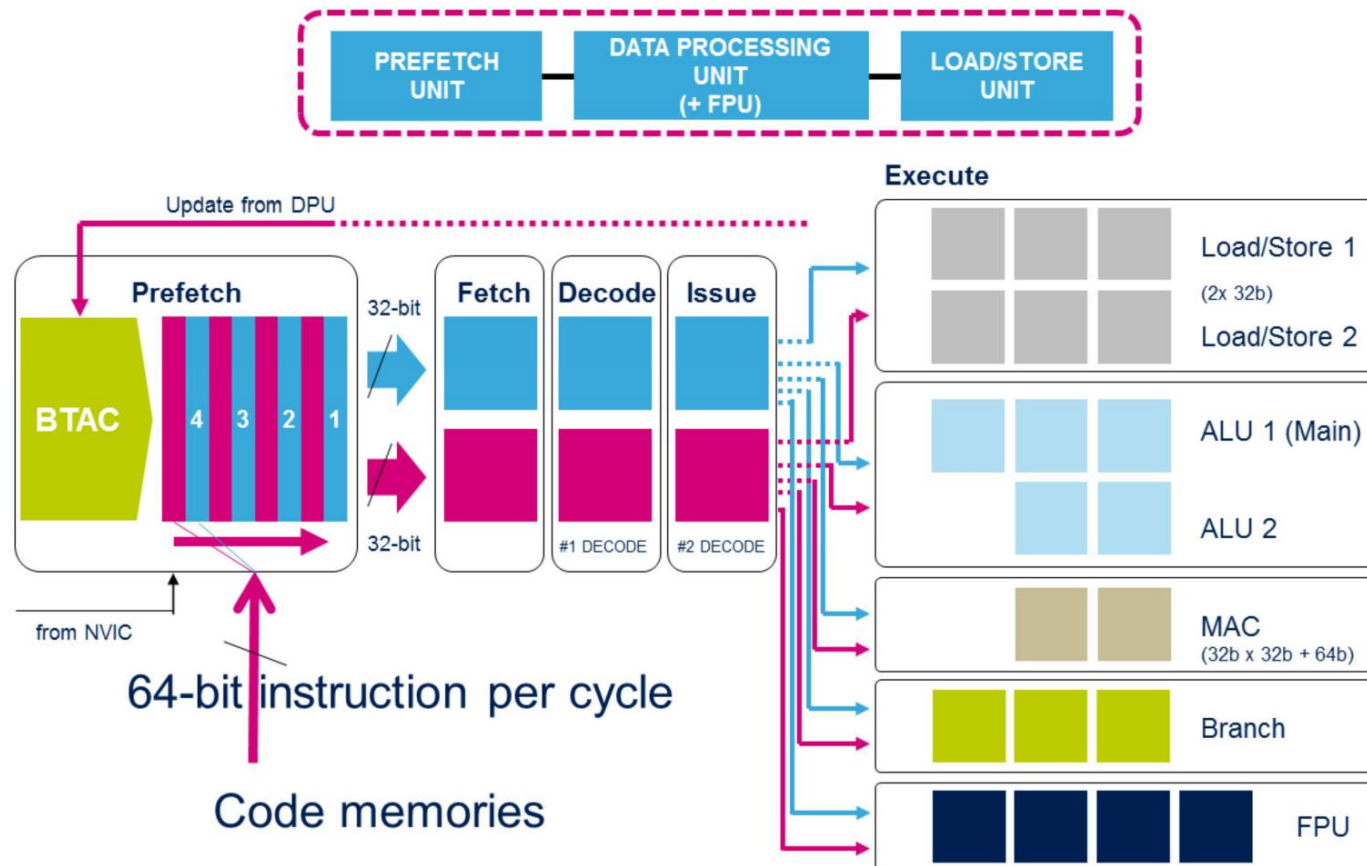
Primer cevovoda s 5 stopnjami
(FRI-SMS):



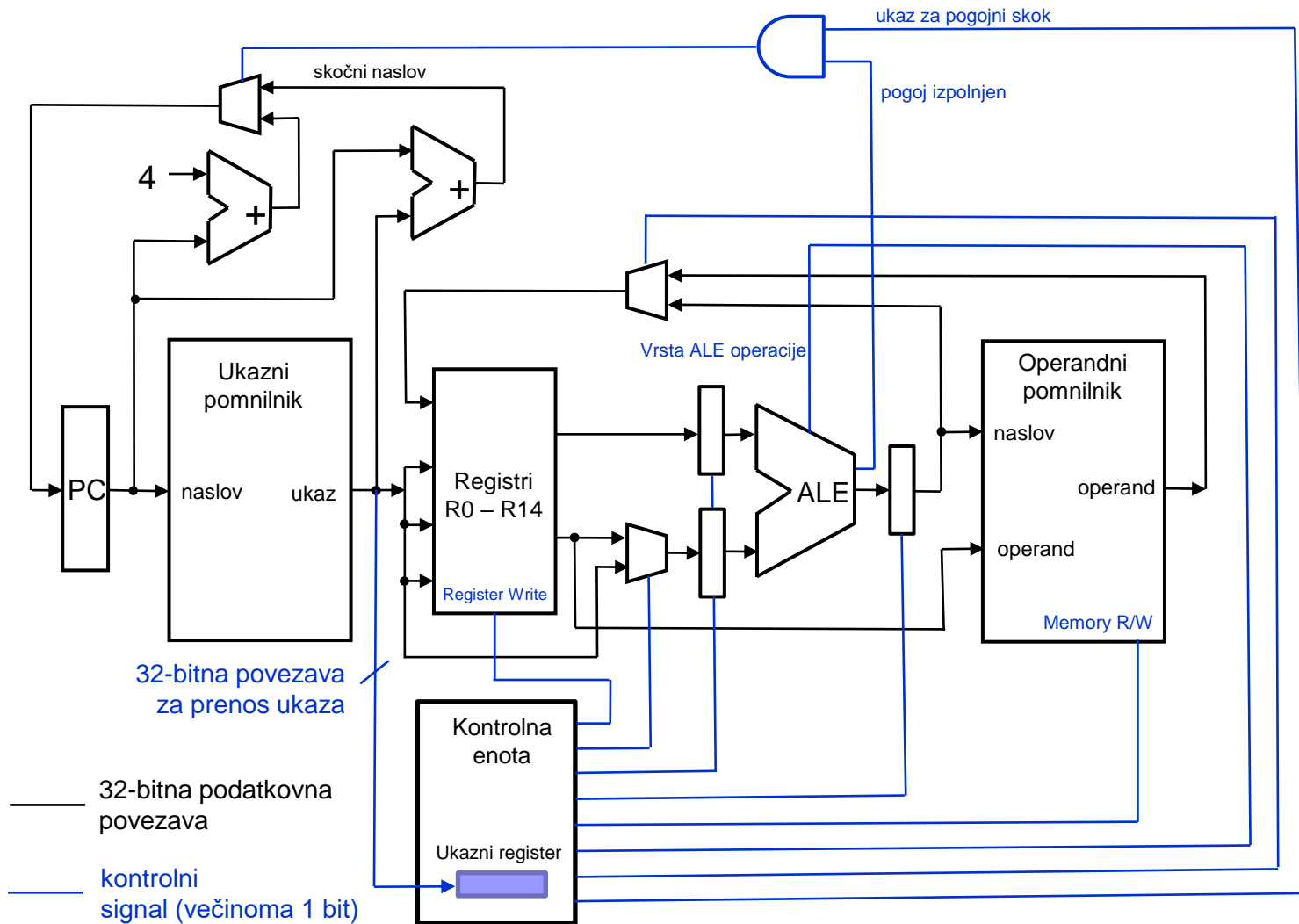
4.1 Zgradba cevovodne CPE

Primer dvo-izstavitvenega cevovoda s 6 stopnjami (ARM Cortex M7):

ARM Cortex-M7 → Dual-issue



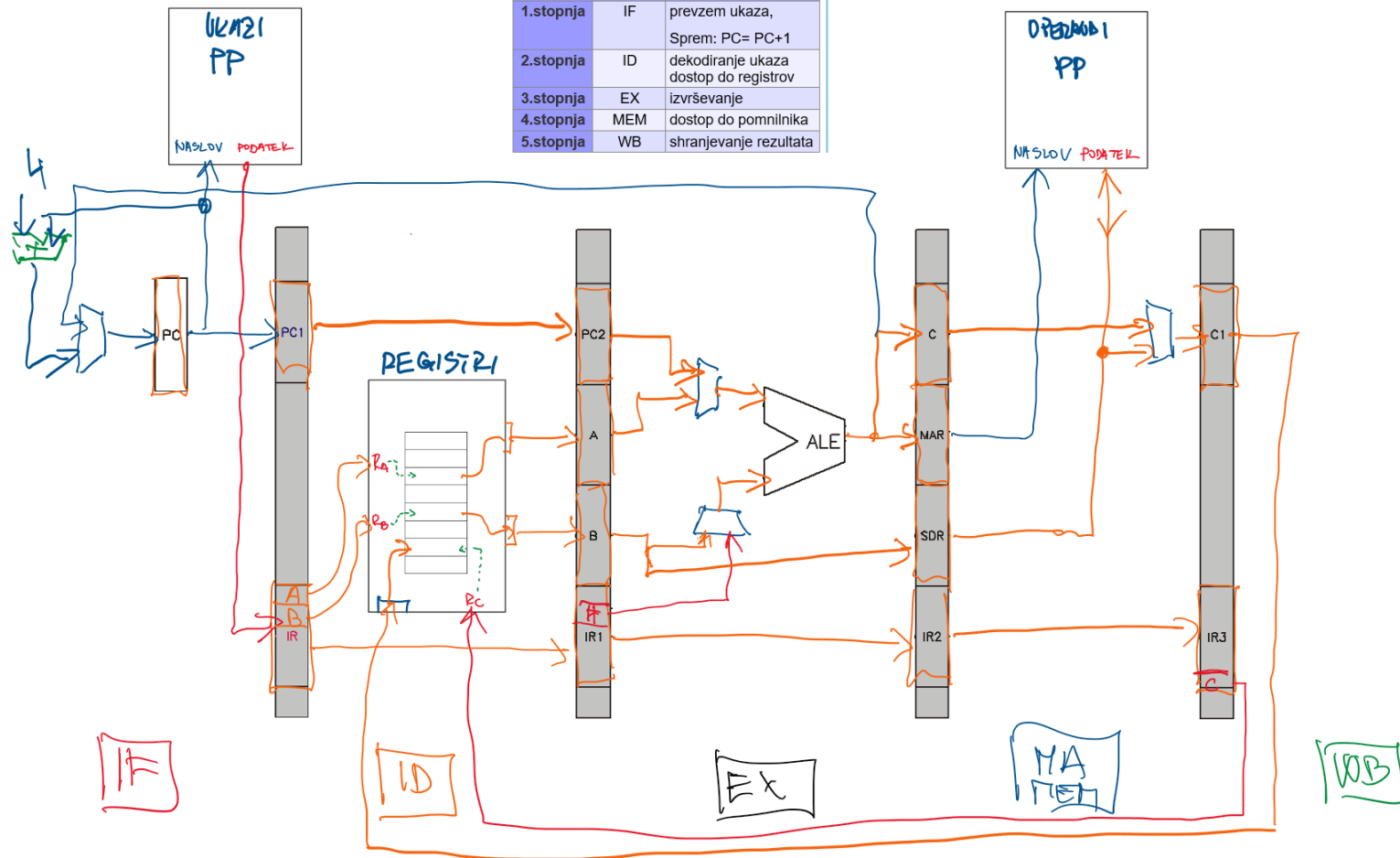
CPE ARM LEGv8 (RA) s podatkovno in kontrolno enoto ter kontrolnimi signali



4.1 Zgradba cevovodne CPE

Primer splošnega cevovoda s 5 stopnjami (FRI-SMS,HiP):

STOPNJA	OZNAKA	PODOPERACIJA
1.stopnja	IF	prevzem ukaza, Sprem: PC= PC+1
2.stopnja	ID	dekodiranje ukaza dostop do registrov
3.stopnja	EX	izvrševanje
4.stopnja	MEM	dostop do pomnilnika
5.stopnja	WB	shranjevanje rezultata



4.1 Zgradba cevovodne CPE

Problem dostopa do pomnilnika:

- N-kratno povečanje dostopov
- 2 dostopa v enem ukazu

Problemi pri izvajanju ukazov

- čas izvajanja ukaza se v cevovodni CPE podaljša – vzroka:
 - strojne narave
 - prenos med stopnjami (vmesni registri),
 - uravnoteženost (skupna perioda)
 - programske narave
 - cevovodne nevarnosti

4.2 Cevovodne nevarnosti

„takrat, kadar ukazi v izvajanju niso med seboj neodvisni...“

Glede na vzrok ločimo:

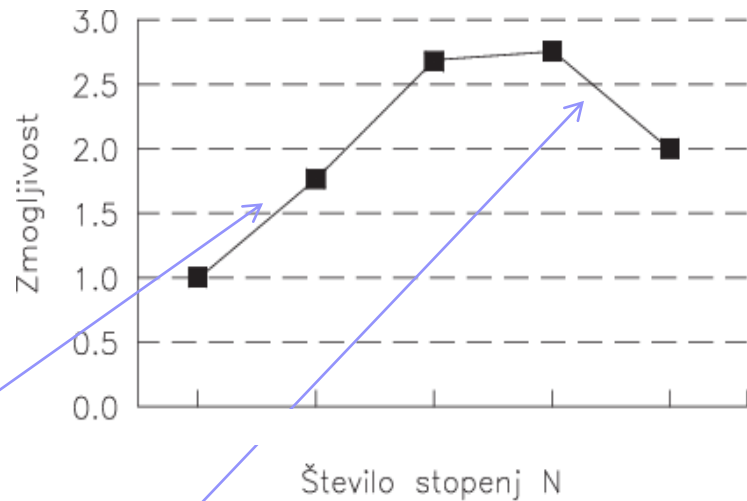
- **strukturne**
- **podatkovne (operandne)**
- **kontrolne**

S povečevanjem števila stopenj

- se povečuje število ukazov, ki se hkrati izvršujejo (manjšata se **CPI** in t_{CPE})
- se povečuje pogostost cevovodnih nevarnosti (hkrati v cevovodu več ukazov) in učinkovitost cevovoda pada (**CPI** se povečuje).

Cevovodne nevarnosti moramo zaznati in reševati:

- SW (prevajalnik):
 - vstavlja NOP, preureja vrstni red
- HW (dodatna logika):
 - zaklenitev
 - na srečo tudi učinkovitejše rešitve za posamezne vrste nevarnosti



4.2.1 Strukturne nevarnosti

Def: „kadar več stopenj cevovoda rabi eno enoto, ki lahko izvede le eno podoperacijo naenkrat.“:

- registri, ALE, pomnilnik, predpomnilnik

Odpravimo :

- **z bolj zmogljivim in ločenim predpomnilnikom („Harvardska arhitektura“)**
- **večje število FE**
- **stopnja EX v obliki cevovoda**

Zakaj bolj previdno z odpravljanjem teh nevarnosti?

- **drage rešitve**
- **Odločitve pri načrtovanju**

Praksa pokaže, da so te nevarnosti „... še najmanj škodljive ...“

4.2.2 Podatkovne (operandne) nevarnosti

„Ukaz potrebuje operand, ki še ni dostopen“

Imamo zaporedje dveh ukazov:

- Ukaz 1: `sub r3, r4, r5`
- Ukaz 2: `add r1, r6, r3`



3 skupine podatkovnih nevarnosti :

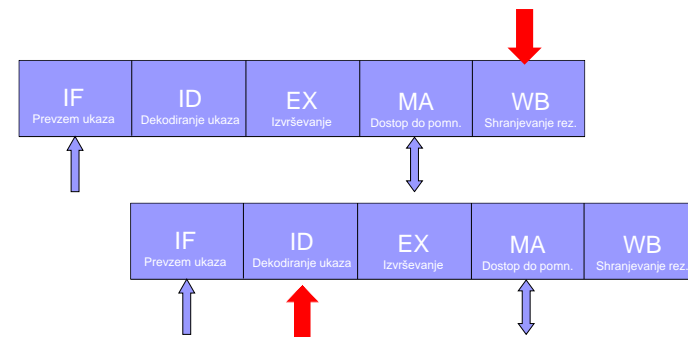
- **RAW** – Read After Write
- **WAR** – Write After Read
- **WAW** – Write After Write
- ~~RAR~~ ?

RAW (read after write)

- Ukaz 2 bere operand preden ga ukaz 1 shrani, zato prebere napačno vrednost.
 - To je najpogostejša vrsta podatkovne nevarnosti in jo v večini primerov lahko odpravimo s premoščanjem.

L2: sub **r3**, r4, r5
 add r1, r6, **r3**

RAW



Ukaz add bere operand v registru r3 preden ga ukaz sub shrani, zato prebere napačno vrednost

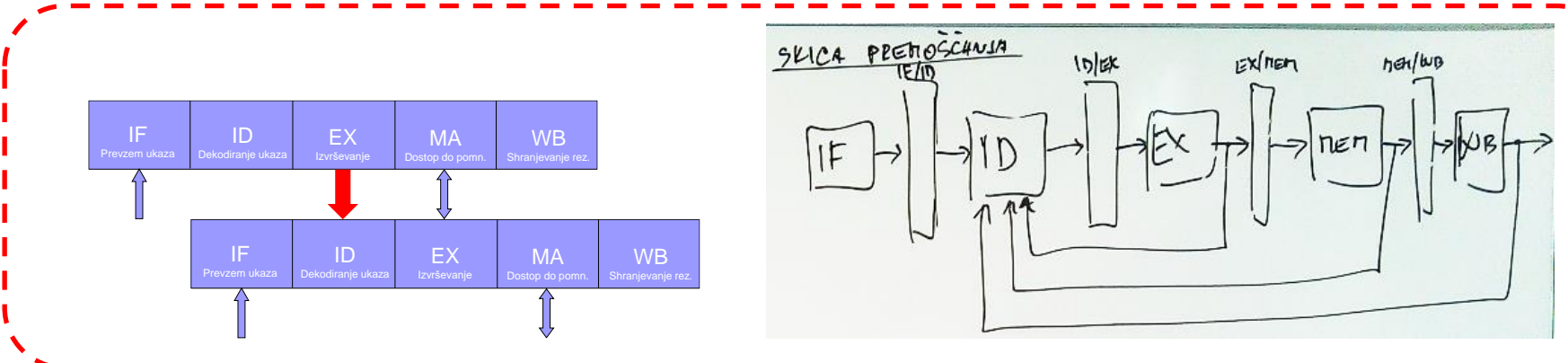
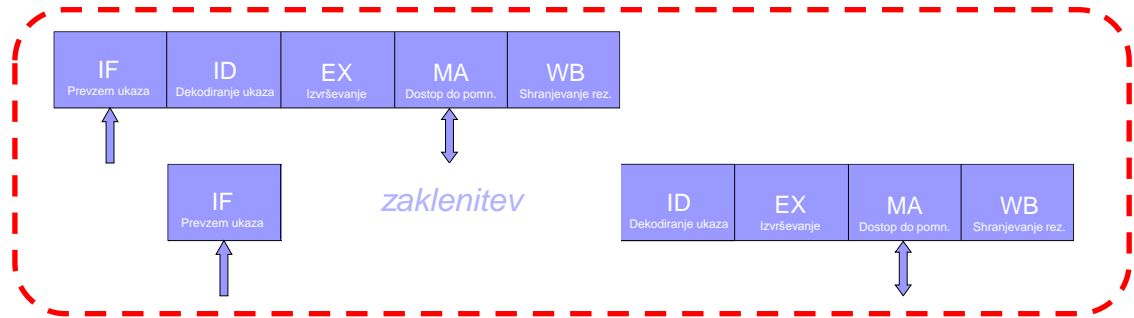
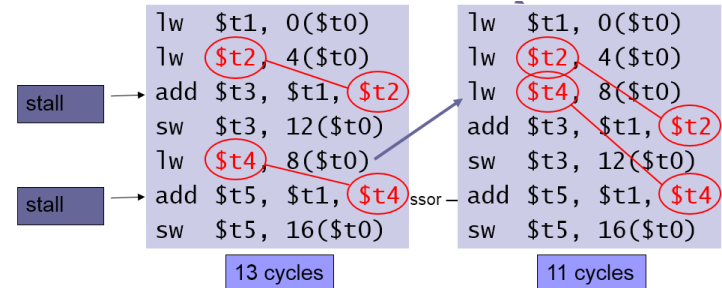
Rešitve :

- *cevovodno razvrščanje (prevajalnik)*
- *zaklenitev cevovoda*
- *premoščanje*

RAW (read after write) :

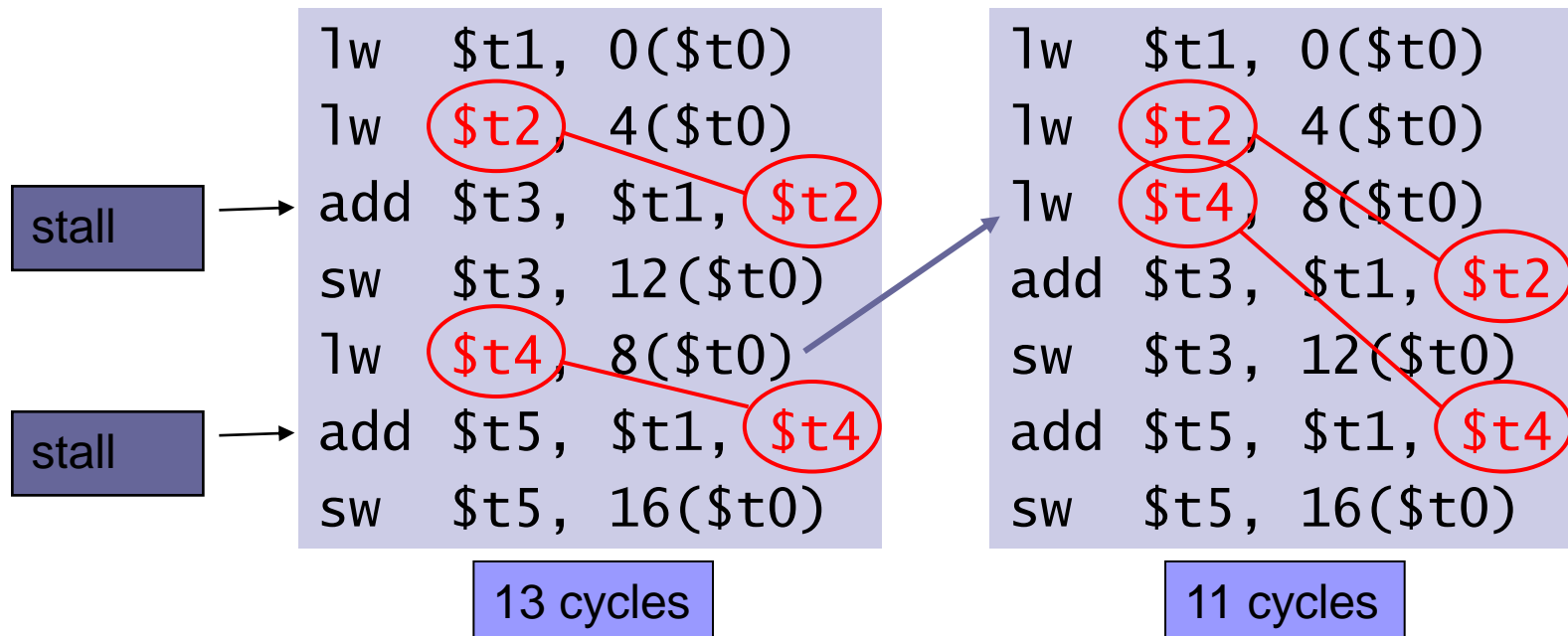
Rešitve :

- cevovodno razvrščanje (prevajalnik)
- zaklenitev cevovoda
- premoščanje



Cevovodno razvrščanje

- Spremeni vrstni red, da bo manj zaklenitev cevovoda
- C programska koda za : $A = B + E$; $C = B + F$;



Potrebno znanje o delovanju konkretnega cevovoda !

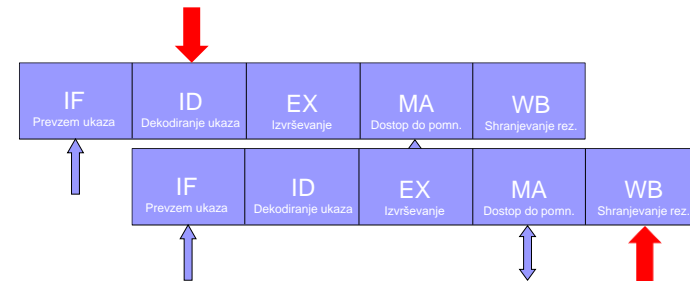
(npr. ARM-Cortex M7 ne kaže teh razlik v internih pomnilnikih)

WAR (write after read):

- Ukaz 2 piše v register, preden ukaz 1 prebere vsebino registra. Ukaz 1 dobi tako novo vrednost namesto stare.

```
L2:      mov r3, r7
         lw  r8, (r3)
         add r3, r3, 4
         lw  r9, (r3)
         ble r8, r9, L2
```

WAR



WAR (write after read) Ukaz *add* piše v register r3, preden ukaz *lw* prebere vsebino registra. Ukaz *lw* dobi tako novo vrednost namesto stare

Nastopa le :

- v nekaterih izvedbah cevovodov in
- pri dinamičnem razvrščanju ukazov

WAW (write after write) :

- Ukaz 2 piše v register, preden vanj piše ukaz 1. Vrstni red pisanja v register je drugačen kot v programu, zato ima register vrednost od ukaza 1 namesto od 2.

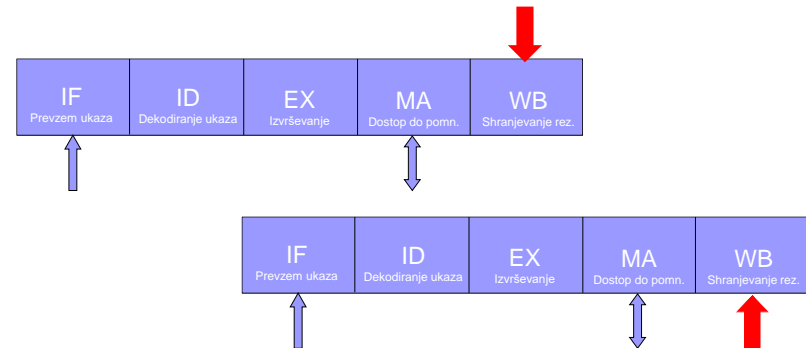
L2 :

```

mov  r3, r7
lw   r8, (r3)
add  r3, r3, 4
lw   r9, (r3)
ble  r8, r9, L2

```

WAW



WAW (write after write) Ukaz *add* piše v register, preden vanj piše ukaz *mov*. Vrstni red pisanja v register je drugačen kot v programu, zato ima register vrednost od ukaza *mov* namesto od *add*.

Nastopa le:

- v nekaterih izvedbah cevovodov (več pisalnih stopenj) in
- pri dinamičnem razvrščanju ukazov

4.2.3 Kontrolne nevarnosti

Kontrolne nevarnosti se pojavijo pri vseh ukazih, ki spremenijo vsebino programskega števec PC drugače kot po običajnem pravilu $PC \leftarrow PC + 4(1)$.

To se dogaja pri kontrolnih ukazih:

- **pogojni skoki**
- **brezpogojni skoki, klici, vrnitve (krajše „skoki“)**

Slednji so ugodnejši, ker prej zremo novi naslov.

Pri **pogojnih zremo kasneje** in ali se sploh skok izvede!

Delovanje pogojnih skokov (ob napovedi neizpolnjenega pogoja):

- če je pogoj za skok izpolnjen, se v PC prenese skočni ali ciljni naslov (v primeru na sliki naslov ukaza 5); že naloženi ukazi se razveljavijo, prevzemati se začno ukazi z novega naslova...
- če pogoj ni izpolnjen, se izvršijo ukazi, ki so že v cevovodu – čakanja ni.

Meritev na programih (10 programov iz Spec92):

- zaradi kontrolnih nevarnosti se CPI poveča za 21.8%!

4.2.3 Kontrolne nevarnosti

Izvajanje ukaza za pogojni skok

ukaz 1 - pogojni skok na ukaz 5

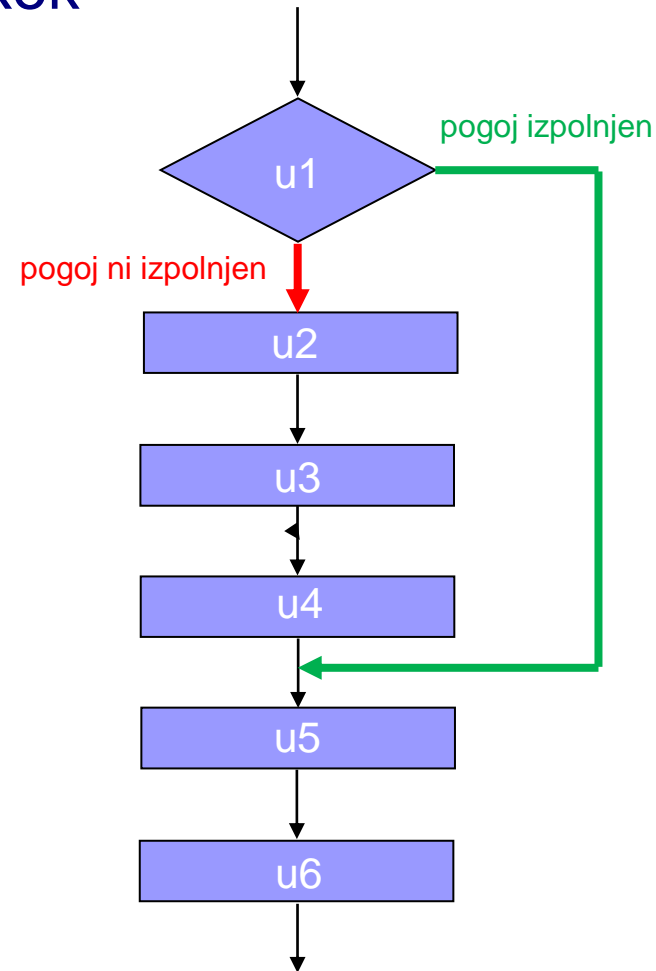
ukaz 2

ukaz 3

ukaz 4

ukaz 5

ukaz 6



4.2.3 Kontrolne nevarnosti

Pogoj ni izpolnjen

ukaz 1 - pogojni skok na ukaz 5

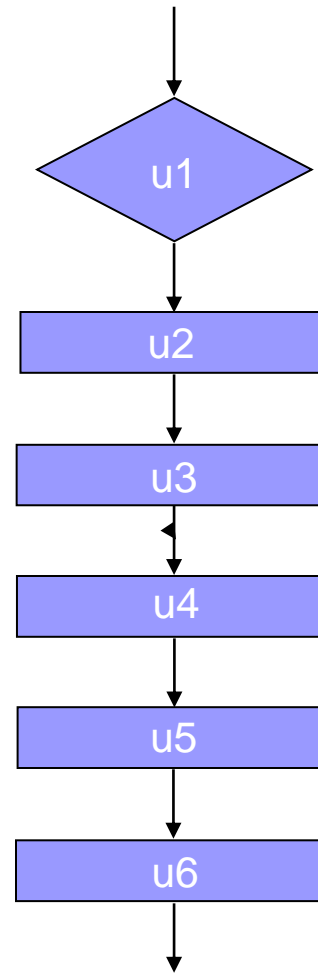
ukaz 2

ukaz 3

ukaz 4

ukaz 5

ukaz 6



4.2.3 Kontrolne nevarnosti

Pogoj je izpolnjen

ukaz 1 - pogojni skok na ukaz 5

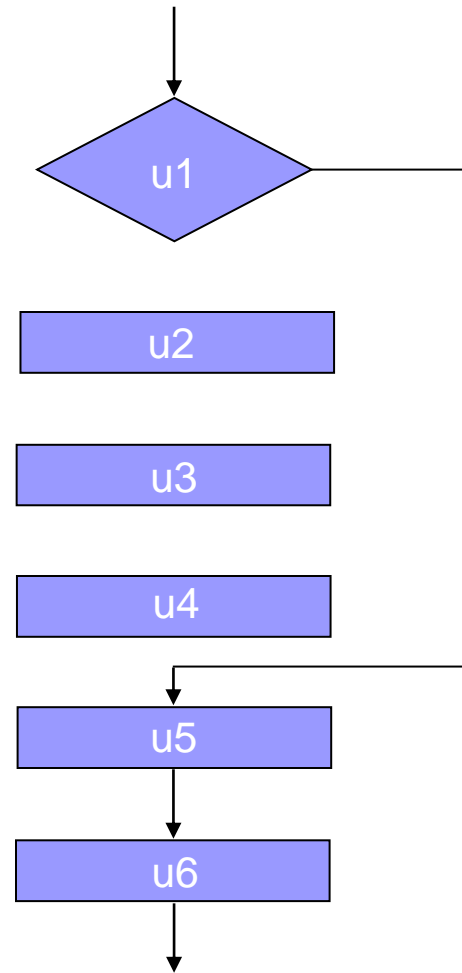
ukaz 2

ukaz 3

ukaz 4

ukaz 5

ukaz 6



■ Najpreprostejša rešitev te nevarnosti:

- v urini periodi, ko stopnja EX spremeni PC, v predhodne stopnje vstavimo **mehurčke**.

■ Skočna zakasnitev: čakanje toliko urinih period, kolikor je ?.

Primer zakasnitve pri statični predikciji :

urine periode	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈	T ₉
ukaz 1	IF ₁	ID ₁	EX ₁	MA ₁	WB ₁				
ukaz 2		IF ₂	ID₂						
ukaz 3			IF₃						
ukaz 5				IF ₅	ID ₅	EX ₅	MA ₅	WB ₅	
ukaz 6					IF ₆	ID ₆	EX ₆	MA ₆	WB ₆

predpostavimo „neizpolnjeni pogoj“

4.2.3.1 Zmanjšanje zakasnitev pri kontrolnih nevarnosti

Pri gradnji cevovoda naredimo, da:

- se preverjanje pogoja za skok izvaja čim bolj na začetku cevovoda, saj je tako manj ukazov, ki bi jih morali zavreči
- se tudi izračun skočnega naslova izvaja čim bolj na začetku cevovoda

Skočne zakasnitve pa se lahko zmanjšajo ali odpravijo tudi z uporabo predikcije – napovedi:

- izpolnitve pogoja za skok in
- skočnega naslova

Rešitve (strojne ali strojne+programske) delimo v dve skupini:

- statična predikcija:
 - predpostavka neizpolnjenega pogoja („branch not taken“)
 - z zakasnjnimi skoki („delayed branch“)
- dinamična predikcija (med delovanjem)

4.2.3.1.1 Statična predikcija neizpolnjenega pogoja („branch not-taken“)



■ Primer ARM 9:

□ Statična napoved neizpolnjenega pogoja

□ Bxx (npr. BNE):

■ Pogoju xx neizpolnjen :

□ ?? t_{cpe}

■ Pogoju xx izpolnjen :

□ ?? t_{cpe}

□ B (npr. BNE):

■ Brezpogojni skok :

□ ?? t_{cpe}

urine periode	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈	T ₉
ukaz 1	IF ₁	ID ₁	EX ₁	MA ₁	WB ₁				
ukaz 2		IF ₂	ID₂						
ukaz 3			IF₃						
ukaz 5				IF ₅	ID ₅	EX ₅	MA ₅	WB ₅	
ukaz 6					IF ₆	ID ₆	EX ₆	MA ₆	WB ₆

A box labeled "predpostavimo „neizpolnjeni pogoj“" has an arrow pointing to the ~~ID₂~~ cell in the table.

4.2.3.1.1 Statična predikcija z zakasnjnimi skoki („delayed branch“)

Skuša prevajalnik (med prevajanjem) napovedati izid skoka:

- napoved se ne spreminja več („statična“)

Skočne reže :

- ukazi, ki sledijo skoku so v t.i. skočnih režah
- št. skočnih rež je enako številu stopenj cevovoda pred aktivno (EX) stopnjo (2 pri 5 stopenjskem)

Vstavitev ukazov v skočne reže:

- ukazi ne smejo vplivati na izid skoka
- če ni primernih, vstavi NOP-e

Primer: 2 skočni reži, kar ustreza našemu primeru 5-stopenjskega cevovoda, kjer je EX tretja stopnja:

	<i>LD</i>	<i>R1, #100</i>		<i>LD</i>	<i>R1, #100</i>	
<i>ZN</i>	<i>SUB</i>	<i>R1, #4</i>		<i>ZN</i>	<i>SUB</i>	<i>R1, #4</i>
	<i>ADD</i>	<i>R2, R4, R5</i>	→	<i>BEQ</i>	<i>R1, ZN</i>	<i>pogojni skok</i>
	<i>SUB</i>	<i>R7, R6, R5</i>		<i>ADD</i>	<i>R2, R4, R5</i>	<i>skočna reža 1</i>
	<i>BEQ</i>	<i>R1, ZN</i>		<i>SUB</i>	<i>R7, R6, R5</i>	<i>skočna reža 2</i>
	<i>AND</i>	<i>R8, R6, R5</i>		<i>AND</i>	<i>R8, R6, R5</i>	

- Ker se ukazi, ki se vstavijo v skočne reže, vedno izvedejo, ne smejo vplivati na izid skoka.
- Če prevajalnik v programu ne najde ukazov, ki bi jih lahko vstavil v skočne reže, ...

	<i>LD</i>	<i>R1, #100</i>		<i>LD</i>	<i>R1, #100</i>	
<i>ZN</i>	<i>SUB</i>	<i>R1, #4</i>		<i>ZN</i>	<i>SUB</i>	<i>R1, #4</i>
	<i>ADD</i>	<i>R2, R4, R5</i>	→	<i>SUB</i>	<i>R7, R6, R5</i>	
	<i>SUB</i>	<i>R7, R6, R5</i>		<i>BEQ</i>	<i>R7, ZN</i>	
	<i>BEQ</i>	<i>R7, ZN</i>		<i>ADD</i>	<i>R2, R4, R5</i>	<i>skočna reža 1</i>
	<i>AND</i>	<i>R8, R6, R5</i>		<i>NOP</i>		<i>skočna reža 2</i>
				<i>AND</i>	<i>R8, R6, R5</i>	

4.2.3.1.1 Statična predikcija z zakasnenimi skoki („delayed branch“)

Prednosti, slabosti:

- + preprosta
- + učinkovita pri krajših cevovodih
- spremeni se program
- problem pri daljših cevovodih

Meritev na programih:

- zaradi kontrolnih nevarnosti se CPI poveča za **21.8%**!
- ob uporabi **statične predikcije** (zak.skoki) se CPI poveča le še za **8.7%**!

Primer pohitritve kode z uporabo cevovodnega razvrščanja, zakasnenih skokov in razpeljave zank:

```
for (i=1; i<=1000; i++)
  x[i] = x[i] + s;
```



Loop:	LD	F0, 0(R1)	;F0 - array element
	ADDD	F4, F0, F2	;add scalar in F2
	SD	0(R1), F4	;store result
	SUBI	R1, R1, #8	;decrement pointer ;8 bytes (per double)
	BENZ	R1, Loop	;branch R1 != zero

■ Cevovodno razvrščanje

Without any scheduling			Scheduled		
		Cycles			Cycles
Loop: LD	F0, 0(R1)	1	Loop: LD	F0, 0(R1)	1
stall		2	stall		2
ADDD	F4, F0, F2	3	ADDD	F4, F0, F2	3
stall		4	SUBI	R1, R1, #8	4
stall		5	BENZ	R1, Loop	5
SD	0(R1), F4	6	SD	8(R1), F4	6
SUBI	R1, R1, #8	7			
BENZ	R1, Loop	8			
stall		9			
9 clock cycles per element			6 clock cycles per element		

Vir: <http://web.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/loopUnrolling.html>

Primer pohitritve kode z uporabo cevovodnega razvrščanja, zakasnenih skokov in razpeljave zank:

```
for (i=1; i<=1000; i++)
    x[i] = x[i] + s;
```



Loop:	LD	F0, 0(R1)	:F0 - array element
	ADDD	F4, F0, F2	:add scalar in F2
	SD	0(R1), F4	:store result
	SUBI	R1, R1, #8	:decrement pointer :8 bytes (per double)
	BENZ	R1, Loop	:branch R1 != zero

Razvezava zanke + razvrščanje

Without any scheduling				Scheduled			
Loop:	LD	F0, 0(R1)	1	Loop:	LD	F0, 0(R1)	1
	stall		2		LD	F6, -8(R1)	2
	ADDD	F4, F0, F2	3		LD	F10, -16(R1)	3
	stall		4		LD	F14, -24(R1)	4
	stall		5		ADDD	F4, F0, F2	5
	SD	0(R1), F4	6		ADDD	F8, F6, F2	6
	LD	F6, -8(R1)	7		ADDD	F8, F6, F2	7
	stall		8		ADDD	F16, F14, F2	8
	ADDD	F8, F6, F2	9		SD	0(R1), F4	9
	stall		10		SD	-8(R1), F8	10
	stall		11		SD	-16(R1), F12	11
	SD	-8(R1), F8	12		SUBI	R1, R1, #32	12
	LD	F10, -16(R1)	13		BENZ	R1, Loop	13
	stall		14		SD	8(R1), F16	14
	ADDD	F12, F10, F2	15				
	stall		16				
	stall		17				
	SD	-16(R1), F12	18				
	LD	F14, -24(R1)	19				
	stall		20				
	ADDD	F16, F14, F2	21				
	stall		22				
	stall		23				
	SD	-24(R1), F16	24				
	SUBI	R1, R1, #32	25				
	BENZ	R1, Loop	26				
	stall		27				

27 clock cycles per iteration:
27/4 = **6.8 clock cycles per element**

14 clock cycles per iteration:
14/4 = **3.5 clock cycles per element**

4.2.3.1.2 Dinamična predikcija skokov

Napoved se spreminja med delovanjem

- uporabi se informacija o delovanju skoka do sedaj

Uporablja se več vrst dinamične predikcije:

- 1-bitna prediktorska tabela:
 - (branch prediction (history) table, prediction buffer)
- 2-bitna prediktorska tabela
- korelacijski prediktor
- turnirski prediktor

4.2.3.1.2.1 1-bitna prediktorska tabela

0x4000003c: BEQZ R2, *label*

LSB	Predictor
⋮	⋮
111100	0
⋮	⋮

- najenostavnejša oblika
- 1-bitni pomnilnik :
 - naslov:
 - spodnji biti naslovov skočnih ukazov (nepopolni naslov)
 - vsebina:
 - izid zadnjega skoka s tem naslovom (1..izpolnjen, 0..neizpolnjen pogoj)

Delovanje:

- prevzem ukaza v skladu s prediktorskim bitom
- če napoved napačna:
 - se prevzeti ukazi zavržejo
 - prediktorski bit invertira

Zanka:

- dve obvezni zgrešitvi: na začetku in koncu
- zato boljše večbitna tabela

4.2.3.1.2.2 2-bitna prediktorska tabela

2-bitna prediktorska tabela:

- najenostavnejša oblika
- 2-bitni pomnilnik :
 - naslov:
 - spodnji biti naslovov skočnih ukazov (nepopolni naslov)
 - vsebina je lahko 0-3:
 - 2-3: napoved izpolnjenega pogoja
 - 0-1: napoved neizpolnjenega pogoja

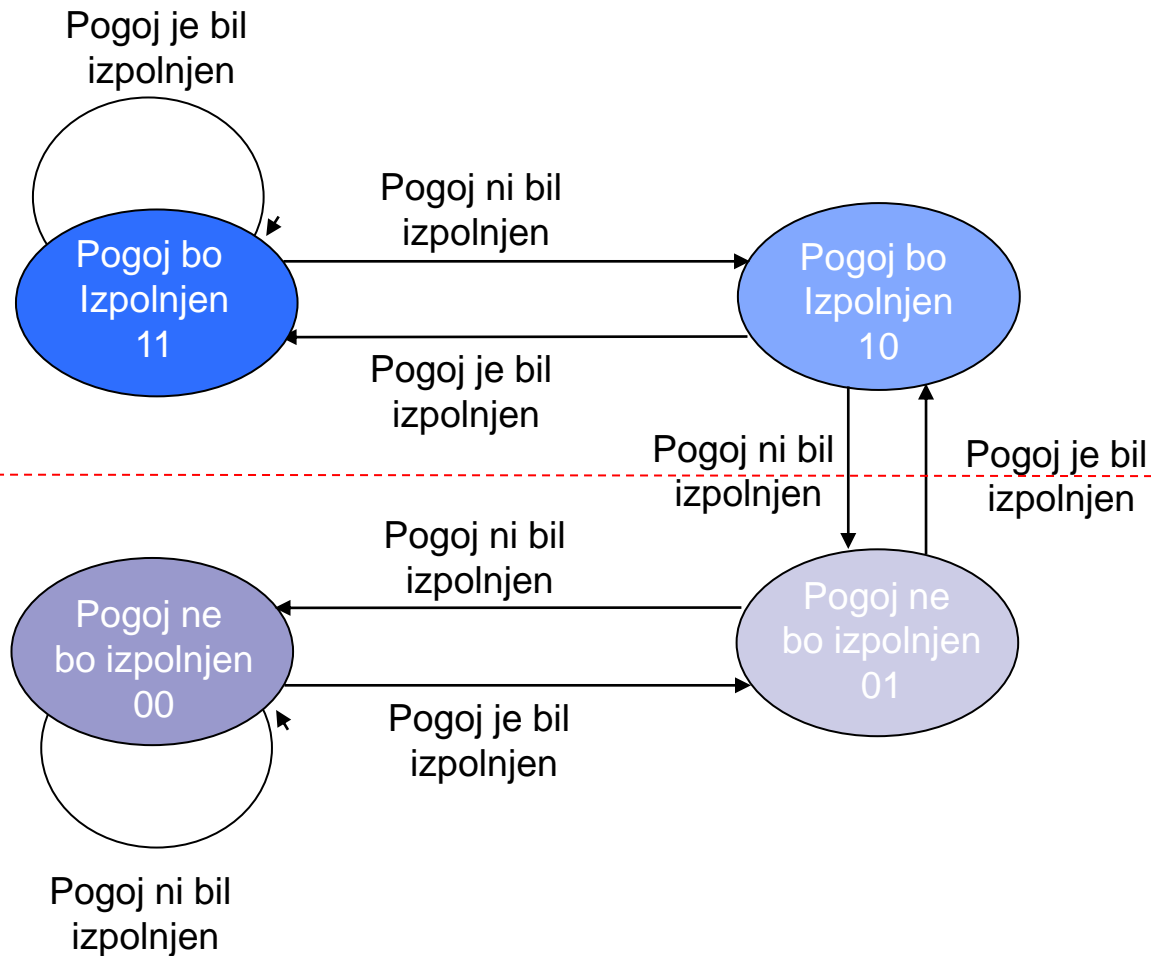
Delovanje:

- prevzem ukaza v skladu s predikcijo
- če pogoj izpolnjen, se vrednost poveča za 1 (maks.vr. 3)
- če pogoj ni izpolnjen, se vrednost zmanjša za 1 (min.vr. 0)

Meritev na programih:

- 2-bitna tabela s 4096 naslovi: **7%** napačnih napovedi

Prehajanje stanj pri 2-bitnem prediktorju



4.2.3.1.2.3 Korelacijski prediktor (m,n) – 2. stopenjski adaptivni pred.

Korelacijski prediktor (m,n) uporabi:

- informacijo o (lokalnem) skočnem ukazu
 - n-bitna prediktorska tabela
- informacijo o globalnem obnašanju zadnjih m-skočnih ukazov

Tipični korelacijski prediktor (2,2): →

- štiri (2^2) 2-bitne prediktorske table 0,1,2,3
- glede na izpolnjenost pogoja zadnjih dveh skokov se uporabi različna napoved (ena izmed tabel 0,1,2,3).

Delovanje:

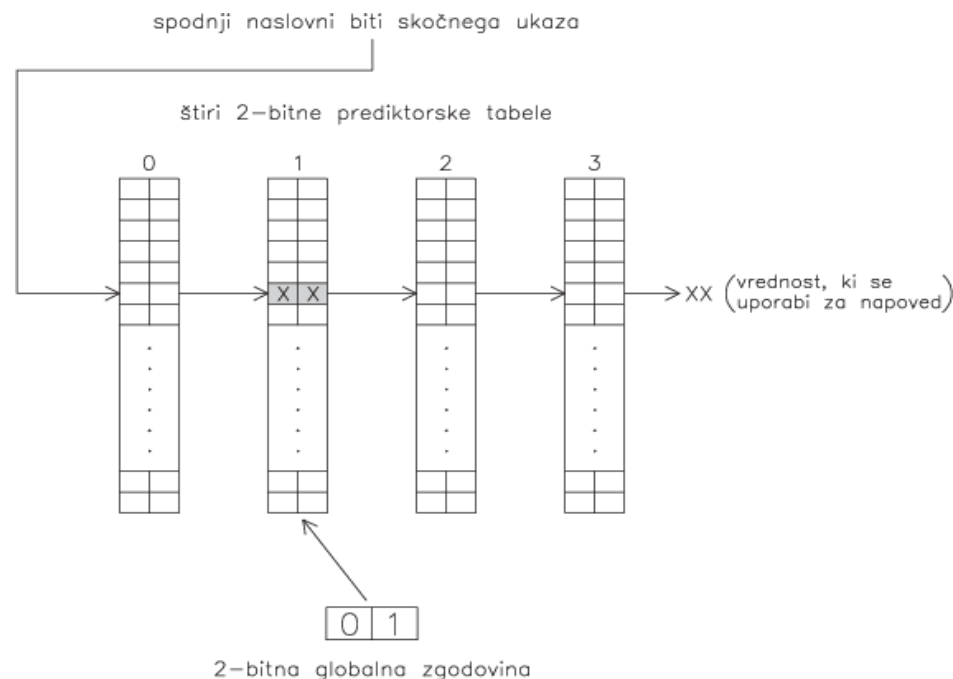
- prevzem ukaza v skladu s predikcijo
- v izbrani 2-bitni tabeli:
 - če pogoj izpolnjen, se vrednost poveča za 1
 - če pogoj ni izpolnjen, se vrednost zmanjša za 1

(maks.vrednost 3)

(min.vrednost 0)

Meritev na programih:

- (0,2) : 2-bitna tabela s 4096 naslovi: **7%** napačnih napovedi
- (2,2) : 4x 2-bitna tabela s 1024 naslovi: **4.3%** napačnih napovedi



4.2.3.1.2.4 Turnirski prediktor – „hibridni“ prediktorji

Turnirski prediktor uporablja:

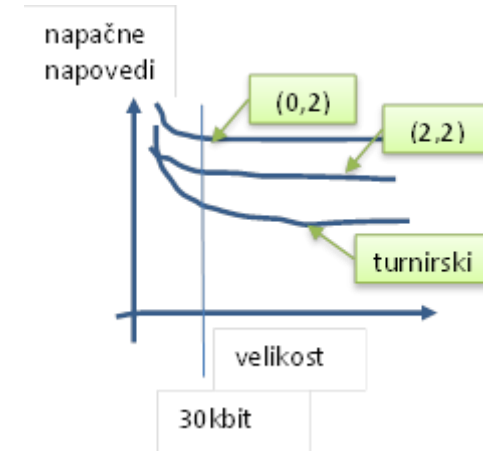
- več prediktorjev (tabel) in
- za vsak skočni ukaz ugotavlja, kateri prediktor (tabela) daje boljši rezultat.

Turnirski prediktor sestavljajo trije deli:

- globalni prediktor (podoben korelacijskemu prediktorju)
- lokalni prediktor
- selektor (izbere napoved globalnega ali lokalnega prediktorja)

Ko se ugotovi resnična izpolnjenost skočnega pogoja:

- se po potrebi osveži vsebina obeh prediktorjev
- se osveži vsebina selektorja samo, če sta različno napovedala.



Hibridni prediktorji – novejša nadgradnje ideje, z združevanjem napovedi različnih prediktorjev

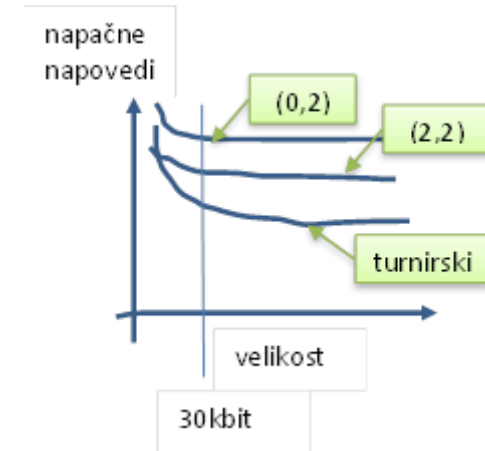
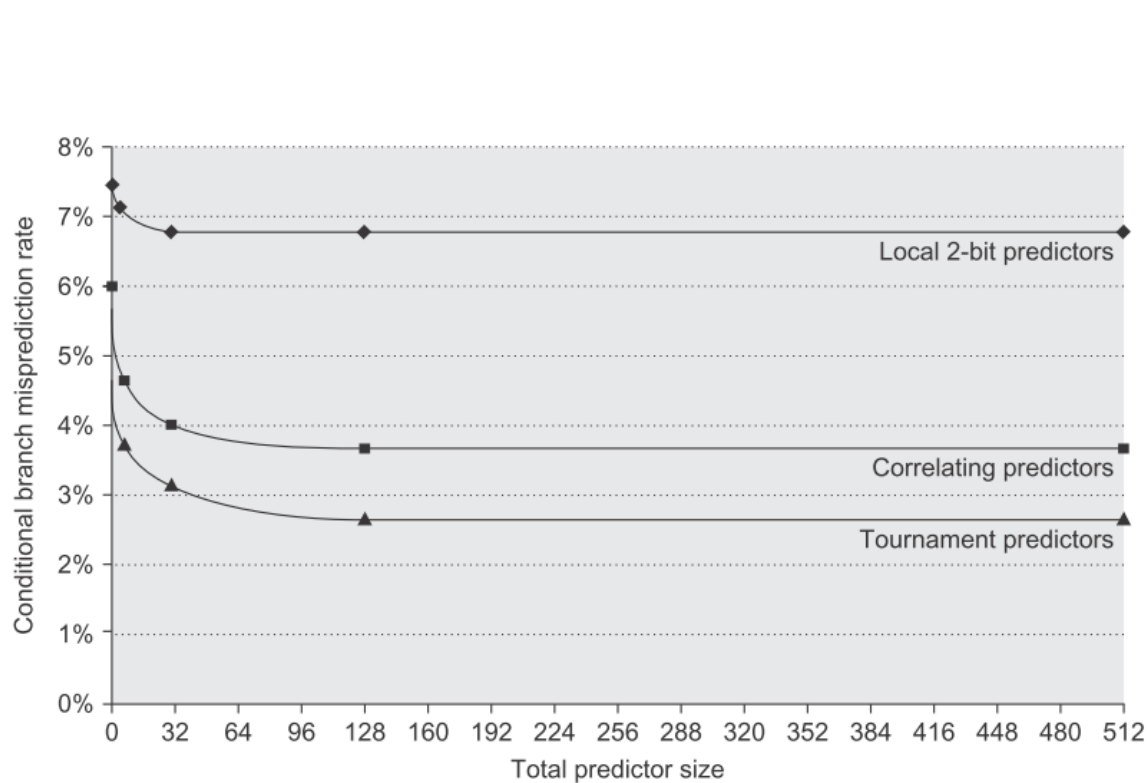


Figure 3.6 The misprediction rate for three different predictors on SPEC89 versus the size of the predictor in kilobits. The predictors are a local 2-bit predictor, a correlating predictor that is optimally structured in its use of global and local information at each point in the graph, and a tournament predictor. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks show similar behavior, perhaps converging to the asymptotic limit at slightly larger predictor sizes.

Nekateri novejši prediktorji

„Loop counter“:

- šteje in določa periodo ponovitev zanke
- običajno del hibridnega prediktorja

Vrnitveni prediktor:

- Povratni naslovi znani - > preprost sklad

Hibridni prediktor:

- Meta prediktor – selektor
 - Lahko tudi npr. saturacijski števec
- Lokalna prediktorja :
 - Pred. #1,
 - Pred. #2
- Primer:

4.2.3.1.2.5 Prediktorji – trenutno stanje

Aktualni:

- „gshare“ prediktor
 - kombinira „lokalno“ in „globalno“

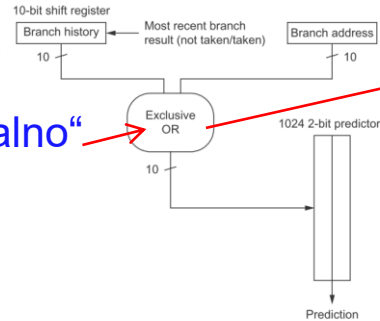


Figure 3.4 A gshare predictor with 1024 entries, each being a standard 2-bit predictor.

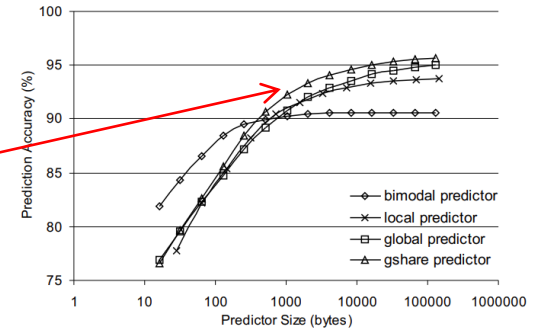


FIGURE 5. Branch prediction accuracy in function of predictor size for bimodal, global history, local history, and gshare prediction scheme.

- TAGE prediktor
 - več dolžin zgodovine

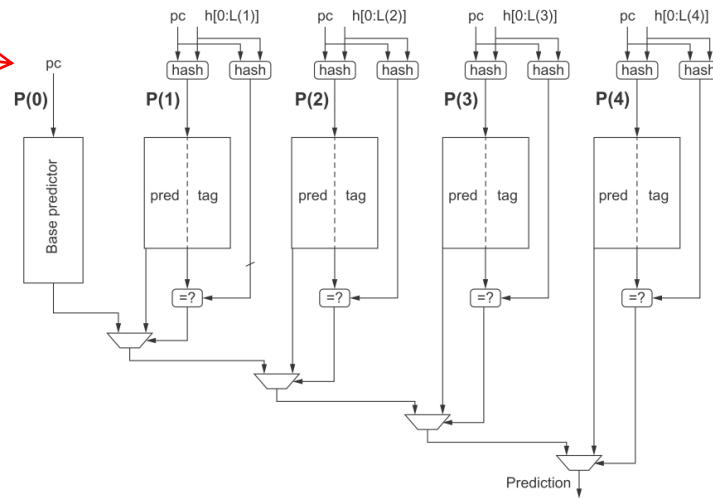
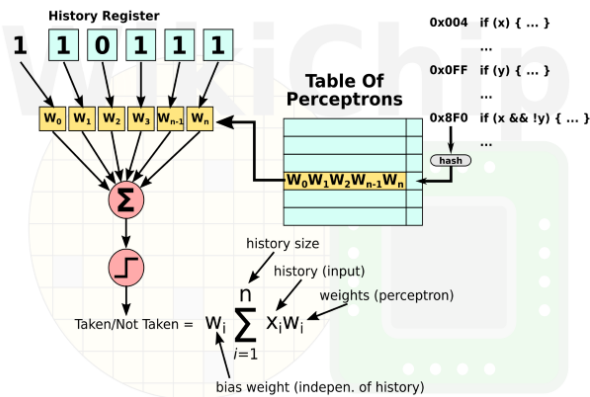


Figure 3.7 A five-component tagged hybrid predictor has five separate prediction tables, indexed by a hash of the branch address and a segment of recent branch history of length 0–4 labeled “h” in this figure. The hash can be as simple as an exclusive-OR, as in gshare. Each predictor is a 2-bit (or possibly 3-bit) predictor. The tags are typically 4–8 bits. The chosen prediction is the one with the longest history where the tags also match.

- Perceptron



4.2.3.1.2.5 Prediktorji – trenutno stanje

Izhodišča:

- Malo informacij o podrobnostih
- Do sedaj „gshare“ prediktor
- Trenutno najverjetneje TAGE prediktor ali hibridna kombinacija
- AMD: hibridni (Zen 2)
 - Perceptron (hiter) + TAGE (boljša napoved)
 - Perceptron hitro napove, TAGE kasneje preveri

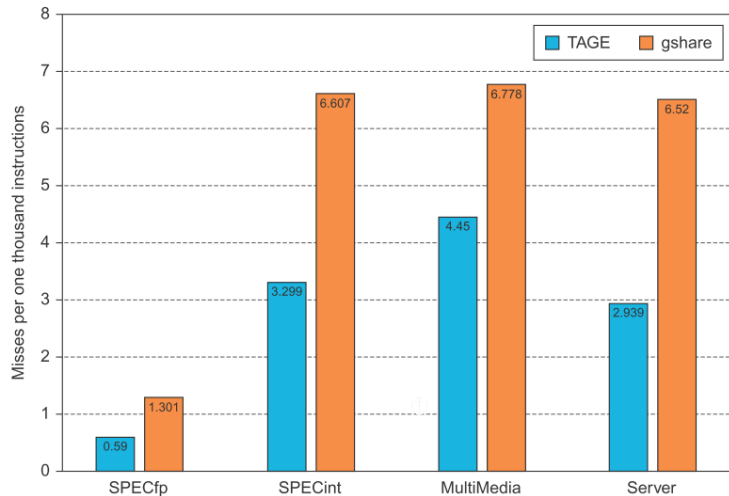
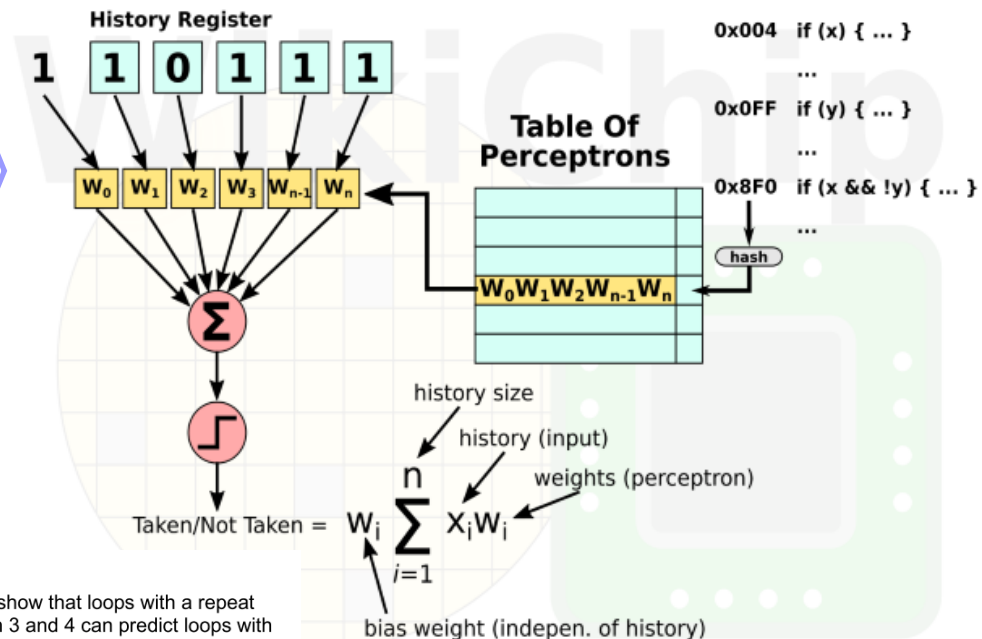


Figure 3.8 A comparison of the misprediction rate (measured as mispredicts per 1000 instructions executed) for **tagged hybrid versus gshare**. Both predictors use the same total number of bits, although tagged hybrid uses some of that storage for tags, while gshare contains no tags. The benchmarks consist of traces from SPECfp and SPECint, a series of multimedia and server benchmarks. The latter two behave more like SPECint.

4.2.3.1.2.5 Prediktorji – pogled naprej

Potenciali:

- izpopolnitev hibridnih prediktorjev ?
- izvedba obeh vej ?
- perceptroni – AMD ?
- „context switch“
 - tudi prediktorji ?



3.13 Branch prediction in AMD Zen

Branch prediction in Zen is based on perceptrons. My tests show that loops with a repeat count of up to 12 are predicted very well in Zen 1 and 2. Zen 3 and 4 can predict loops with a count of up to 64. Loops with higher repeat counts usually have one misprediction after the last iteration. Nested loops and branches inside loops are predicted well. Repetitive patterns are predicted very well, with no apparent limit to the length of the period. Multiway branches in the form of indirect jumps or calls are also predicted, though not as efficiently as conditional jumps.

3.

The microarchitecture of Intel, AMD, and VIA CPUs

An optimization guide for assembly programmers and compiler makers

By Agner Fog, Technical University of Denmark.
Copyright © 1996 - 2022. Last updated 2022-11-04.

<https://www.agner.org/optimize/microarchitecture.pdf>

4.2.3.1.3 Skočni predpomnilnik

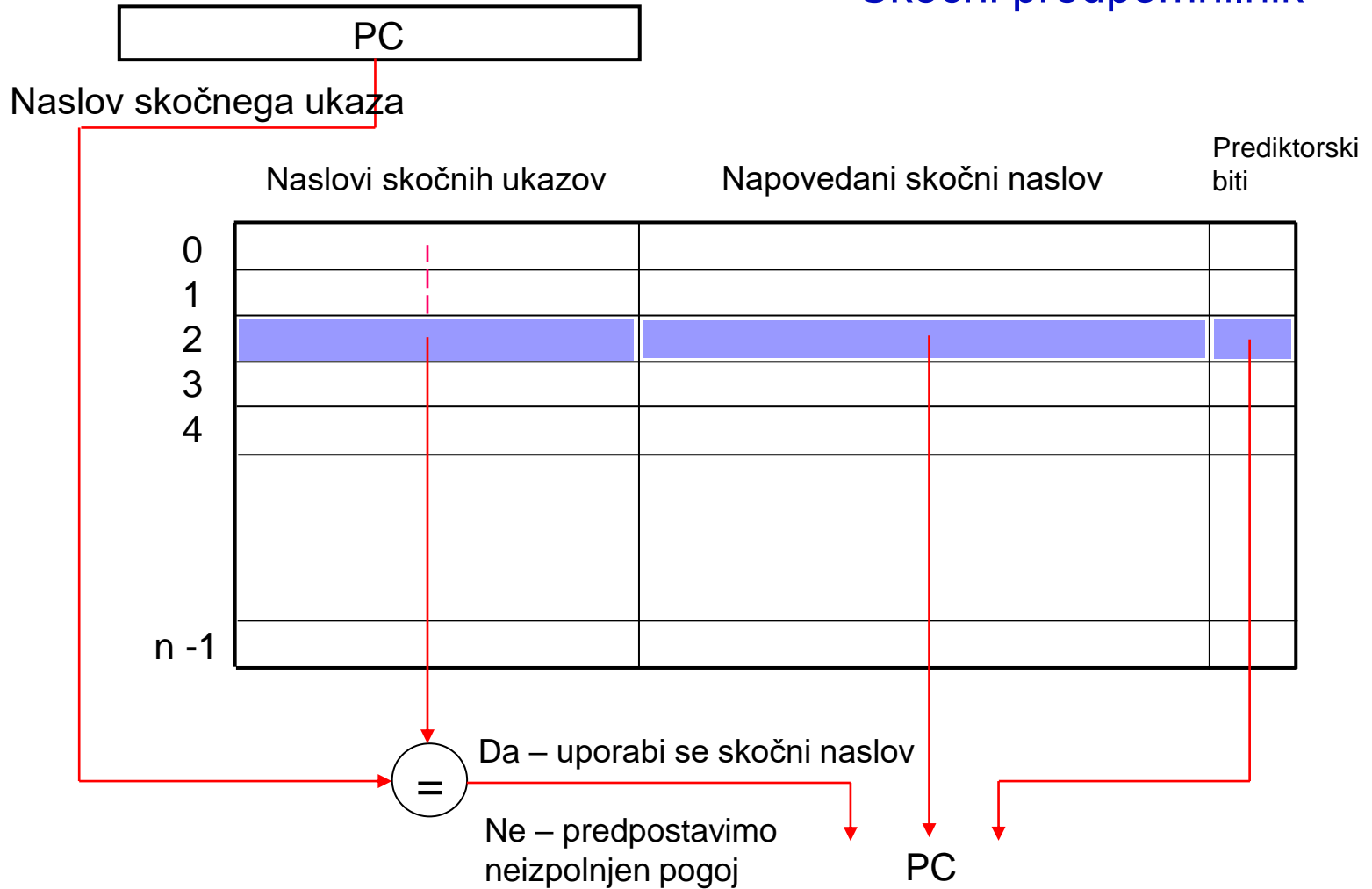
Za napoved skoka poleg napovedi pogoja rabimo še napoved skočnega naslova (sicer še vedno skočna zakasnitev):

- potrebujemo skočne naslove zadnjih nekaj skokov, kjer je bil pogoj za skok izpolnjen
- to je vsebina skočnega predpomnilnika.

V stopnji prevzema ukaza se hkrati zgodi:

- prevzem ukaza in
- dostop do skočnega predpomnilnika:
 - če zadetek, se prebere skočni naslov in prenese v PC.
 - če zgrešitev, predpostavimo neizpolnjen pogoj

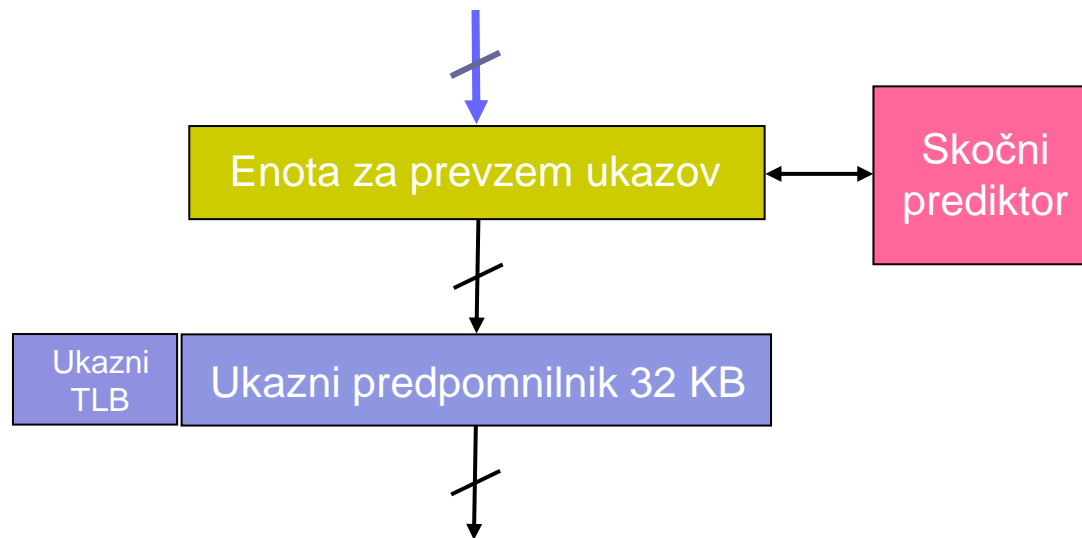
Skočni predpomnilnik



4.2.3.1.3 Skočni predpomnilnik

Če sta napoved izida skoka ali skočni naslov napačna, se mora ukaze, ki so bili v cevovodu, zavreči (zamenjati z mehurčki)

Pri današnjih superskalarnih računalnikih, ki lahko istočasno prevzamejo več ukazov, je namesto prve stopnje cevovoda (**IF**) enota za prevzem ukazov.



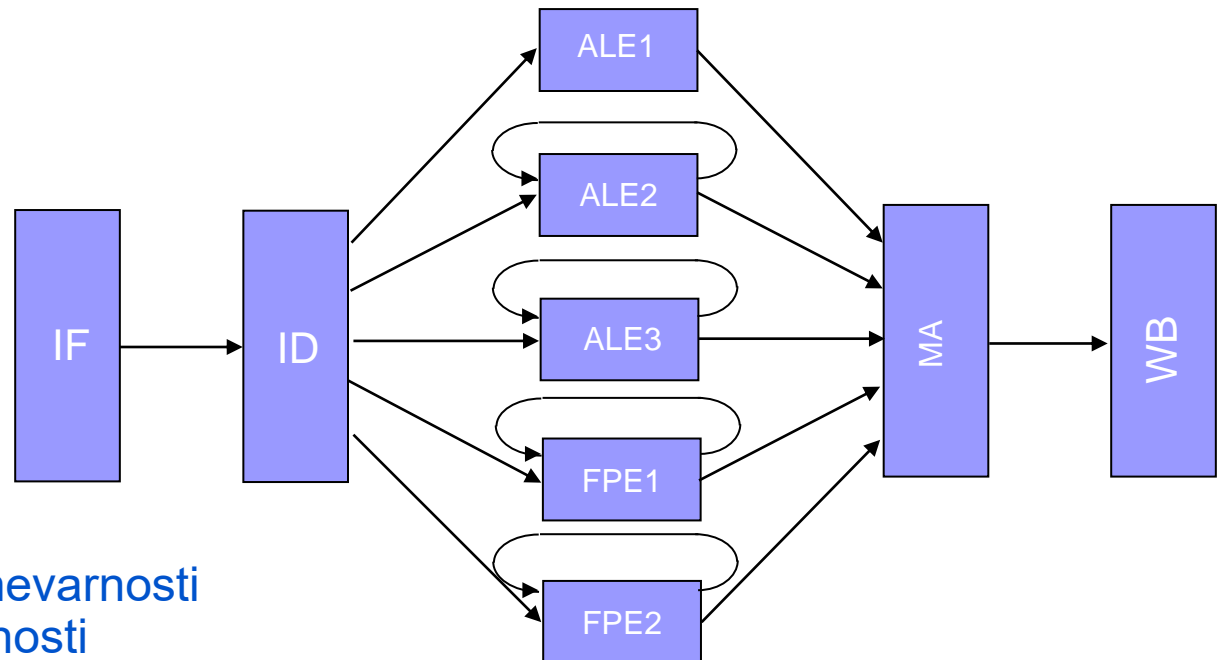
Prevzem ukazov pri procesorjih Intel Core

4.3 Izvajanje operacij, ki trajajo več urinih period

- Zakaj več urinih period?
 - Če bi pri kompleksnih operacijah kot so npr. celoštevilčno množenje in deljenje ter operacije v plavajoči vejici, zahtevali, da se izvršijo v eni urini periodi, bi
 - morala biti urina perioda zelo dolga (nizka frekvenca ure) ali
 - morali uporabiti zelo obsežno in kompleksno logično vezje.
- Noben od teh načinov ni dober, računalniki se gradijo tako, da se:
 - večina ukazov izvrši v eni urini periodi,
 - pri kompleksnih operacijah pa traja izvajanje ukaza več urinih period:
 - Če tak ukaz za pride v stopnjo EX, bi se moral cevovod ustaviti in čakati toliko urinih period, kolikor bi trajalo izvajanje v enoti EX
 - Zato: več stopenj EX ~ funkcijskih enot (FE)

Primer: petstopenjski cevovod s štirimi dodatnimi funkcijskimi enotami, ki izvajajo operacije, ki trajajo različno število urinih period.

- Izvrševanje v prvi funkcijski enoti (ALE1) traja eno urino periodo, v vseh ostalih pa več urinih period.



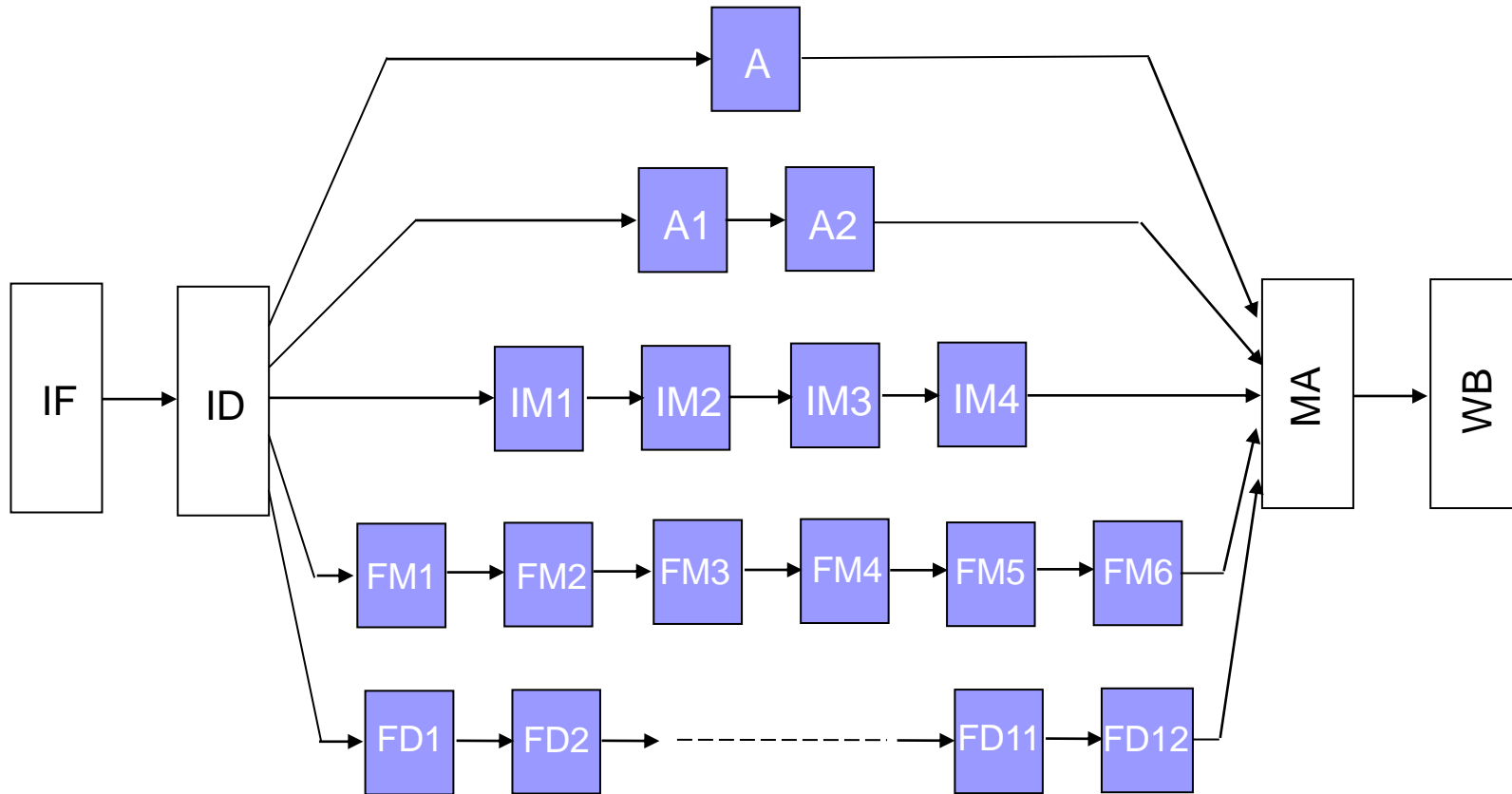
Večperiodne FE:

- povzročajo strukturne nevarnosti
- več podatkovnih nevarnosti
- več ukazov hkrati v MA, WB
 - (vrstni red!)

Rešitev: cevovodna realizacija (N ciklov = N stopenj)

+ odpravi strukturne nevarnosti

- ostanejo podatkovne nevarnosti (ni premoščanja med FE, več ukazov -> več nevarn.)
- več ukazov v MA, WB (lahko drugačni vrstni red – WAW, WAR)



Rešitev za boljšo izkoriščenost FE enot -> dinamično razvrščanje ukazov

Problemi cevovodne realizacije FE:

- več ukazov hkrati v stopnjah MA in WB:
 - v isti urini periodi lahko v stopnjo MA pride več ukazov
 - je treba več rezultatov shraniti v registre v stopnji WB.
 - v stopnjo WB rezultati prihajajo tudi v drugačnem vrstnem redu kot v programu.
- podatkovne nevarnosti :
 - v obdelavi je več ukazov hkrati -> več podatkovnih nevarnosti
 - premoščanje pri podatkovnih nevarnostih tipa RAW iz notranjosti cevovodne funkcijske enote ni možno.
- vsi ti problemi prinesejo dodatne čakalne periode, tako da so lahko nekatere funkcijske enote nekaj časa neizkoriščene:
 - Npr: zaporedje ukazov povzroči 10 čakalnih period (skupaj se izvaja 18 period):

Ukaz	Urina perioda																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
<i>FLD F4,0(R2)</i>	IF	ID	EX	MEM	WB												
<i>FMUL F0,F4,F6</i>		IF	ID	○	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
<i>FADD F2,F0,F8</i>			IF	○	ID	○	○	○	○	○	○	A1	A2	A3	A4	MEM	WB
<i>FST 0(R2),F2</i>					IF	○	○	○	○	○	○	ID	EX	○	○	○	MEM

- rešitev kako čim bolj zaposliti funkcijske enote je v dinamičnem razvrščanju ukazov.

4.4 Odpravljanje podatkovnih nevarnosti z dinamičnim razvrščanjem ukazov

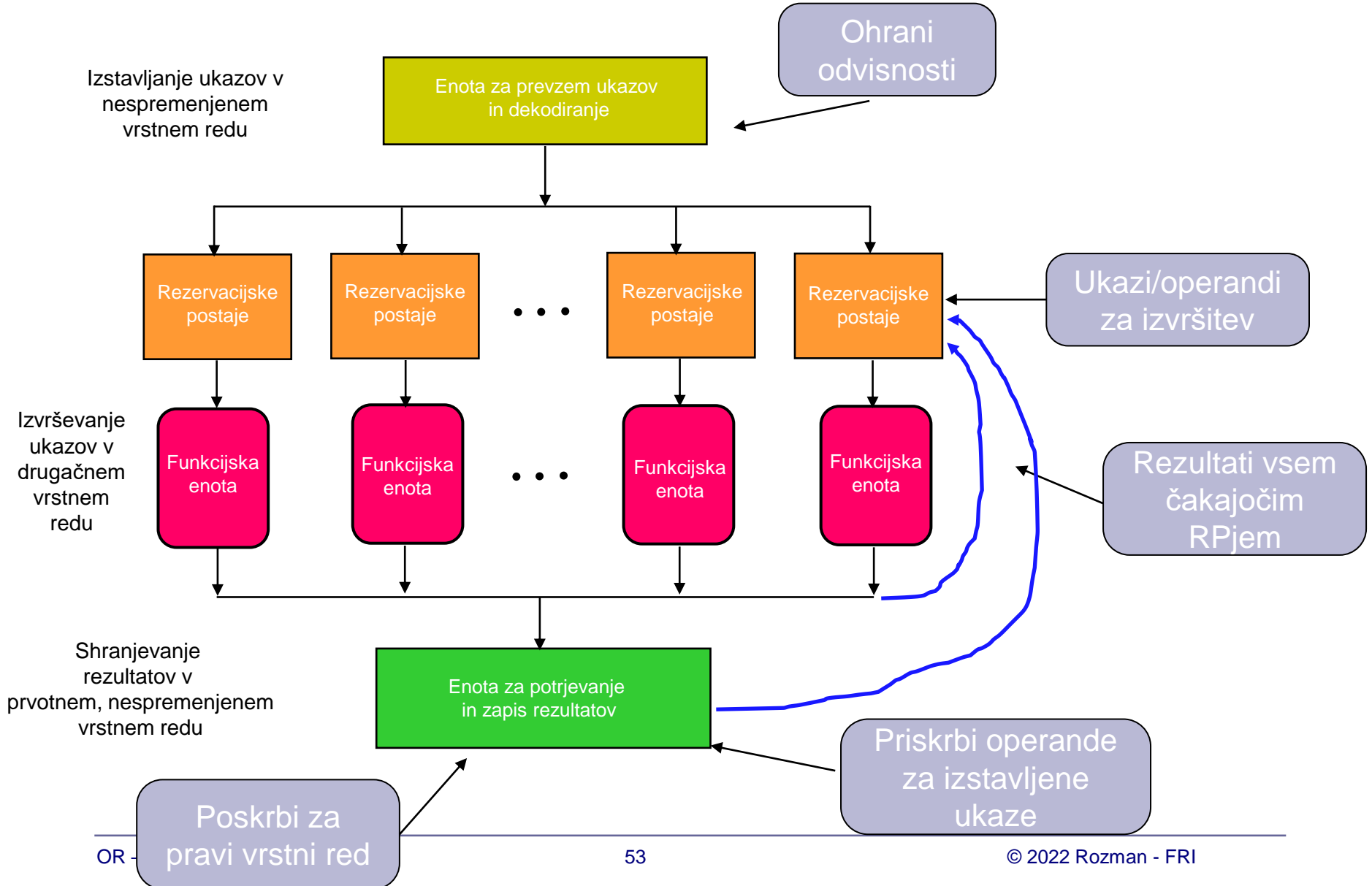
- Dinamično razvrščanje: način delovanja CPE, pri katerem se strojno spremeni vrstni red izvrševanja ukazov, da bi se zmanjšalo število čakalnih urinih period.
- Pri naslednjem zaporedju ukazov mora ukaz IDIV čakati, dokler se ne razreši podatkovna nevarnost prejšnjih dveh ukazov:

<i>IMUL</i>	<i>R2, R3, R4</i>	$R2 \leftarrow R3 * R4$	
<i>ADD</i>	<i>R6, R2, R5</i>	$R6 \leftarrow R2 + R5$	RAW
<i>IDIV</i>	<i>R7, R8, R5</i>	$R7 \leftarrow R8 / R5$	

Realizacija:

- ID - stopnjo za dekodiranje ukazov je potrebno razdeliti v dve stopnji:
 - izstavljanje : („in-order“)
 - ta stopnja dekodira ukaz
 - ugotavlja strukturne nevarnosti
 - branje operandov: („out-of-order“)
 - tu se preverjajo podatkovne nevarnosti
 - operandi se berejo ko ni podatkovne nevarnosti, sicer čakanje
- EX – izvrševanje ukazov („out-of-order“)
- (MEM, WB) shranitev rezultatov v registre ali pomn. („in-order“)

4.4 Dinamično razvrščanje ukazov – Tomasulov algoritem



4.4 Dinamično razvrščanje ukazov

Tomasulo algorithm simulator (protoype)

This simulates [Tomasulo's algorithm](#) for a floating-point MIPS-like instruction pipeline, demonstrating out-of-order execution. The source is on [GitHub](#).

Click instructions on the right to issue and execute them. Instructions will only execute if all of their data dependencies have been resolved, but they may issue in any order (though at least issuing them in order is recommended). Currently, loads have two-step execution and still require a writeback cycle. Regs[x] is the value at location x from the register file.

Color codes are as follows: *destination* *source* *occupied source* *occupied destination*.

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A	Result
Load0	true	L.D		Regs[R2]				
Load1	true	L.D		Regs[R3]				
Add0	true	ADD.D		Regs[F8]				
Add1	true	SUB.D			Load1	Load0		
Add2	true	ADD.D			Add1	Load1		
Mult0	true	MUL.D		Regs[F4]	Load1			
Mult1	true	DIV.D			Mult0	Load0		
Store0	true	S.D			Add2		Regs[R2] + 32	
Store1	false							

Instruction Status

Instruction	Issue	Execute	Write result
L.D F6,32(R2)	true		
L.D F2,44(R3)	true		
MUL.D F0,F2,F4	true		
ADD.D F10,F12,F8	true		
SUB.D F8,F2,F6	true		
DIV.D F10,F0,F6	true		
ADD.D F6,F8,F2	true		
S.D F6,32(R2)	true		

Spletna demonstracija:

■ <http://nathantypanski.github.io/tomasulo-simulator/>

Slabosti dinamičnega razvrščanja ukazov:

- se povečajo težave pri kontrolnih nevarnostih:
 - več ukazov v urini periodi :
 - v primeru napačne predikcije -> težko povemo, katere je potrebno razveljaviti
 - rešitev: čakanje do izida skoka in s tem naslednjega ukaza

Čimmanj teh čakanj do izida skokov ->
rešitev: špekulativno izvajanje ukazov....

4.5 Špekulativno izvrševanje ukazov

Špekulacija: način izvajanja ukazov, ki dovoljuje prevajalniku ali procesorju, da ugiba o lastnostih ukaza in s tem omogoči pogojno izvajanje ukazov, ki so odvisni od ukaza o katerem se špekulira.

Primeri:

- Špekuliranje o izidu skočnega ukaza, tako da se ukazi za skokom lahko prej izvedejo.
- Špekulacija je lahko tudi predvidevanje, da STORE in LOAD ukaz ne dostopata do iste lokacije in lahko zamenjamo vrstni red njunega izvajanja.

Težava pri špekulativnem izvrševanju ukazov:

- špekulacija (*špekulacija* ~ „manj zanesljiva napoved“) je pogosto napačna.

Potreben mehanizem, ki vključuje:

- način za preverjanje, ali je bila predpostavka pravilna.
- način za izničenje vsega, kar so naredili napačno napovedani in izvedeni ukazi.



- Špekulativno izvrševanje lahko izvaja prevajalnik (programsko - SW) in/ali procesor (strojno - HW).

Procesor s špekulativnim izvrševanjem ukazov običajno izvaja kombinacijo omenjenih pristopov:

- Dinamično predikcijo skokov, pri kateri se kljub skokom izberejo ukazi, ki gredo v izstavljanje.
- Dinamično razvrščanje ukazov.
- Špekulativno izvrševanje ukazov brez predhodnega čakanja na preverjanje pravilnosti skočne predikcije in z možnostjo izničenja vpliva napačno izvršenih ukazov pri napačni napovedi.

Da se prepreči vpliv izvršenih ukazov na stanje programsko dostopnih registrov dokler ni potrjena pravilnost napovedi skoka, se uporablja

preureditveni izravnalnik – PI („ReOrder Buffer“ - ROB)

F4 nastane v
ROB5

ukaz-vrsta

veljavnost

Preureditveni izravnalnik (ROB):

ponor

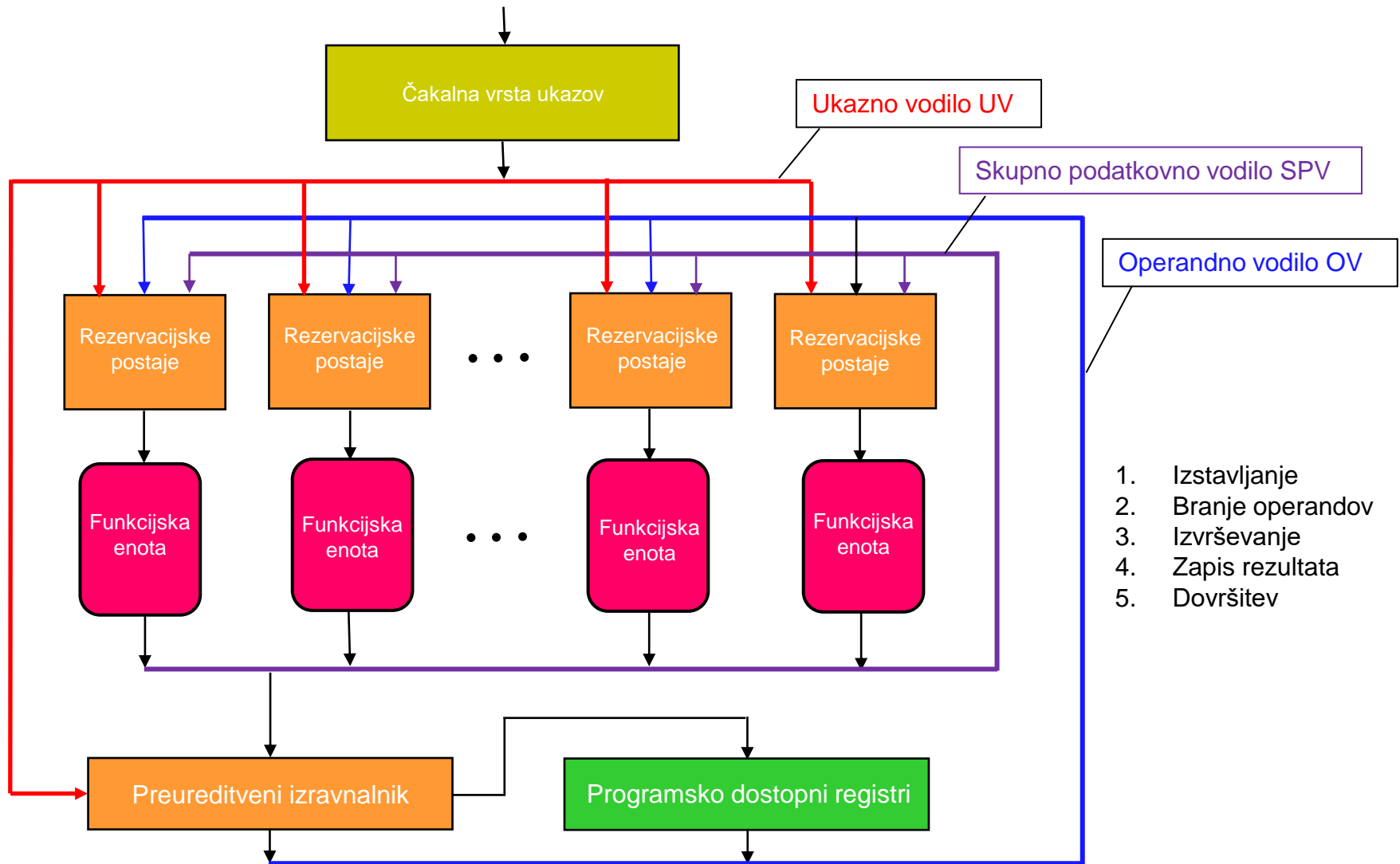
Done?

--	ROB5	ST 0 (R3) , F4	N	ROB7
F0		ADDD F0 , F4 , F6	N	ROB6
F4		LD F4 , 0 (R3)	N	ROB5
--		BNE F2 , <...>	N	ROB4
F2		DIVD F2 , F10 , F6	N	ROB3
F10		ADDD F10 , F4 , F0	N	ROB2
F0		LD F0 , 10 (R2)	N	ROB1

Newest
↓
Oldest

- FIFO vmesnik, ki vsebuje ukaze (operande in rezultate), dokler niso dovršeni (potrjena pravilnost špekulacije)
- loči „novo“ in „staro“ stanje
- koraki, ki so potrebni pri špekulativnem izvrševanju ukazov:
 - Izstavljanje, Branje operandov, Izvrševanje, Zapis rezultata, Dovršitev

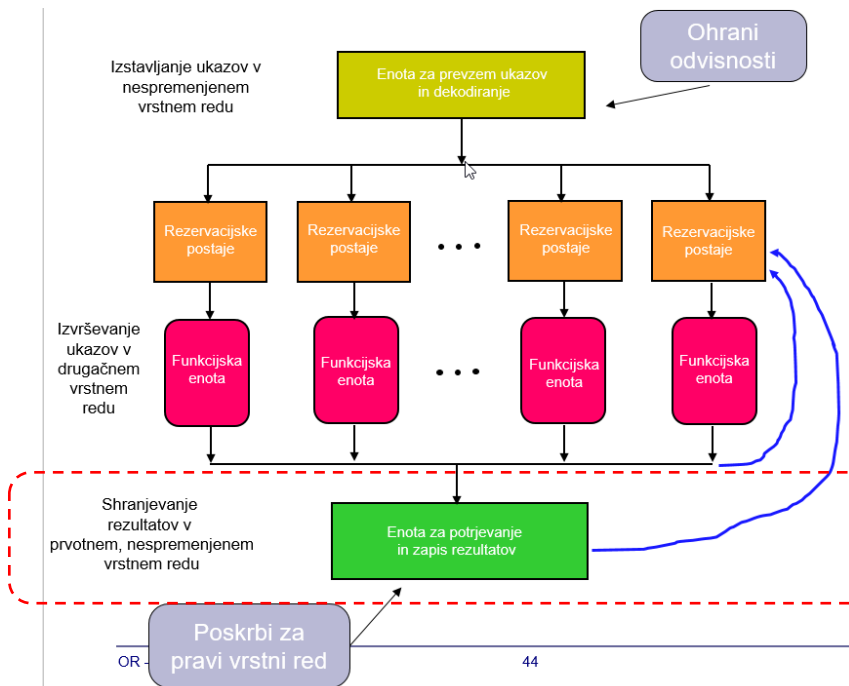
4.5 Špekulativno izvrševanje ukazov



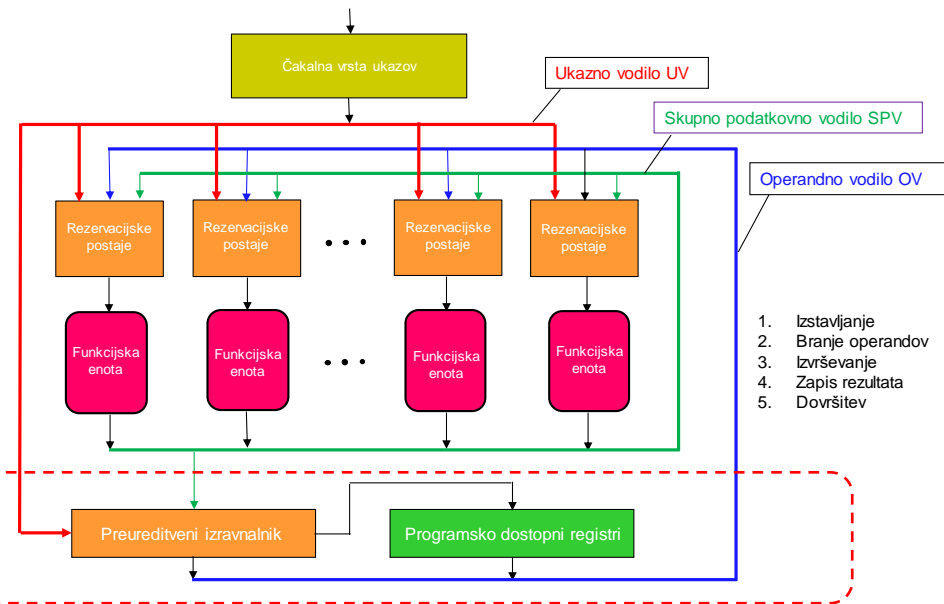
1. Izstavljanje
2. Branje operandov
3. Izvrševanje
4. Zapis rezultata
5. Dovršitev

4.5 Špekulativno izvrševanje ukazov – primerjava z dinamičnim razvrščanjem ukazov

4.4 Dinamično razvrščanje ukazov – Tomasulov algoritem



4.5 Špekulativno izvrševanje ukazov



Koraki pri špekulativnem izvajanju ukazov:

■ Izstavljanje (UV):

- Ukazi se iz čakalne vrste jemljejo v enakem vrstnem redu kot v programu.
 - Ukaz se prenese (če je prostor) v:
 - RP (rezervacijsko postajo) pri funkcijski enoti, ki jo ukaz potrebuje
 - in v PI (preureditveni izravnalnik).
 - Strukturna nevarnost:
 - če ni prostora v preureditvenem izravnalniku ali v RPju funkcijske enote

■ Branje operandov (OV):

- Operand, ki ga potrebuje ukaz, se lahko nahaja v:
 - preureditvenem izravnalniku
 - ali v programsko dostopnem registru.
- Operand lahko da še ni na voljo – označimo mesto, kjer se bo pojavil

■ **Izvrševanje (RP,FE,SPV):**

- Če v ukazu manjka eden ali oba operanda, ukaz v RP čaka, dokler se operand ne pojavi na skupnem podatkovnem vodilu (SPV).
- Ko ima ukaz vse operande:
 - gre v izvrševanje v funkcijsko enoto,
 - rezervacijska postaja pa se izprazni.

■ **Zapis rezultata (SPV):**

- Ko je na izhodu funkcijske enote rezultat ukaza, se prenese (SPV):
 - v vse rezervacijske postaje, ki čakajo nanj
 - in v preureditveni izravnalnik, v ukaz ki čaka na ta rezultat.

■ **Dovršitev (PI):**

- Ko se na izhodu preureditvenega izravnalnika pojavi veljaven ukaz, se dovrši.
 - skočni ukazi: preveri se pravilnost napovedi:
 - Če je napoved pravilna, je skočni ukaz izvršen.
 - Če pa je napoved napačna:
 - se izbriše celotna vsebina preureditvenega izravnalnika in vsebina čakalne vrste ukazov.
 - Prevzame se ukaz s pravilnega naslova.
 - ALE ali LOAD ukazi: se vrednost zapiše v programsko dostopni register.
 - STORE ukazi: se vrednost zapiše v pomnilnik.

- Če ukaz dovršen, se izbriše iz PI:
 - vsebina PI se pomakne navzdol
 - izhod:
 - se pojavi naslednji ukaz v vrsti
 - na vhodu:
 - se sprost prostor za novi ukaz

Velikost preureditvenega izravnalnika določa največje število ukazov, ki se lahko špekulativno izvršijo – **ukazno okno**.

Tip	ukazno okno	RP in FE	
AMD Opteron (»Barcelona«):	60	60RP za 11 FE	
Intel Core iX (»Nehalem«):	128	36RP za 12 FE	
Intel Core iX (»Sandy ridge«):	168	54 RP za 12 FE	
Intel Core iX (»Haswell«):	192	64 RP za 20 FE	
Intel Core iX (»Skylake«):	224	97 RP za 22 FE	
Intel Core iX (»Sunny Cove«): I. 2019	352	>125 (???)	„...352 entries might be equivalent to as much as 525 μ OPs...“
Intel Core iX (»Golden Cove«): I. 2021	512	?	

Spletna demonstracija delovanja PI :

http://www.ecs.umass.edu/ece/koren/architecture/ROB/rob_simulator.htm

Instructions

Opcode	Operands		
	Dest.	Op1	Op2
LD ▾	F6 ▾	F0 ▾	F0 ▾
LD ▾	F2 ▾	F0 ▾	F0 ▾
MULD ▾	F0 ▾	F2 ▾	F4 ▾
ADD.D ▾	F0 ▾	F2 ▾	F4 ▾
S.D ▾	F0 ▾	F0 ▾	F6 ▾

Reorder Buffer

	Entry Number	Instruction	Destination	Value	Valid	Exception
	1					
HEAD	2	L.D F2 x Ry	F2	[Mem[...]]	1	0
	3	MULD F0 F2 F4	F0		0	0
	4	ADD.D F0 F2 F4	F0		0	0
	5	S.D F0 x Ry	Mem[...]	(from #4)	0	0
TAIL	6					

Reservation Stations

Name	Busy	Opcode	Vj	Vk	Qj	Qk	Destination
Add1-1	yes	ADD.D	[Mem[...]]	Regs[F4]			#4
Add1-2	no						
Add1-3	no						
Mul1-1	yes	MULD	[Mem[...]]	Regs[F4]			#3
Div1-1	no						

4.6 Večizstavitveni procesorji

Z znanimi metodami:

- dinamično predikcijo skokov,
- špekulativnim izvrševanjem in
- dinamičnim razvrščanjem ukazov

se CPI (povprečno število urinih period za izvedbo ukaza) približa 1.

CPI se lahko zmanjša pod 1, če se v vsaki urini periodi prevzame in izstavi v izvrševanje več ukazov.

Pri takih procesorjih se običajno uporablja enota IPC (instructions per clock) to je povprečno število izvršenih ukazov v urini periodi.

$$IPC = \frac{1}{CPI}$$

Delovanje n -kratnega večizstavitvenega procesorja:

- Pri prevzemu n -ukazov mora ukazni predpomnilnik dostavljati n ukazov v urini periodi v čakalno vrsto.
- Iz čakalne vrste se prevzame n ukazov v enakem vrstnem redu kot so v programu.
- Če predpostavimo, da med prevzetimi ukazi ni skokov, je potrebno preveriti odvisnost med operandi.

Strojno ugotavljanje medsebojnih podatkovnih odvisnosti je zahtevno za realizacijo:

- Poleg HW rešitve se pojavi še programska

Večizstavitvene procesorje tako delimo v dve vrsti:

- **VLIW procesorji** – število ukazov, ki so prevzeti in izstavljeni v eni urini periodi je določeno s programom (prevajalnik) vnaprej in se med delovanjem ne spreminja.
- **Superskalarni procesorji** – število prevzetih in izstavljenih ukazov se med izvajanjem programa dinamično spreminja in ga določa logika v procesorju.

4.6.1 Superskalarni procesorji

Izraz superskalarni procesor:

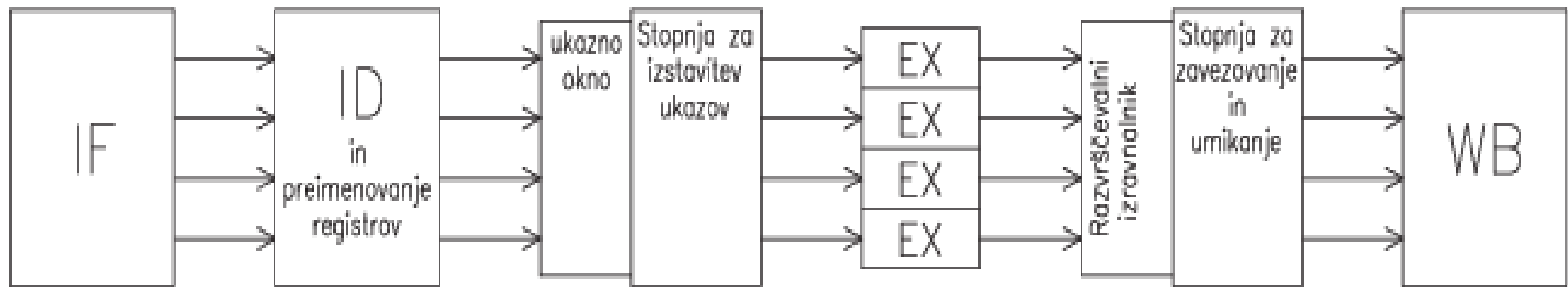
- prvič uporabljen pri IBM projektu America, ki je bil osnova za serijo procesorjev IBM RS/6000 (Power1),
- nekateri viri: „da je bil že superračunalnik CDC 6600 leta 1963 superskalarni“.

Superskalarni procesor dinamično določa, kateri ukazi se v eni urini periodi izstavijo v izvrševanje.

n -kratni superskalarni procesor:

- Za ugotavljanje medsebojnih podatkovnih odvisnosti je pri n tri-operandnih registrskih ukazih potrebnih $n^2 - n$ primerjav.
 - Pri predpostavki, da so vsi operandi registrski in da ima vsak ukaz dva vhodna in en izhodni operand, je pri n ukazih potrebnih $n^2 - n$ primerjav.
- Pri $n = 6$ je to 30 primerjav, ki jih je težko narediti v eni urini periodi. Zato se pri superskalarnih računalnikih to primerjanje opravi v več stopnjah cevovoda.
- Ko so morebitne medsebojne odvisnosti operandov ugotovljene, se nadaljevanje izstavljanja ukazov ne razlikuje bistveno od špekulativnega izvrševanja ukazov.
- Razlika je, da se v eni urini periodi namesto enega, v rezervacijske postaje izstavi do največ n ukazov.

4.6.1 Superskalarni procesorji



Poenostavljena shema superskalarnega procesorja

- Če več ukazov potrebuje isto funkcijsko enoto ali pa je preureditveni izravnalnik poln, pride seveda do strukturne nevarnosti, kar povzroči čakanje.
- Zato je pri superskalarnih procesorjih:
 - število funkcijskih enot običajno večje od največjega števila ukazov, ki so izstavljeni v eni urini periodi,
 - večji pa je tudi preureditveni izravnalnik.

Lastnosti nekaterih superskalarnih procesorjev

Procesor	Prevzeti-izstavljeni-dovršeni ukazi	Število funkcijskih enot
IBM RS/6000 (1990)	2 – 2 – 2	2
Digital Alpha 21264 (1998)	4 – 4 – 11	6
Intel Pentium 4 (2000)	3 – 3 – 4 **	7
IBM Power5 * (2003)	8 – 4 – 8	8
AMD Opteron X4 * (2007)	9 – 6 – 4 **	11
Intel Core iX * (2008)	6 – 6 – 4 **	12
Intel Core iX * Haswell (2013)	8 – **	20
Intel Core iX * SunnyCove (2019)	12 – 10 – 7 ** (unfused)	6

* Pri večjedrnih procesorjih se podatki nanašajo na eno jedro

** Velja za μ -operacije

■ Preureditveni izravnalnik je pri superskalarnih procesorjih zelo zapleten.

- Rezultate, ki pridejo iz funkcijskih enot je potrebno v isti urini periodi zapisati v vse ukaze, ki v izravnalniku čakajo nanje.
- V koraku branje operandov je potrebno operande, ki so že v izravnalniku, prenesti v rezervacijske postaje.
- Ob izvršitvi ukazov je treba operande zapisati v registre.

Primer: Pentium III: :

- PI porabi 27% celotne energije !!!
- „Preveč zapleteno, preveč porabe!“

Novejši procesorji (po letu 2000) zato običajno uporabljajo **eksplicitno preimenovanje registrov**:

■ razširjena množica registrov:

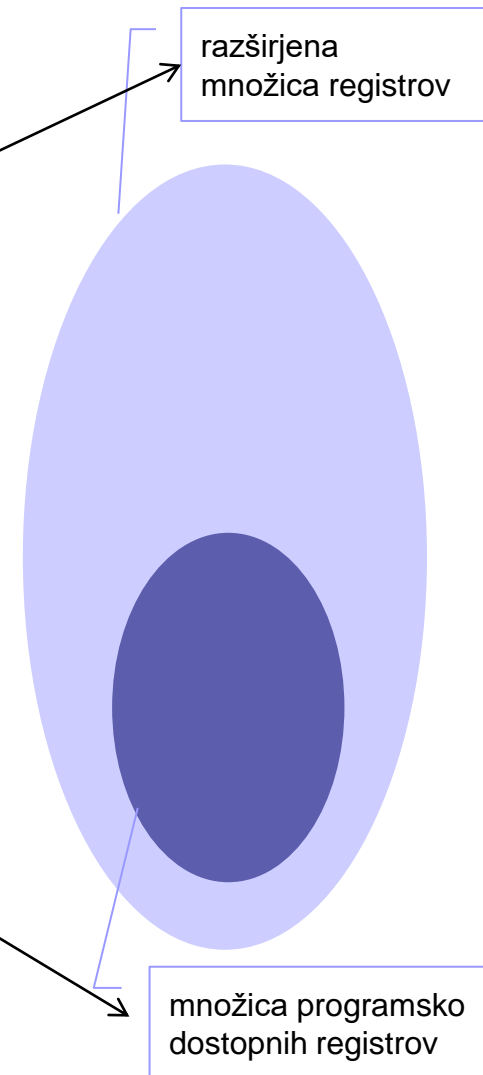
- procesor ima poleg programsko dostopnih registrov še precej več dodatnih (začasnih) registrov
- Oboji skupaj tvorijo razširjeno množico registrov.

■ preimenovalna tabela:

- določa ali je nek register v določeni urini periodi:
 - programsko dostopen,
 - prost ali
 - zaseden

■ preureditveni izravnalnik se poenostavi:

- zagotavlja le dovršitev ukazov v enakem vrstnem redu kot so v programu in
- v njem ni več začasnih rezultatov, so v začasnih registrih

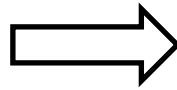


Preimenovanje registrov – osnovni pristop

Želimo izvesti 2 operaciji :

- $M[1024] + 2 \rightarrow M[1032]$
- $M[2048] + 4 \rightarrow M[2056]$

#	Instruction
1	$R1 = M[1024]$
2	$R1 = R1 + 2$
3	$M[1032] = R1$
4	$R1 = M[2048]$
5	$R1 = R1 + 4$
6	$M[2056] = R1$



#	Instruction	#	Instruction
1	$R1 = M[1024]$	4	$R2 = M[2048]$
2	$R1 = R1 + 2$	5	$R2 = R2 + 4$
3	$M[1032] = R1$	6	$M[2056] = R2$

Razreši WAW/WAR
RAW ostanejo

Preimenovalna tabela

Prvotni ukazi

```
add r1, r2, r3
sub r3, r2, r1
mul r1, r2, r3
div r2, r1, r3
```

r1	r2	r3
11	12	13
14	12	13
14	12	15
16	12	15
16	17	15

Prosti reg.

```
14, 15, 16, 17
15, 16, 17
16, 17
17
```

Preimenovanje reg.

```
add 14, 12, 13
sub 15, 12, 14
mul 16, 12, 15
div 17, 16, 15
```

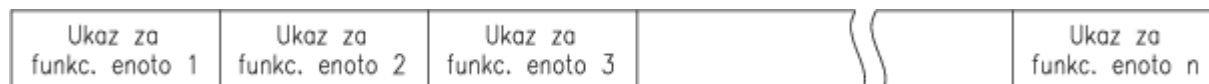
Preimenovalna tabela tako v vsaki urini periodi vsebuje informacije v katere registre razširjene množice so preslikani programsko dostopni registri.

4.6.2 VLIW procesorji - „Very Long Instruction Word“

Ideja v 80. letih, najprej na bolj specializiranih (DSP: TI C6000 še danes)

Značilnosti:

■ t.i. **dolgi ukazi:**



- so sestavljeni iz več enostavnejših ukazov, ki se izvedejo paralelno
- vsak enostavnejši ukaz zaposli eno FE, tipično:
 - 3 celoštevilčni ukazi, 2 FP ukaza, 2 dostopa do pomn. in 1 skok.
- **prevajalnik** oz. program določi in razvrsti ukaze v dolge ukaze tako, da so FE čimbolje izkoriščene (če ne najde -> **NOP**)
- **Potencialne prednosti:**
 - logika v procesorju enostavna :
 - ni preverjanja odvisnosti
 - ni detekcije nevarnosti
 - prevajalnik vidi širšo vsebino (kot ukazno okno pri superskalarnih)
 - odvisnost se preveri le enkrat (ob prevajanju programa)...

4.6.2 VLIW procesorji - „Very Long Instruction Word“

Slabosti VLIW iz prakse:

- večja dolžina programov
 - Prevajalnik vstavlja tudi NOP !
- togost
 - Enote delujejo ali stojijo hkrati
- „statična“ odločitev
 - Prevajalnik naredi pred izvajanjem programa

I. 1997: HP, Intel predlagata EPIC:

- „Explicitly Parallel Instruction Computing“
- Intel: Itanium 1, 2

Praksa ponovno pokaže:

- VLIW niso nič manj zapleteni od superskalarnih
- Fcpe se ni dvignila v skladu s pričakovanji

„...nobeden (VLIW; superskalarni) ni konsistentno boljši od drugega.“

4.7 Primeri izvedbe cevovodov

Tipični primeri izvedbe cevovodov:

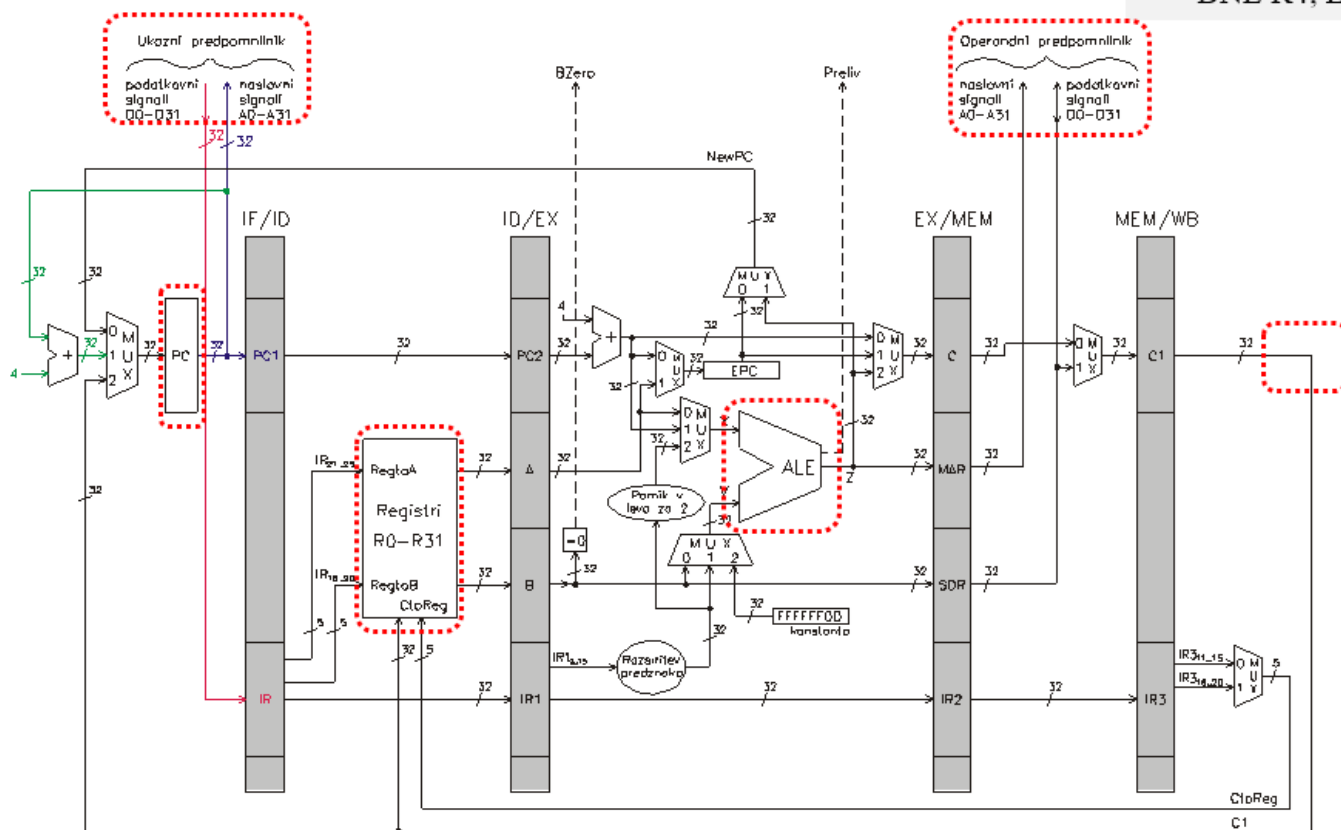
- HiP 5-stopenjski cevovod :
 - brez premoščanja $19 t_{cpe}$ period
 - s premoščanjem $11 t_{cpe}$ period
- ARM9TDMI 5-stopenjski cevovod (FRI SMS)
- ARM Cortex A8 cevovod
- Core I7 cevovod

4.7.1 HiP 5-stopenjski cevovod

HiP_cevovod_OR_v3.pdf (brez premoščanja)

HiP – shema cevovoda brez premostitev

```
Program:  
LOOP: LW R1,0(R2)  
      ADDI R1,R1,#1  
      SW 0(R2),R1  
      SUB R4,R3,R2  
      BNE R4, LOOP
```

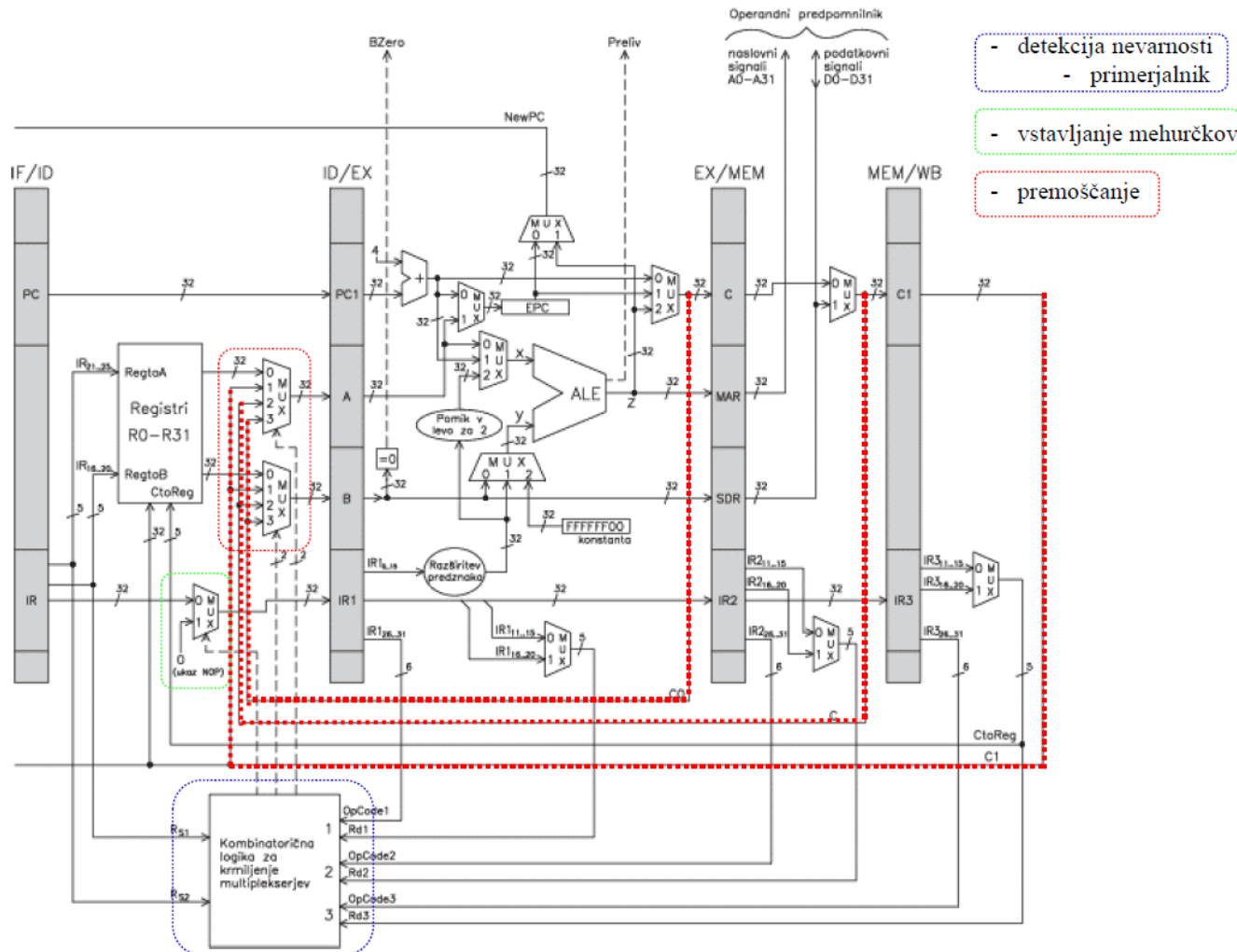


4.7.1 HiP 5-stopenjski cevovod

HiP_cevovod_OR_v3_premoscanja.pdf

(premoščanja)

HiP – shema z logiko za podatkovne nevarnosti in premoščanje



4.7.2 ARM9TDMI 5-stopenjski cevovod (FRI SMS)

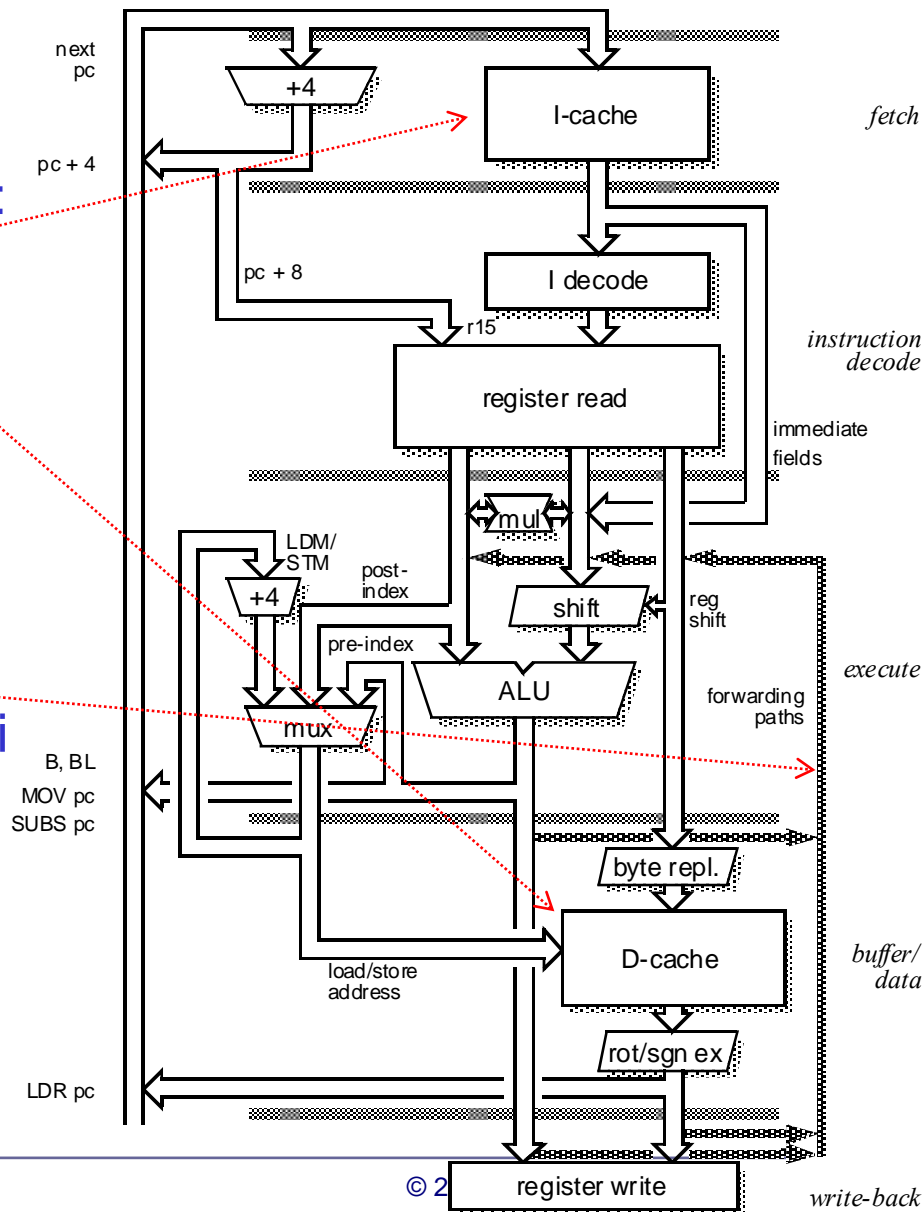
■ Harvardska arhitektura

- poveča VN ozko grlo z ločitvijo:
 - ukazni predpomnilnik
 - operandni predpomnilnik
- hkratni dostop do obeh

■ 5-stopenjski cevovod

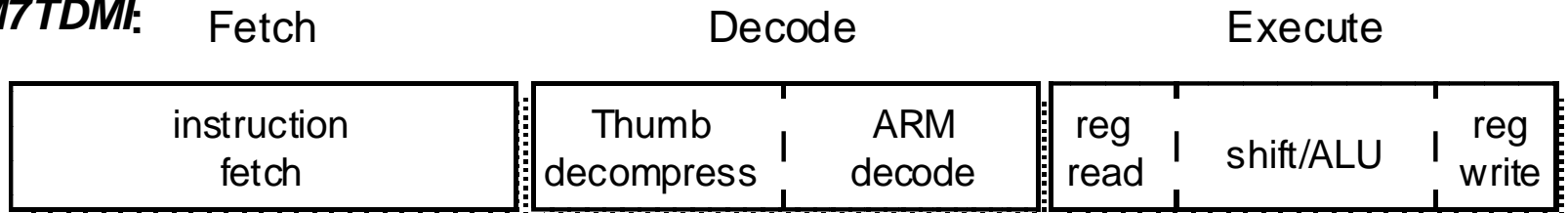
- premoščanje
- statična predikcija „neizpolnjeni pogoj“ :
 - BNE traja $1t_{cpe}$ (pogoj ni izpolnjen)
 - BNE traja $3t_{cpe}$ (pogoj je izpolnjen)

■ Realni (merjeni) CPI ~1.5

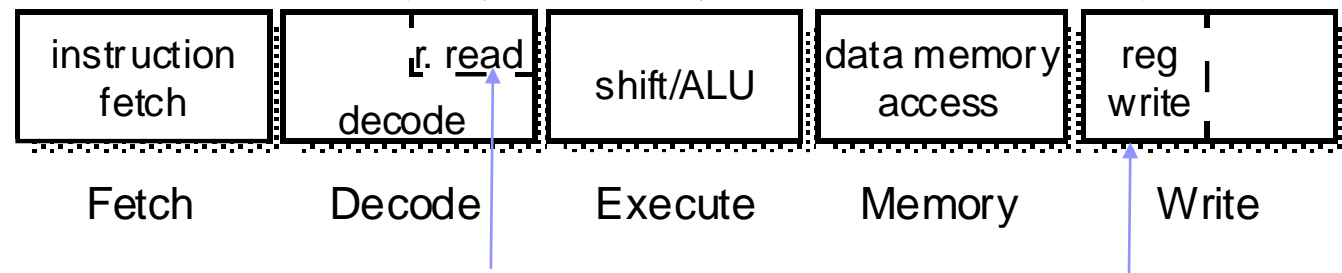


4.7.2 ARM9TDMI 5-stopenjski cevovod (FRI SMS)

ARM7TDMI:



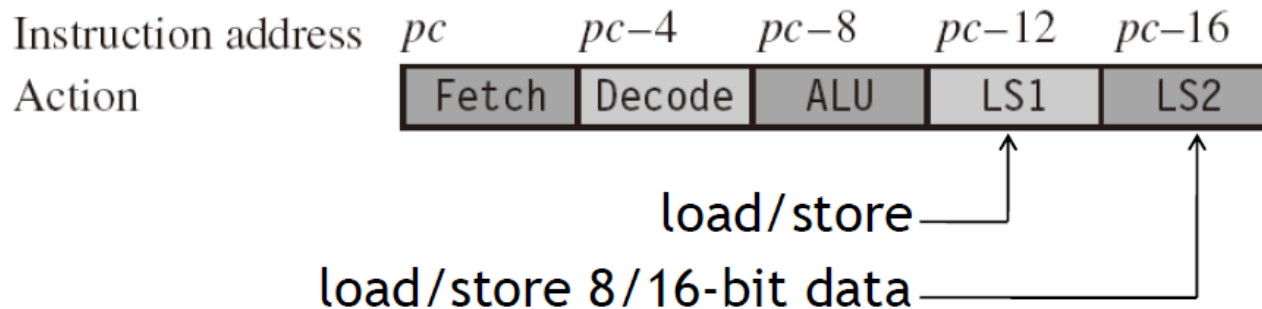
ARM9TDMI:



■ ARM9TDMI podatki:

Process	0.25 um	Transistors	110,000	MIPS	220
Metal layers	3	Core area	2.1 mm ²	Power	150 mW
Vdd	2.5 V	Clock	0 to 200 MHz	MIPS/W	1500

4.7.2 ARM9TDMI 5-stopenjski cevovod (FRI SMS)



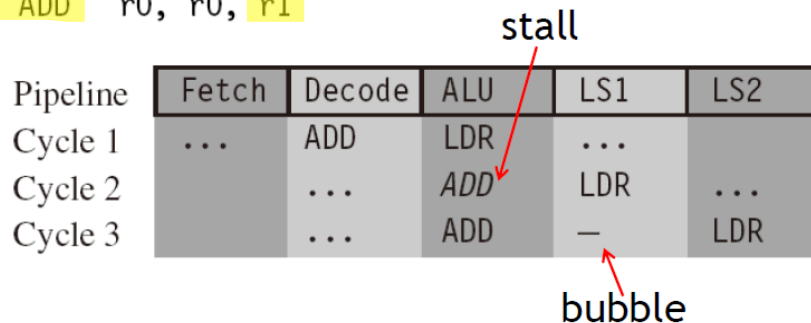
- No hazard, 2 cycles

```
ADD r0, r0, r1
ADD r0, r0, r2
```

- Premoščanje reši podatkovno nevarnost

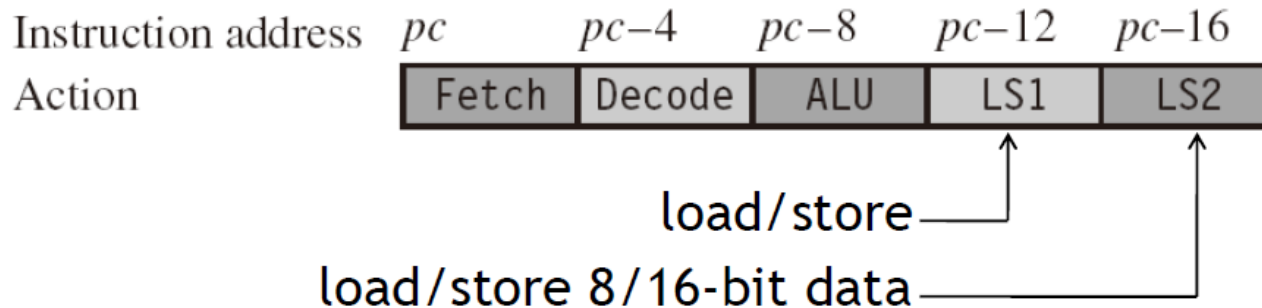
- One-cycle interlock

```
LDR r1, [r2, #4]
ADD r0, r0, r1
```



- LDR : dobi rezultat šele v LS1 !

4.7.2 ARM9TDMI 5-stopenjski cevovod (FRI SMS)



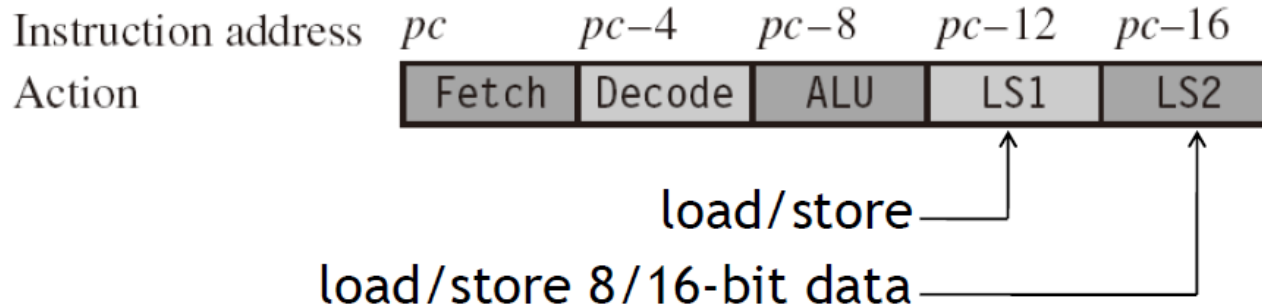
- One-cycle interlock, 4 cycles

LDRB $r1$, $[r2, \#1]$
 ADD $r0, r0, r2$; no effect on performance
 EOR $r0, r0, r1$

Pipeline	Fetch	Decode	ALU	LS1	LS2
Cycle 1	EOR	ADD	LDRB
Cycle 2	...	EOR	ADD	LDRB	...
Cycle 3	EOR	ADD	LDRB
Cycle 4	EOR	—	ADD

- LDRB : dobi rezultat šele v LS2 !

4.7.2 ARM9TDMI 5-stopenjski cevovod (FRI SMS)



- Branch takes 3 cycles due to stalls

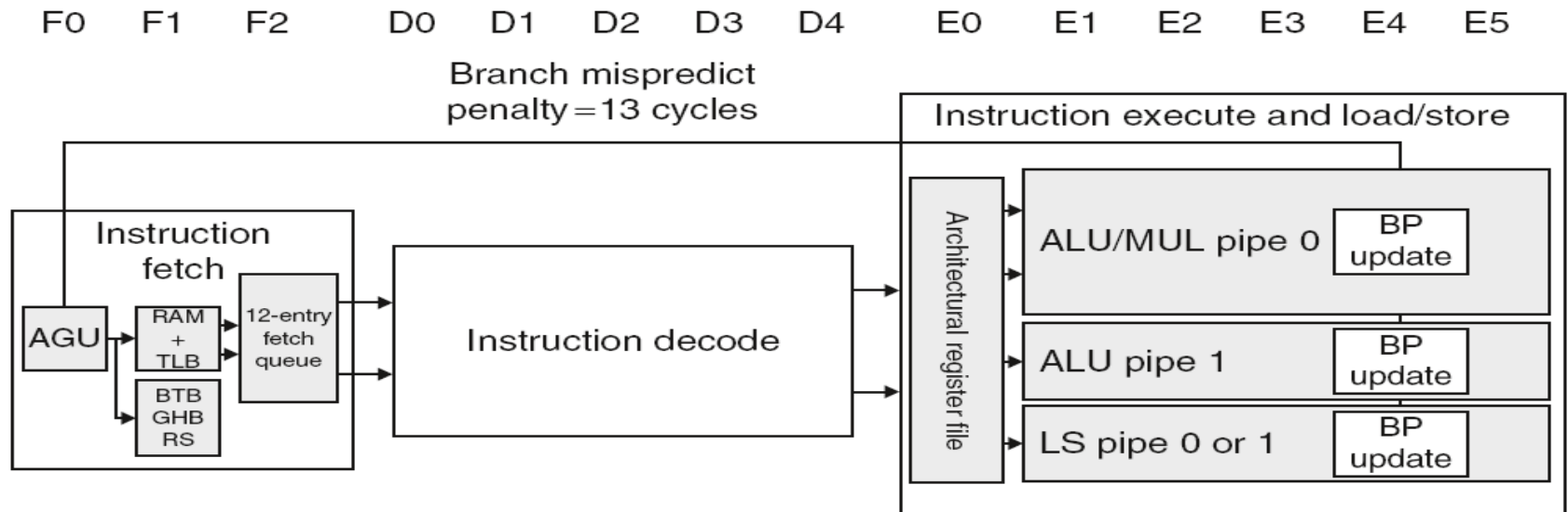
```

MOV  r1, #1
B    case1
AND  r0, r0, r1
EOR  r2, r2, r3
...
case1
SUB  r0, r0, r1
    
```

Pipeline	Fetch	Decode	ALU	LS1	LS2
Cycle 1	AND	B	MOV	...	
Cycle 2	EOR	AND	B	MOV	...
Cycle 3	SUB	—	—	B	MOV
Cycle 4	...	SUB	—	—	B
Cycle 5		...	SUB	—	—

Za B ukazom je potrebno sprazniti cevovod (vstaviti mehurčke) !

4.7.3 ARM Cortex-A8 Cevovod (poenostavljen)

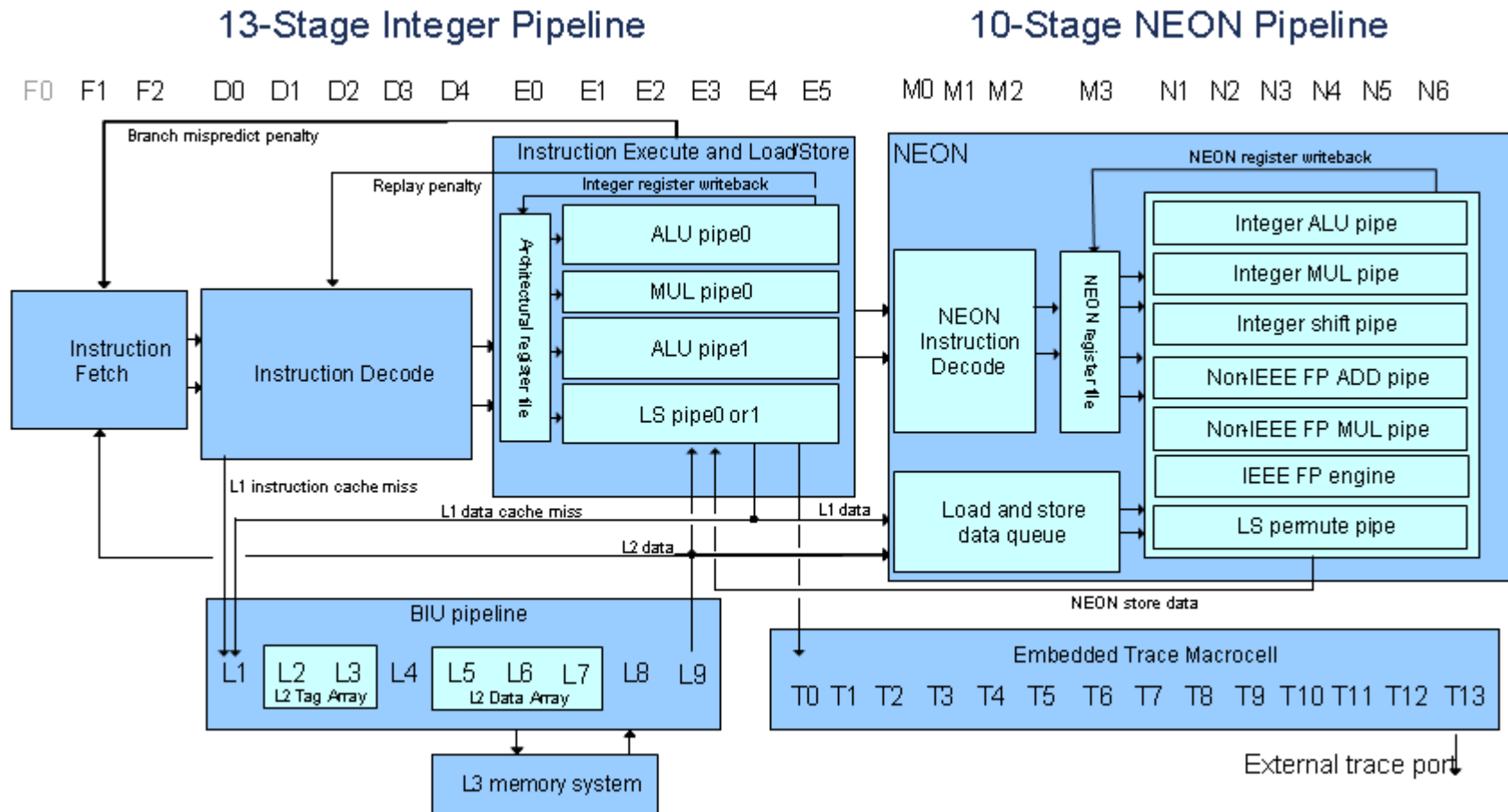


- 14 stopenjski (3 delni) superskalarni cevovod
 - dinamični več-izstavitveni
 - statični „in-order“ cevovod
 - pomembna vloga prevajalnika !
- odprava kontrolnih nevarnosti:
 - 2-stopenjska predikcija
- odprava podatkovnih nevarnosti:
 - polno premoščanje med EX cevovodi.

- 3 sekcije:
 - Fetch
 - Decode
 - Execute

- Execute - 3 poti :
 - 1x LOAD/STORE
 - 2x ALE

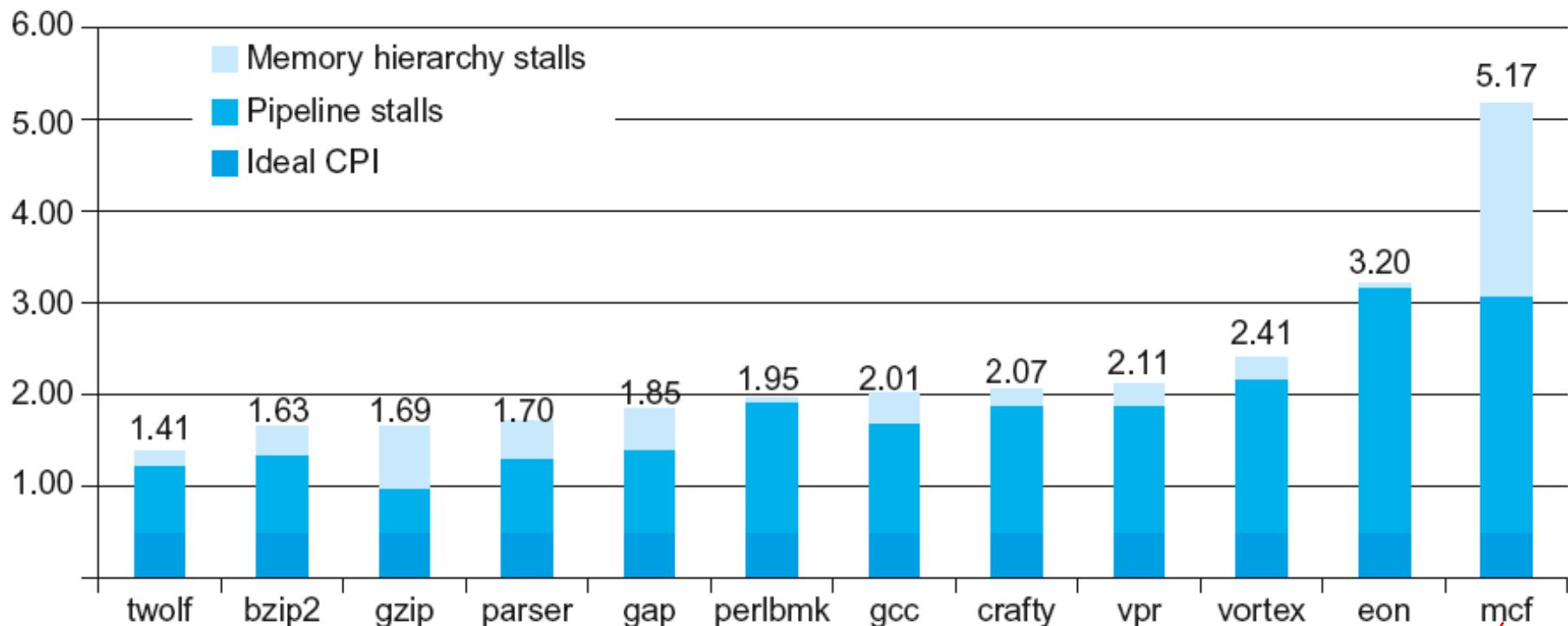
4.7.3 ARM Cortex-A8 Cevovod v celoti



4.7.3 ARM Cortex-A8 Cevovod – Minnespec Benchmarks

Meritve CPI na Minnespec Benchmarks:

- poenostavljen SPEC2000 (bistveno manjša količina vhodnih podatkov)



Idealni CPI = 0.5, v praksi pa izmerimo :

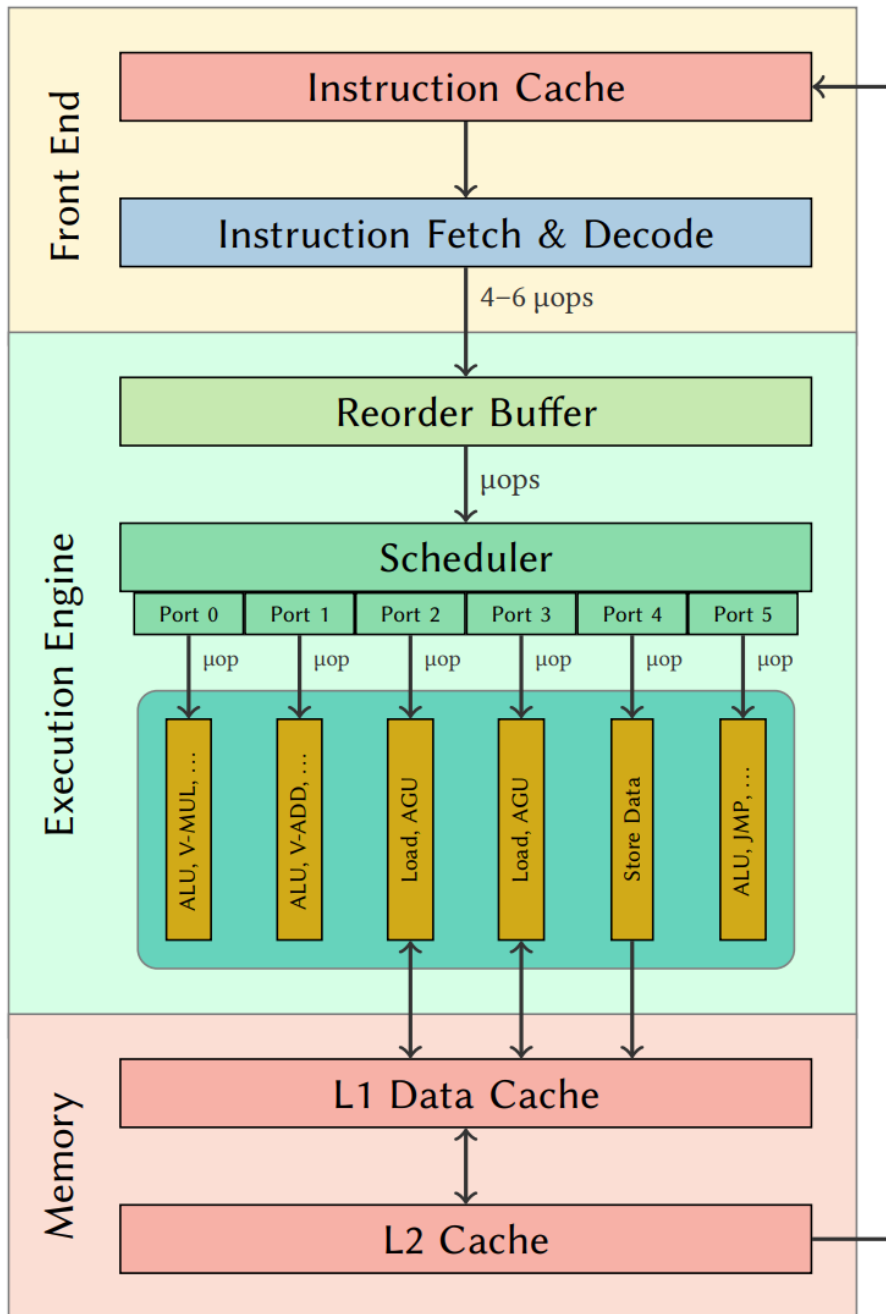
- 1.4 (min),
- 2.0 (median)
- 5.2 (max)...

181.mcf
SPEC CPU2000 Benchmark Description File
Benchmark Program General Category
Combinatorial optimization / Single-depot vehicle scheduling

4.7.3 Cortex A8, Intel i7 (4.7.4), Cortex M7

Processor	ARM A8	Intel Core i7 920	ARM M7
Market	Personal Mobile Device	Server, cloud	Embedded
Thermal design power	2 Watts	130 Watts	2.5 Watts
Clock rate	1 GHz	2.66 GHz	0.480 GHz
Cores/Chip	1	4	1-2
Floating point?	No	Yes	Yes (SP,DP)
Multiple issue?	Dynamic	Dynamic	Dynamic
Peak instructions/clock cycle	2	4	2
Pipeline stages	14	14	6
Pipeline schedule	Static in-order	Dynamic out-of-order with speculation	Static in-order
Branch prediction	2-level	2-level	present, no info
1 st level caches/core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D	16 KiB I, 16 KiB D
2 nd level caches/core	128-1024 KiB	256 KiB	
3 rd level caches (shared)	-	2- 8 MB	

4.7.4 Core i7 superskalarni cevovod - poenostavljen



1. Instruction Fetch (16 bajtov)

2. Predecode Stage (bajti \rightarrow x86 ukaze)

3. μ -op decode (x86 ukazi \rightarrow μ -op)

5. Izstavitev μ -op \rightarrow ROB in RP

6. Izvedba μ -op

7. Dovršitev

4.7.4 Core i7 superskalarni cevovod

14 stopenjski superskalarni cevovod:

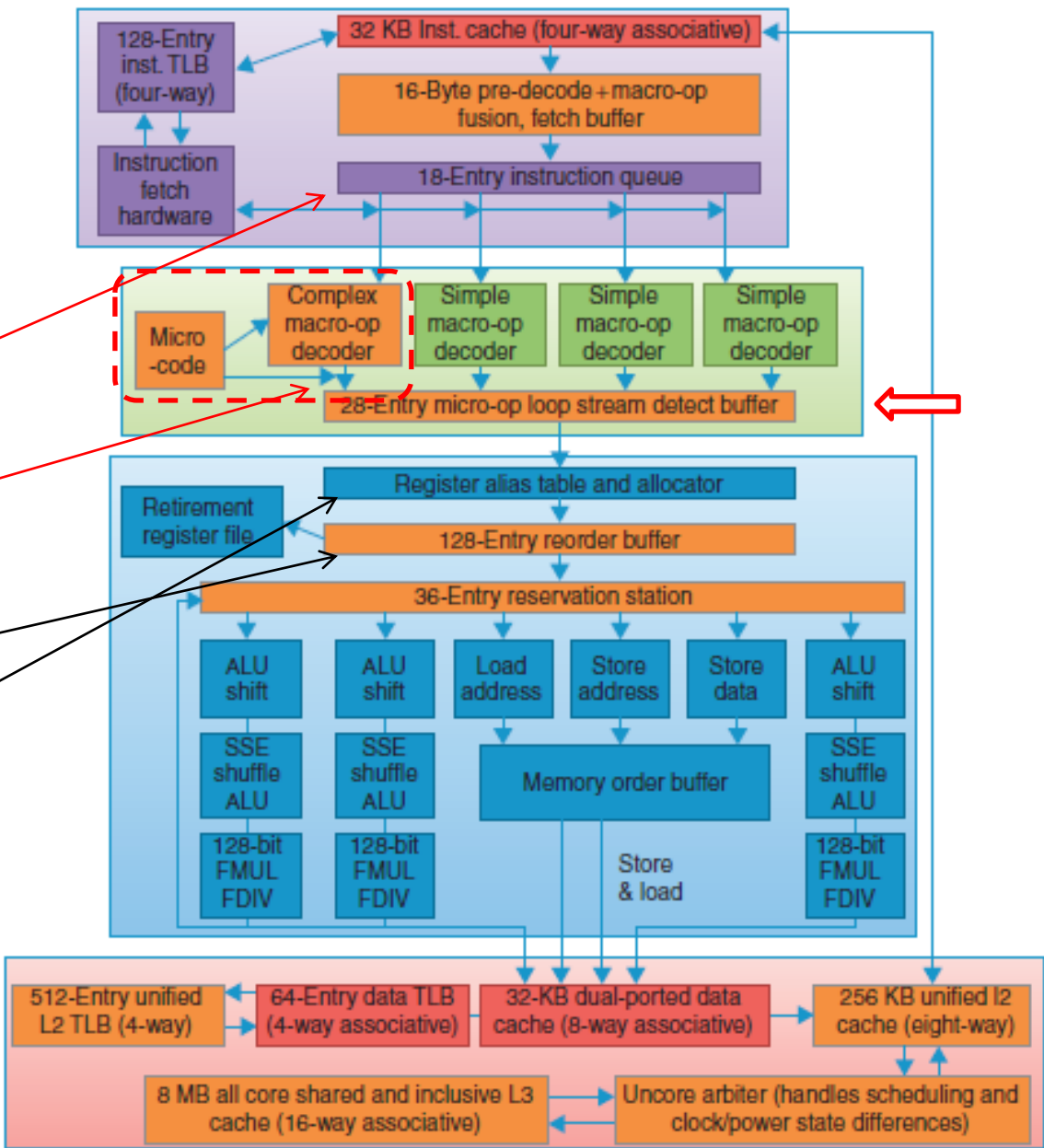
- dinamični več-izstavitveni (6 μ -op/cikel)
- dinamični „out-of-order“ cevovod
- špekulativno izvajanje ukazov

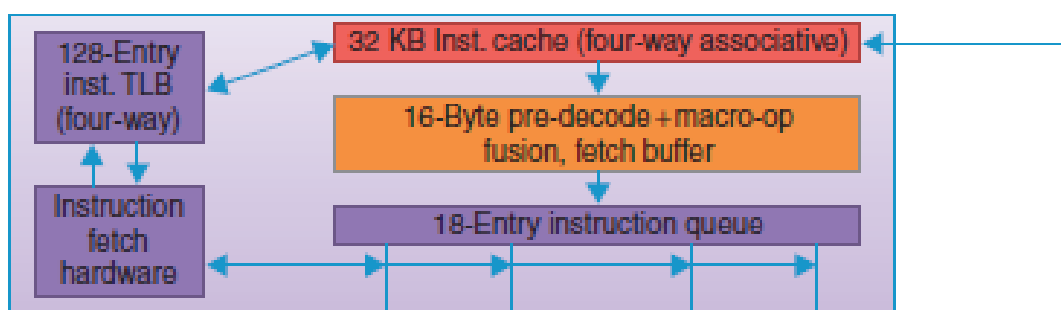
Osnovni problem :

CISC ukazi
 -> μ -operacije
 -> cevovod

- značilnosti:
 - PI (ROB)
 - ekspl. preimenovanje reg.

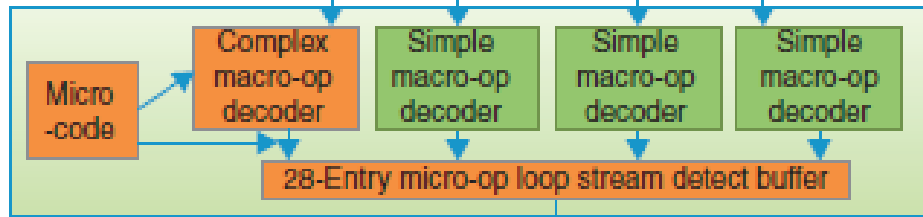
- „kazni“ v t_{cpe} ciklihi:
 - 15 napačno napovedan skok
 - 10 predp. L1 zgrešitev
 - 35 predp. L2 zgrešitev
 - 135 predp. L3 zgrešitev





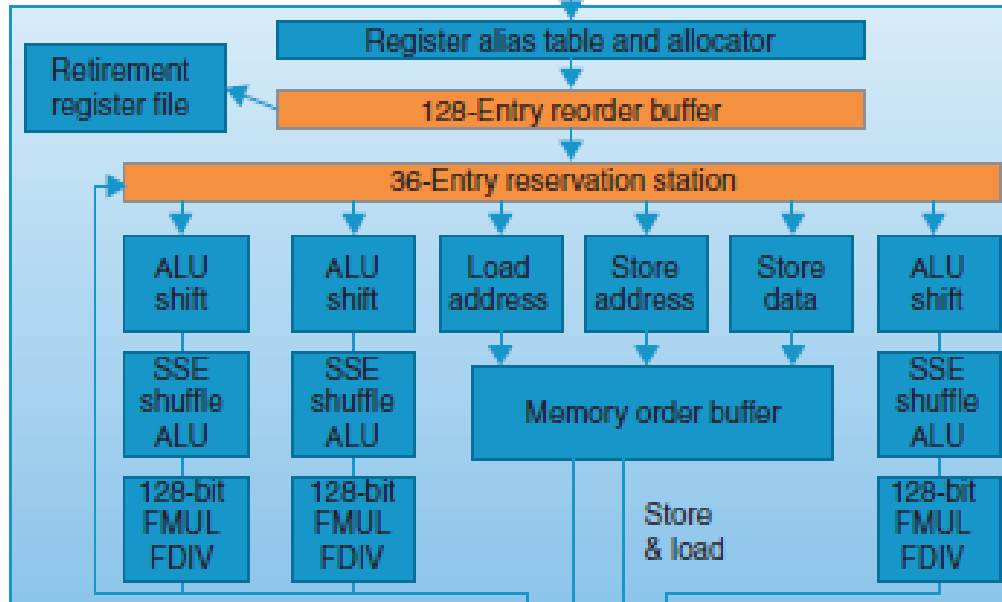
1. Instruction Fetch (16 bajtov)

2. Predecode Stage
(bajti -> x86 ukaze)



3. μ -op decode (x86 ukazi -> μ -op)

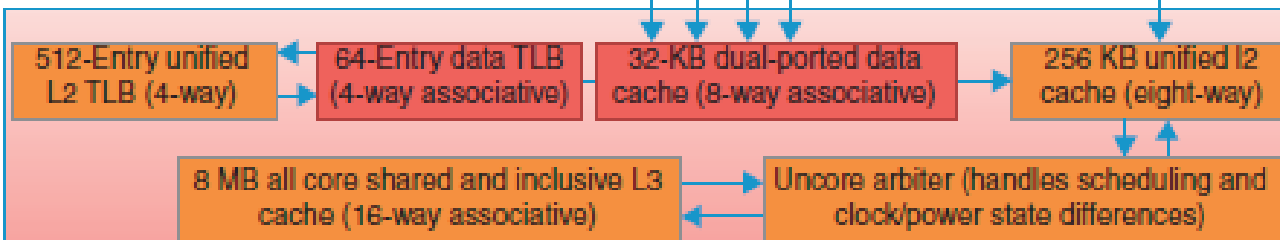
4. Loop Stream Detection



5. Izstavitev μ -op -> ROB in RP

6. Izvedba μ -op

7. Dovršitev



4.7.4 Core i7 superskalarni cevovod

Resource	i7 920 (Nehalem)	i7 6700 (Skylake)
Micro-op queue (per thread)	28	64
Reservation stations	36	97
Integer registers	NA	180
FP registers	NA	168
Outstanding load buffer	48	72
Outstanding store buffer	32	56
Reorder buffer	128	256

Sunny Cove

140 (skupna)

>125 (?)

352

Figure 3.39 The buffers and queues in the first generation i7 and the latest generation i7. Nehalem used a reservation station plus reorder buffer organization. In later microarchitectures, the reservation stations serve as scheduling resources, and register renaming is used rather than the reorder buffer; the reorder buffer in the Skylake microarchitecture serves only to buffer control information. The choices of the size of various buffers and renaming registers, while appearing sometimes arbitrary, are likely based on extensive simulation.

4.7.4 Core i7 superskalarni cevovod

4
191



€163

4 Cores, 8 Threads @2.66GHz, Nehalem.
Release date: Q3 2008.

VS



About



€236

4 Cores, 8 Threads @3.4GHz, Skylake.
Release date: Q3 2015.

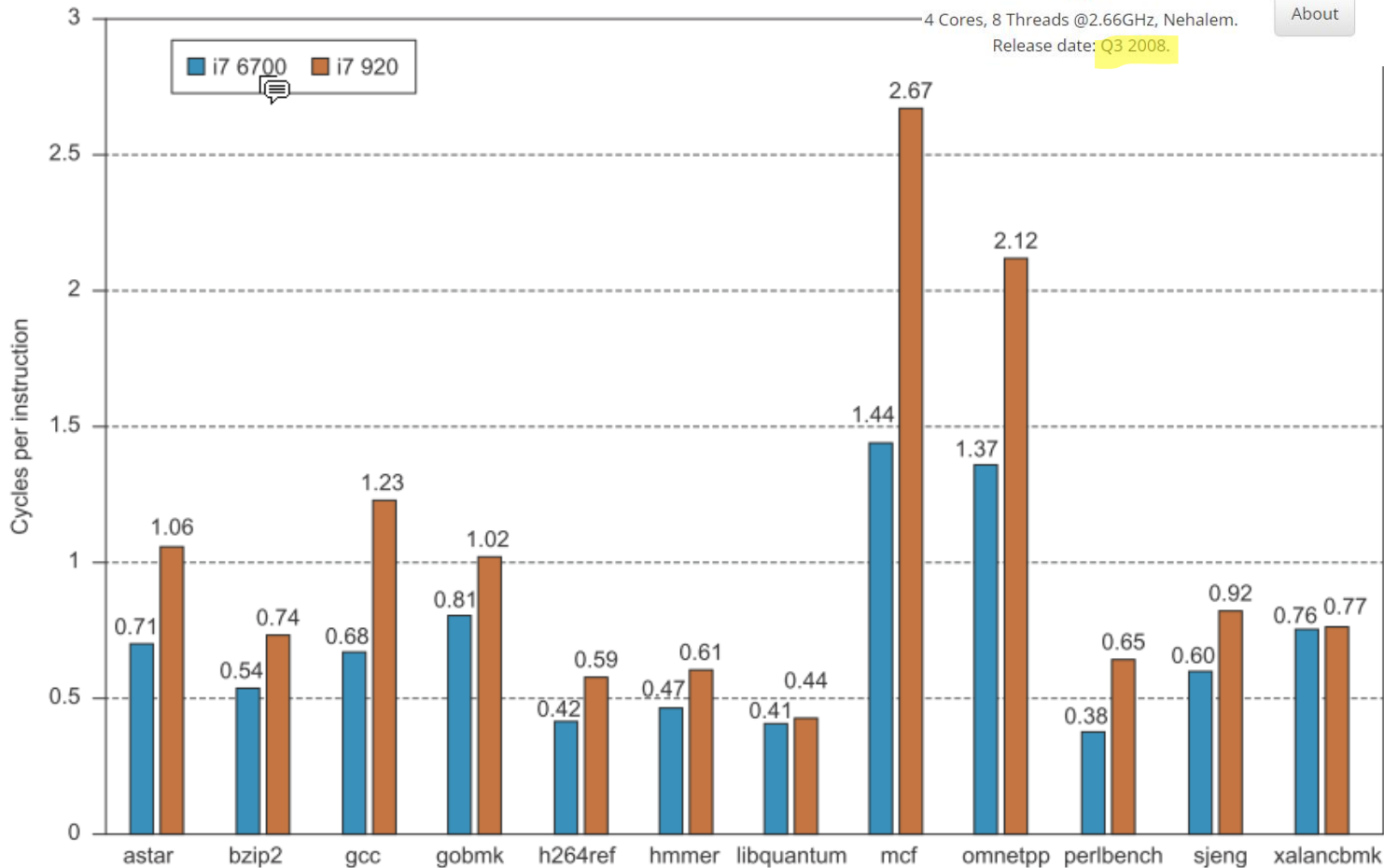
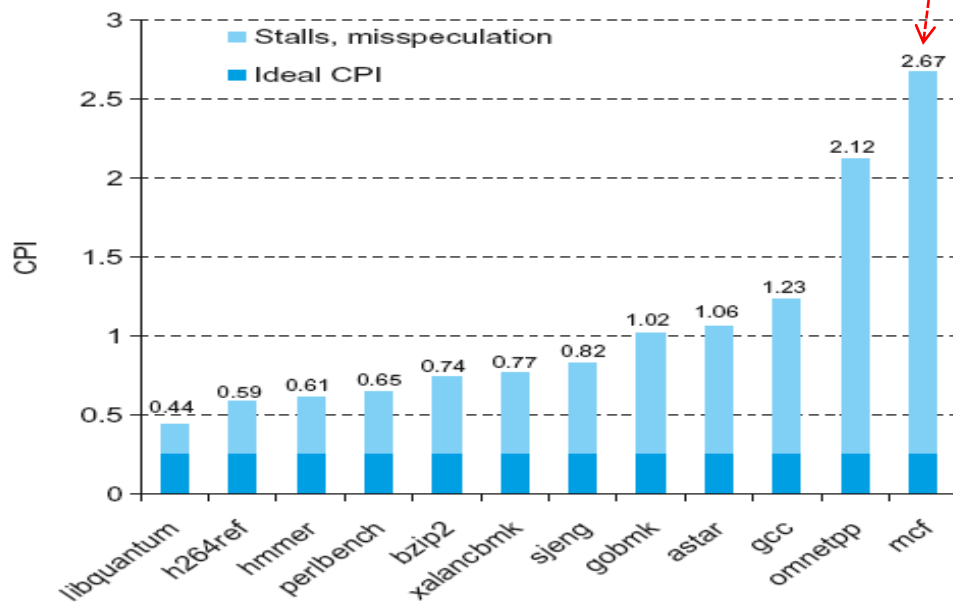


Figure 3.40 The CPI for the SPECint2006 benchmarks on the i7 6700 and the i7 920. The data in this section were collected by Professor Lu Peng and PhD student Qun Liu, both of Louisiana State University.

4.7.4 Core i7 cevovod

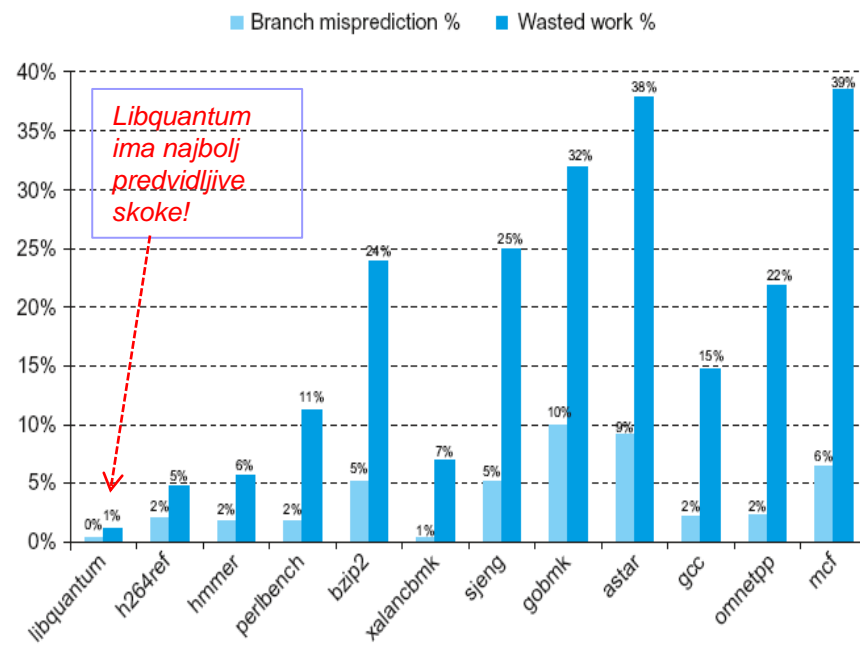
Meritve na SPEC2006 :

CPI



*mcf –
kombinatorična
optimizacija*

učinkovitost napovedi in
špekulativnega izvajanja



*Libquantum
ima najbolj
predvidljive
skoje!*

Idealni CPI = 0.25, v praksi pa izmerimo :

- 0.44 (min),
- 0.79 (median)
- 2.67 (max)...

Branch misprediction:

delež napačnih skočnih napovedi
0% (min), 2% (median) 10% (max)

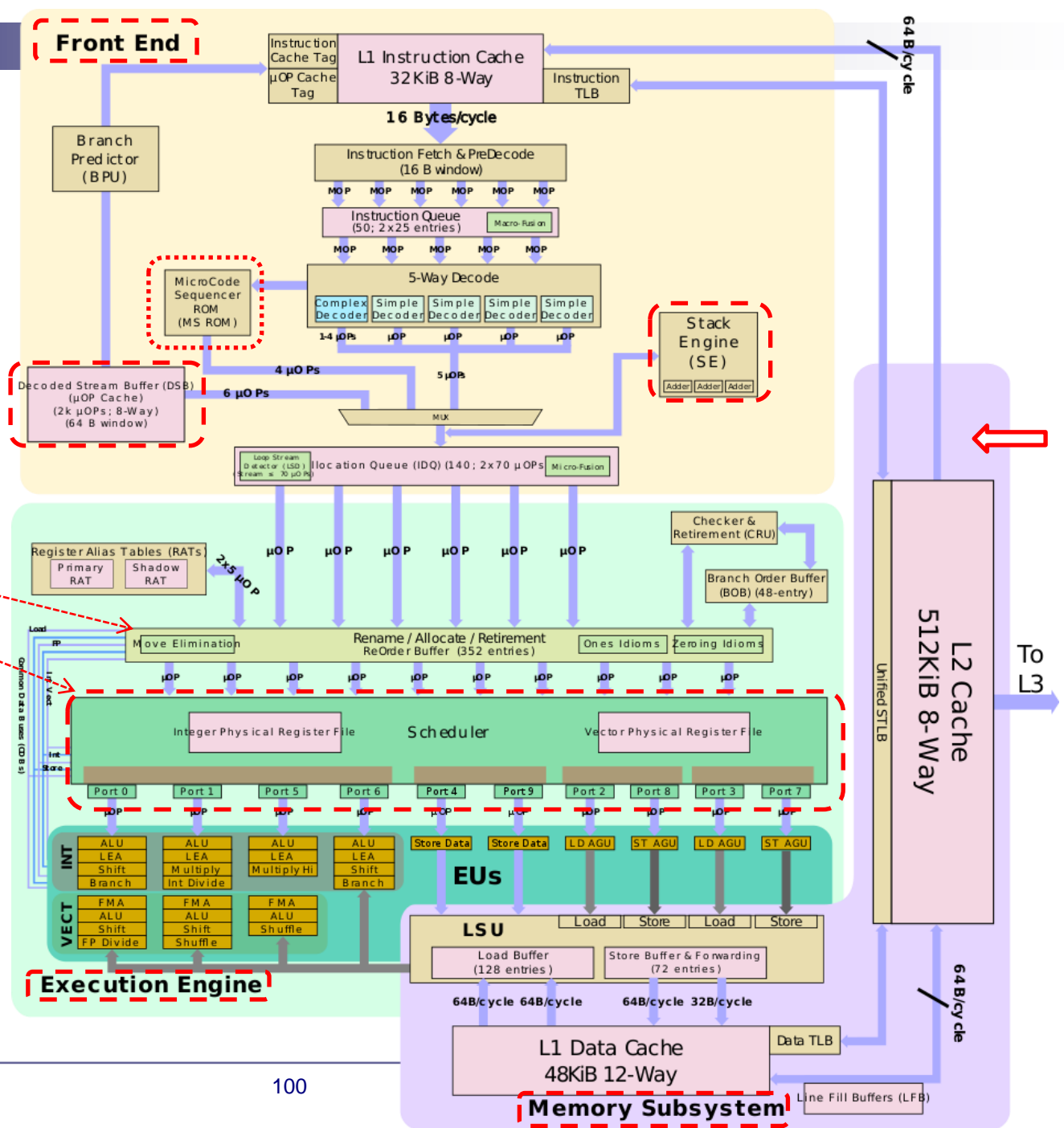
Wasted work:

delež zavrženih μ -operacij
1% (min), 18% (median) 39% (max)

4.7.4 Core i7 superskalarni cevodod

14-19 stopenjski superskalarni cevodod: „Sunny Cove“ (2019)

- ROB (352)
- centralna RP (160) – 10 izhodov
- 280 registrov (ekspl. preimenovanje)



Primer x86 zbirnika :

```
void function1() {  
    int A = 10;  
    A += 66;  
}
```

function1:

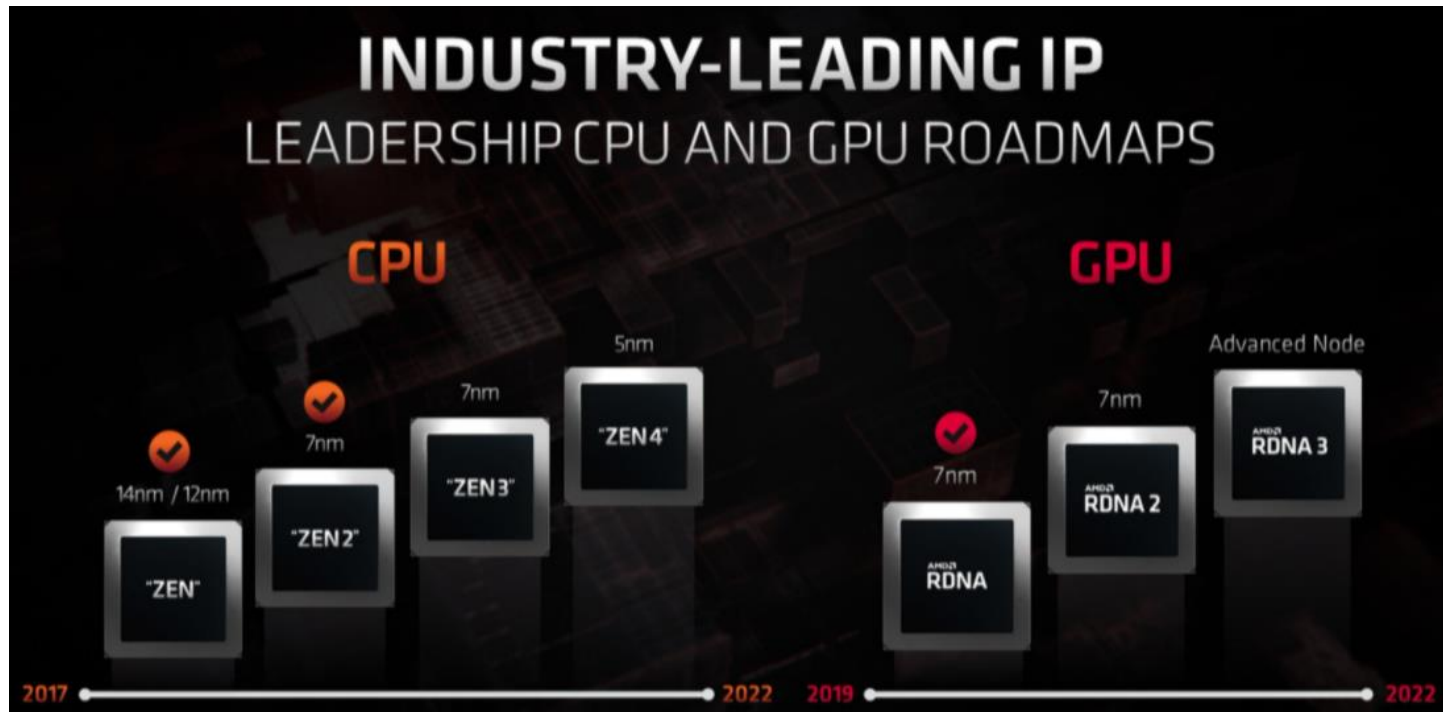
```
1 pushl %ebp #  
2 movl %esp, %ebp #,  
3 subl $4, %esp #,  
4 movl $10, -4(%ebp) #, A  
5 leal -4(%ebp), %eax #,  
6 addl $66, (%eax) #, A
```

...

1. push ebp
2. copy stack pointer to ebp
3. make space on stack for local data
4. put value 10 in A (this would be the address A has now)
5. load address of A into EAX (similar to a pointer)
6. add 66 to A

Vir: <http://www.hep.wisc.edu/~pinghc/x86AssmTutorial.htm>

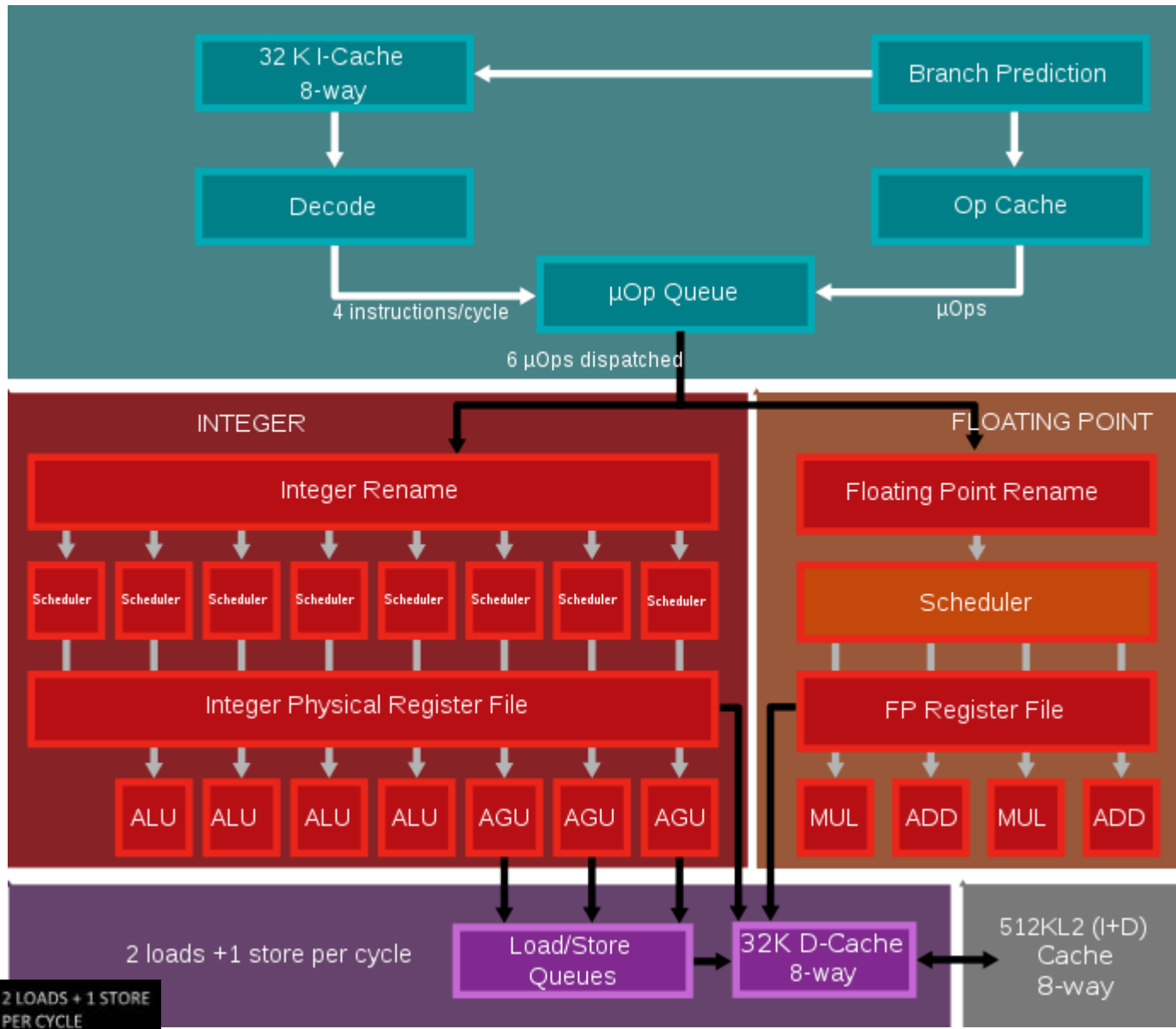
4.7.5 AMD „Zen“ mikroarhitekture



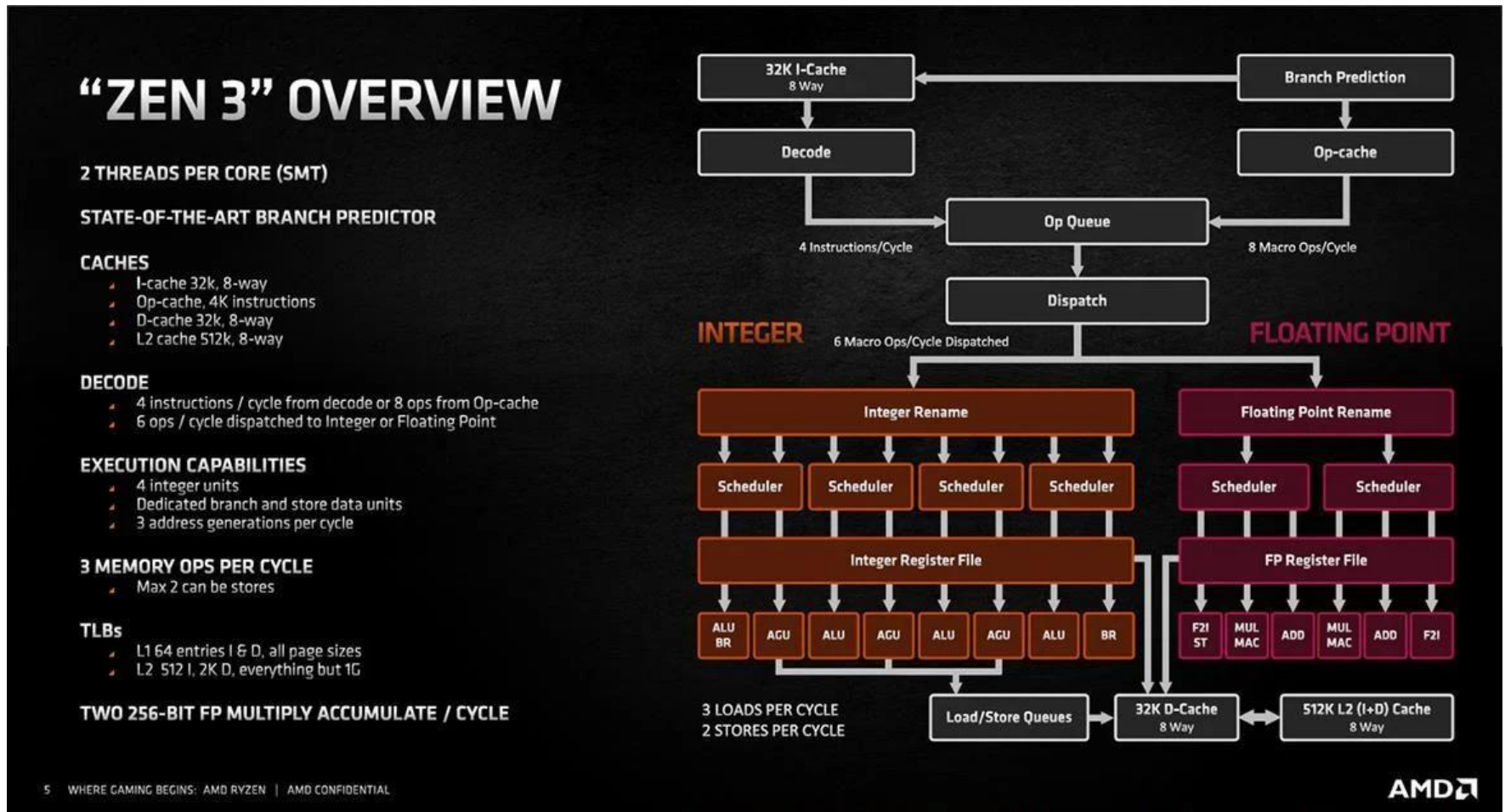
Novosti v „Zen“:

- „Design from scratch“
- 40% višji IPC od predhodnika
- SMT
- Izboljšan predpomnilniški sistem
- Evolucija: Zen+, Zen2, Zen 3, Zen 4, Zen 5

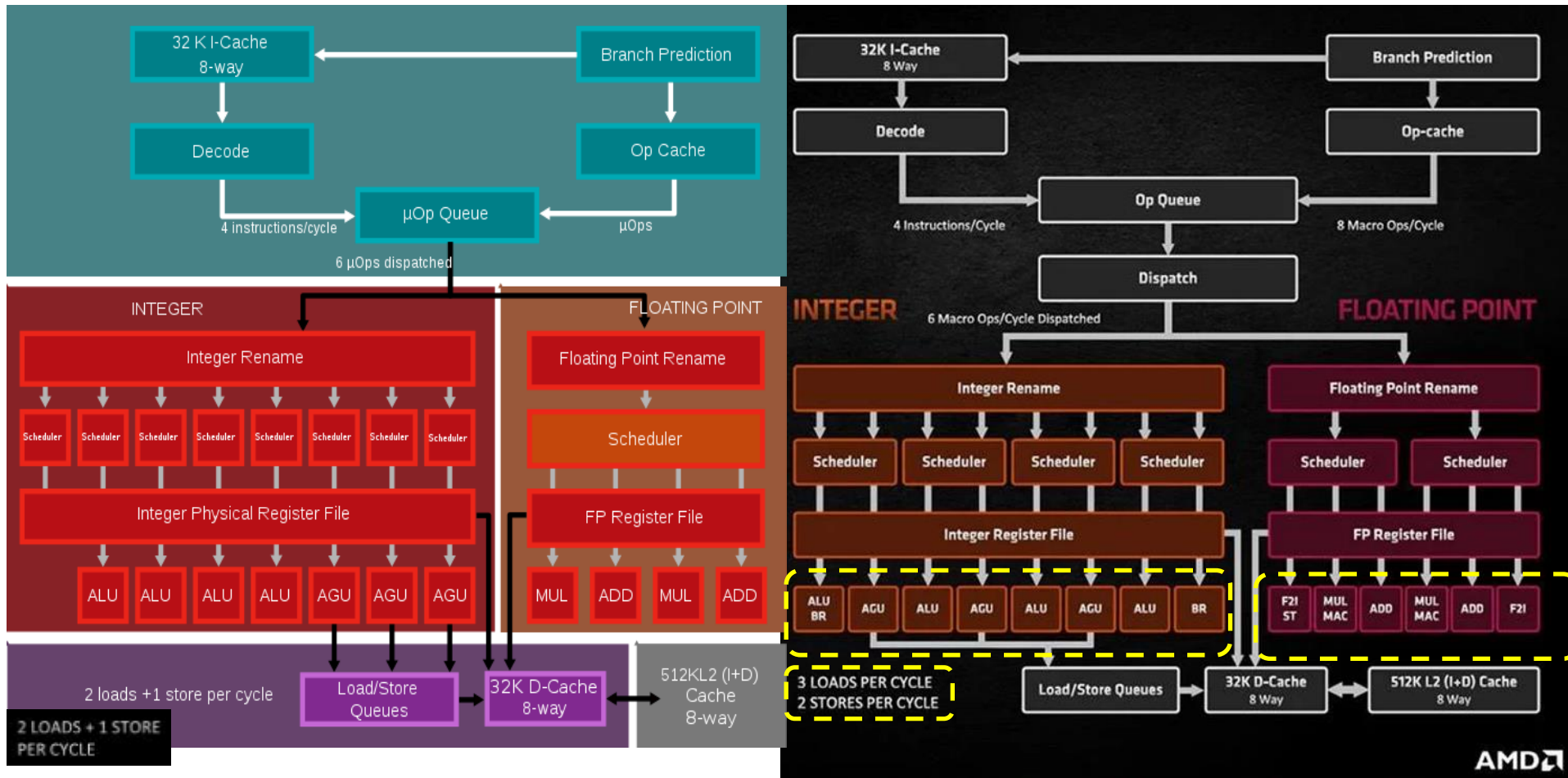
4.7.5 Zen 2 mikroarhitektura – poenostavljen model



4.7.5 Zen 3 mikroarhitektura – poenostavljen model

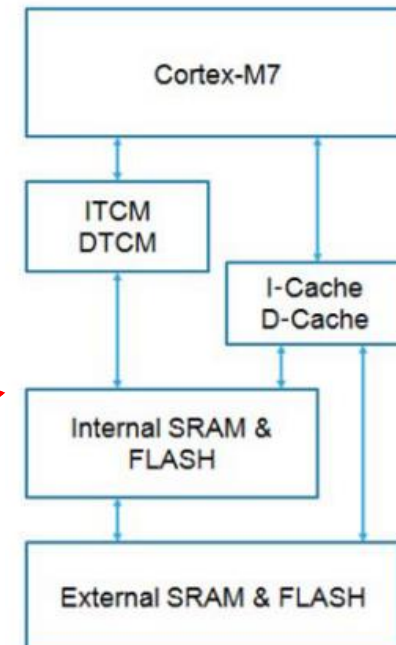
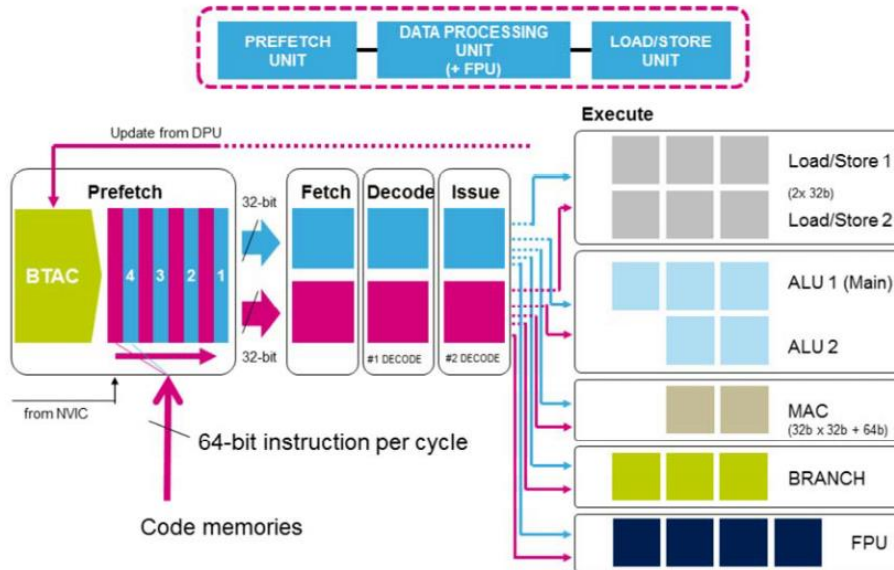


4.7.5 Zen 2 vs Zen 3 mikroarhitekturi – poenostavljena modela



4.7.6 ARM Cortex M7 mikroarhitektura (STM32H750)

ARM Cortex-M7 → Dual-issue



Značilnosti:

- „Dual issue in-order 6 stage pipeline“
 - Fetch 3 stopnje, Execute 3 stopnje
- Pomnilniška hierarhija
- Prefetch enota s skočno predikcijo

4.8 Omejitve paralelizma na nivoju ukazov

■ Izrazita prednost:

- transparentnost

■ Slabost:

- stopnja paralelizma na nivoju ukazov je omejena zaradi končnega števila ukazov med dvema napačnima napovedima.

Z več tranzistorji se lahko poveča:

- ukazno okno in
- natančnost napovedi skokov,
- tudi poraba

vendar od neke točke naprej to le malo pomaga.

Praksa:

število izstavljenih ukazov pri superskalarnih procesorjih se kljub napredku tehnologije praktično ne povečuje (je pod 10).

Količina paralelnosti na nivoju ukazov v programih je omejena, odvisna od algoritmov in ne tehnologije !

- V raziskavah je bilo ocenjeno, da je povprečni IPC, ki bi bil dosegljiv na resničnem računalniku, približno 5 do 10. To pa je več kot danes dosežejo najzmogljivejši superskalarni računalniki.

	realni IPC	teoretični IPC	Komentar
P4	0.8	3	31stopenj,
Core2 Core iX	1.5	4	14stopenj

- Vzroka za nizek realni IPC pri P4 sta :
 - zelo dolg cevovod in
 - zgrešitve v predpomnilniku.
- Večji IPC še ne pomeni nujno bolj zmogljiv procesor. Zmogljivost je odvisna še od frekvence ure f_{CPE} .
- Število ukazov, ki jih procesor izvrši v eni sekundi je
$$\text{št.ukazov/sek} = f_{CPE} \times IPC$$
- Realnost:
 - večji IPC pa je večinoma mogoče doseči le z nižjo frekvenco ure f_{CPE} .

Primerjava Intelovih procesorjev :

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
Core i5(Nehalem)	2010	3300MHz	14	4	Yes	2	87W
Core i5(Ivy Bridge)	2012	3400MHz	14	4	Yes	4	77W

Core i9(Coffee Lake) 2018 5000MHz 14 Yes 8 127W

Core i9(RaptorCove) 2022 5800MHz 12

- kompleksnost dinamičnega razvrščanja ukazov in špekulacije zahteva dodatno logiko in porabo.
- morda je manj enostavnejših jeder lahko kdaj tudi boljša rešitev (ARM,...)

Prelomnica:

Intel P4 CPE: snovalci izbirajo med (okoli l. 2000):

- povečanjem f_{cpe}
 - (poraba, toplota)
- **2 CPE na 1 čip**
 - (dvojni stroški, ni programov)
- dodajanje **FE**
 - (samo po sebi ne pomaga)
- **daljši cevovod**
 - (povečajo se nevarnosti)
- **večnitnost** (se zdi še edina možnost):
 - izkoristimo HW, ki sicer stoji
 - **5% več površine->25% poveča zmoglj.**

Prelomnica I.2000 (Intel P4):

- V mnogih situacijah (programih) obstaja paralelizem tudi na višjem nivoju, ki ga na nivoju ukazov ni mogoče izkoristiti.
- To je paralelizem na nivoju niti, kjer se izvrševanje razdeli v več ločenih poti (niti):
 - Zakaj (I.2000) ?:
 - 5% več površine prinese 25% povečanje zmogljivosti
 - I. 2000 je bila to še najbolj sprejemljiva rešitev

4.9 Paralelizem na nivoju niti (večnitnost)

- **Nit** („thread“) je zaporedje ukazov, ki se lahko izvršuje neodvisno od drugih ukazov. Nit je lahko:
 - del programa, ki ga sestavlja več procesov
 - samostojen program
- Večnitnost **ni transparentna** !
poskrbi programer (razen program = nit)
- Večnitnost na strojnem nivoju (Hardware Multithreading) omogoča, da si več niti deli funkcijske enote enega procesorja.
- Vsaka nit v procesorju mora imeti svoje stanje, ki je neodvisno od stanj drugih niti.

- **Stanje vsake niti mora imeti svojo kopijo:**
 - programsko dostopnih registrov,
 - programskega števca (PC) in
 - potrebnih programsko nedostopnih registrov ter
 - svoje tabele strani.
- **Vse niti si delijo pomnilnik in predpomnilnike:**
 - pri čemer uporabljajo mehanizem navideznega pomnilnika.
- **Vsaka nit vidi kot da sama uporablja procesor:**
 - en procesor je tako videti kot več logičnih procesorjev.
- **Preklop med nitmi je bistveno hitrejši (t.j. trenuten) od preklopa med programi v multiprogramskem načinu, ki lahko traja od nekaj sto do tisoč urinih period.**

- Obstaja več načinov realizacije strojne večnitnosti:
 - Časovna večnitnost
 - drobnozrnata
 - grobozrnata
 - Istočasna večnitnost (večizstavitvene CPE)

4.9 Paralelizem na nivoju niti

Primer večnitnega programa :

```
#include <stdio.h>          /* standard I/O routines          */
#include <pthread.h>        /* pthread functions and data structures */
```

```
/* include <stdio.h>  /* standard I/O routines  */
#include <pthread.h> /* pthread funcs & structures */
```

```
/* function to be executed by the new thread */
```

```
void* do_loop(void* data)
{
    int i; /* counter, to print numbers */
    int j; /* counter, for delay */
    int me = *((int*)data); /* thread ID */

    for (i=0; i<10; i++) {
        /* delay loop */
        for (j=0; j<500000000; j++);
        printf("%d' - Got '%d'\n", me, i);
    }

    /* terminate the thread */
    pthread_exit(NULL);
}
```

```
'3' - Got '0'
'2' - Got '0'
'1' - Got '0'
'3' - Got '1'
'1' - Got '1'
'2' - Got '1'
'1' - Got '2'
'3' - Got '2'
'2' - Got '2'
'1' - Got '3'
'2' - Got '3'
'3' - Got '3'
'1' - Got '4'
'3' - Got '4'
'2' - Got '4'
'1' - Got '5'
'3' - Got '5'
'1' - Got '6'
'3' - Got '6'
'2' - Got '5'
'1' - Got '7'
```

```
/* like any C program, program's execution begins in main */
int main(int argc, char* argv[])
{
```

```
    int        thr_id1, thr_id2; /* thread ID - new thread */
    pthread_t  p_thread;        /* thread's structure */
    int        a                = 1; /* thread 1 id */
    int        b                = 2; /* thread 2 id */
    int        c                = 3; /* thread 3 id */
```

```
/* create a new thread & execute 'do_loop()' */
```

```
thr_id1 = pthread_create(&p_thread, NULL, do_loop, (void*)&a);
```

```
/* create a new thread that will execute 'do_loop()' */
```

```
thr_id2 = pthread_create(&p_thread, NULL, do_loop, (void*)&b);
```

```
/* run 'do_loop()' in the main thread as well */
```

```
do_loop((void*)&c);
```

```
return 0;
```

```
}
```

Opis: funkcija do_loop:

- se izvaja v dveh nitih in še v glavnem programu,
- Vsaka nit z drugim IDjem (1,2,3)
- povsod z zakasnitvijo šteje do 10 in potem se konča

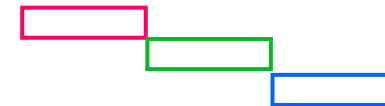
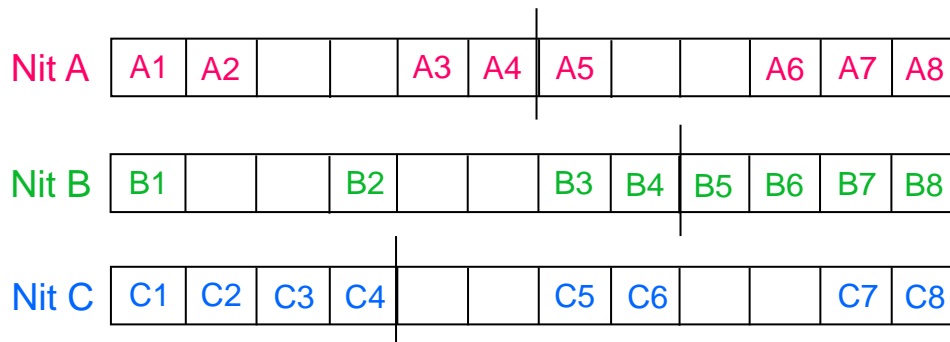
4.9.1 Drobnozrnata večnitnost

(„fine-grained multithreading“).

Procesor preklaplja med nitmi vsako urino periodo.

- Niti se preklaplajo enakopravno ena za drugo.
- Niti, pri katerih se pojavi čakanje zaradi kateregakoli vzroka, se preskočijo.
- Za vsako nit se mora pri preklopu shraniti poleg registrov tudi popolno stanje cevovoda.
- Dobra stran tega načina je, da se prikrije čakalne periode, ker se takrat izvajajo ukazi drugih niti.

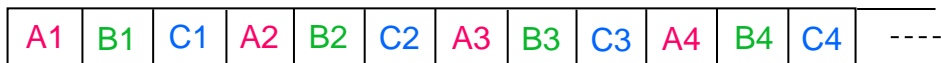
Izvajanje treh niti posamezno pri drobnozrnati večnitnosti :



Niti posamezno = $36 t_{CPE}$

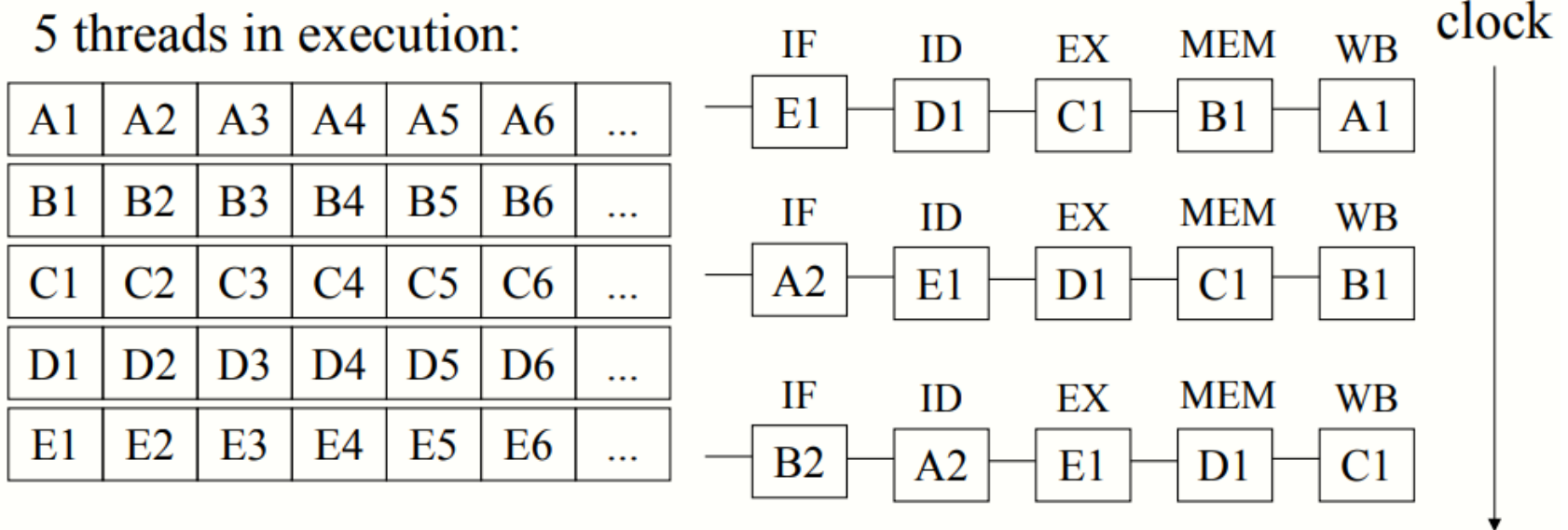
Urine periode → Posamezne niti: $18 t_{CPE} = 6$ (do A4) + 8 (do B4) + 4 (do C4)

Drobnozrnata večnitnost (primer do A4,B4,C4)



Urine periode → Večnitnost: $12 t_{CPE}$

Idealna situacija – 5 stopenjski cevovod in izvajanje petih niti pri drobnozrnati večnitnosti :



Nevarnosti ?
Pogosta situacija ?

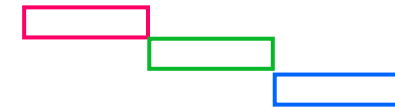
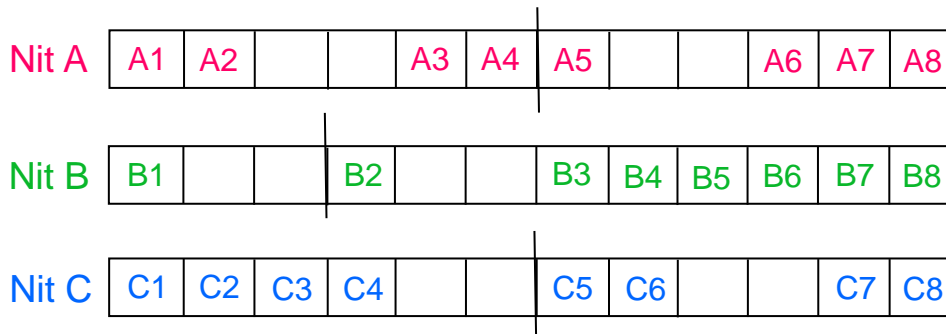
4.9.2 Grobozrnata večnitnost („coarse-grained multithreading“).

Procesor preklopi na naslednjo nit samo kadar pride do daljšega čakanja, kot npr. pri zgrešitvah v predpomnilniku L2.

- Procesor je enostavnejši, ker ob preklopu ni treba za vsako nit shraniti celotnega cevovoda, ker lahko počakamo, da se izprazni.
- Preklopov je manj in so zato lahko počasnejši.
- Izvrševanje posamezne niti se ne upočasni, če nima daljših čakanj.

- Slabost tega načina je, da ne prikrije krajših čakanj. Pri krajših čakanjih procesor stoji.
- Ob preklopu se mora cevovod napolniti z ukazi nove niti.
- Grobozrnata večnitnost je zato uporabna pri daljših čakanjih, kjer je čas polnjenja cevovoda zanemarljiv v primerjavi s časom čakanja.

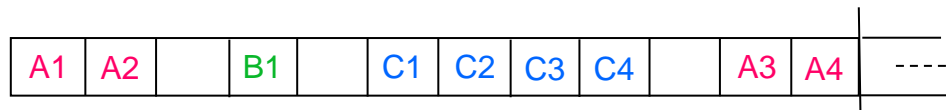
Izvajanje treh niti posamezno in pri grobozrnati večnitosti



36 urinih period

Urine periode →

Posamezne niti: $15 t_{CPE} = 6$ (do A4) + 3 (do B1+) + 6 (do C4)



32 urinih period

Urine periode →

Primer do A4, B1, C4:

Večnost: $12 t_{CPE}$

Ugotovitve in primerjava:

- **Drobnozrnata večnitnost:**
 - lahko pri k-stopenjskem cevovodu in k-nitih cevovod deluje s polno hitrostjo brez čakanja, ker je v vsakem trenutku v cevovodu samo po en ukaz vsake niti.
- **Grobozrnata večnitnost:**
 - Pri majhnem številu niti je boljša grobozrnata večnitnost.
- Pri obravnavi drobno in grobozrnate večnitnosti smo predpostavljali, da procesor izstavi samo en ukaz vsako urino periodo.
- Pri večizstavitvenih procesorjih z dinamičnim razvrščanjem (superskalarni procesorji) pa je možna istočasna večnitnost.

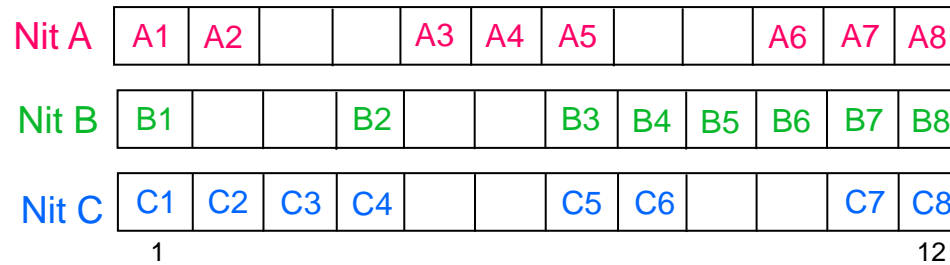
4.9.3 Istočasna večnitnost

(SMT-“**S**imultaneous **M**ulti**T**hreading“, Intel: „Hyperthreading“).

Pri superskalarnih procesorjih je običajno na voljo več funkcijskih enot, kot jih lahko izkoristi ena nit.

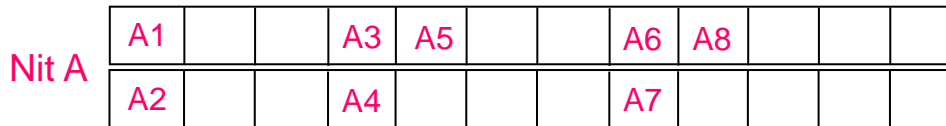
- Zasnova superskalarnega procesorja omogoča, da se lahko izstavlja **več ukazov**, ki lahko pripadajo različnim nitim.
- Med ukazi različnih niti **ne more priti do medsebojnih odvisnosti**, kar poenostavi izstavljanje ukazov.
- Ukazi vsake niti pa se morajo **dovršiti posebej**, zato ima vsaka nit svoj preureditveni izravnalnik.
- Istočasna večnitnost je podobna drobnozrnati večnitnosti, vendar se niti ne izmenjujejo v vsakem ciklu, temveč ko pride pri niti do čakanja.

Izvajanje treh niti (vsake posebej) pri 2-izstavitvenem superskalarnem procesorju
(brez istočasne večnitnosti)

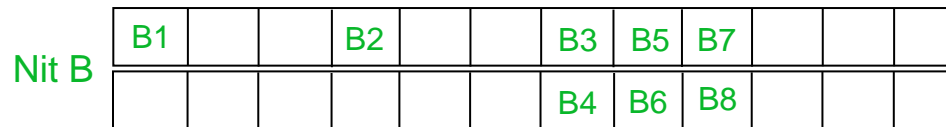


$3 \cdot 12 = 36$ urinih period

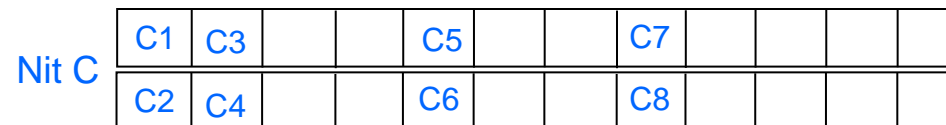
Urine periode →



9 urinih period



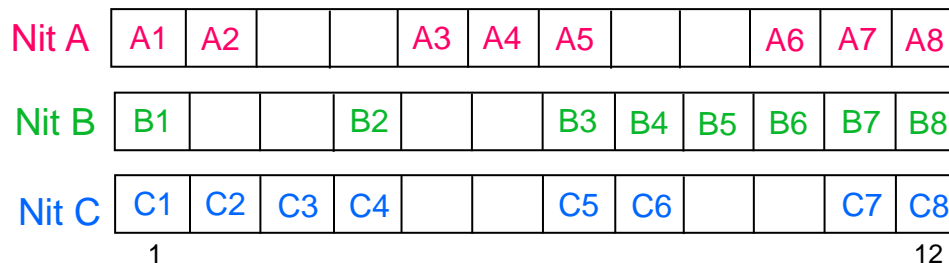
9 urinih period



8 urinih period

$2 \cdot 9 + 8 = 26$ urinih period

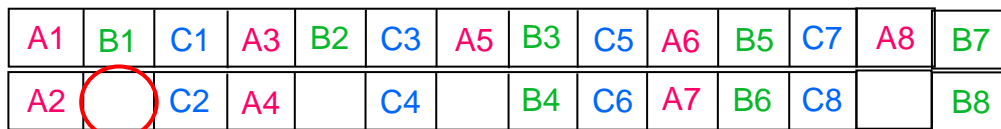
Primerjava: večnitost pri 2-izstavitvenem superskalarnem procesorju



3*12=36 urinih period

Urine periode →

Drobnozrnata
večnitost



14 urinih period

Grobozrnata
Večnitost I
(„preventiva“)



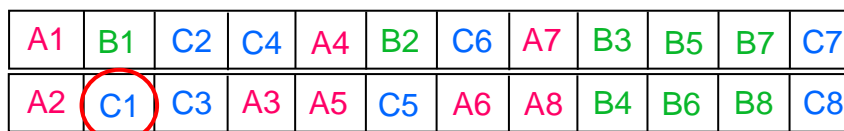
14 urinih period

Grobozrnata
Večnitost II
(„daljši zastoj“)



22 urinih period

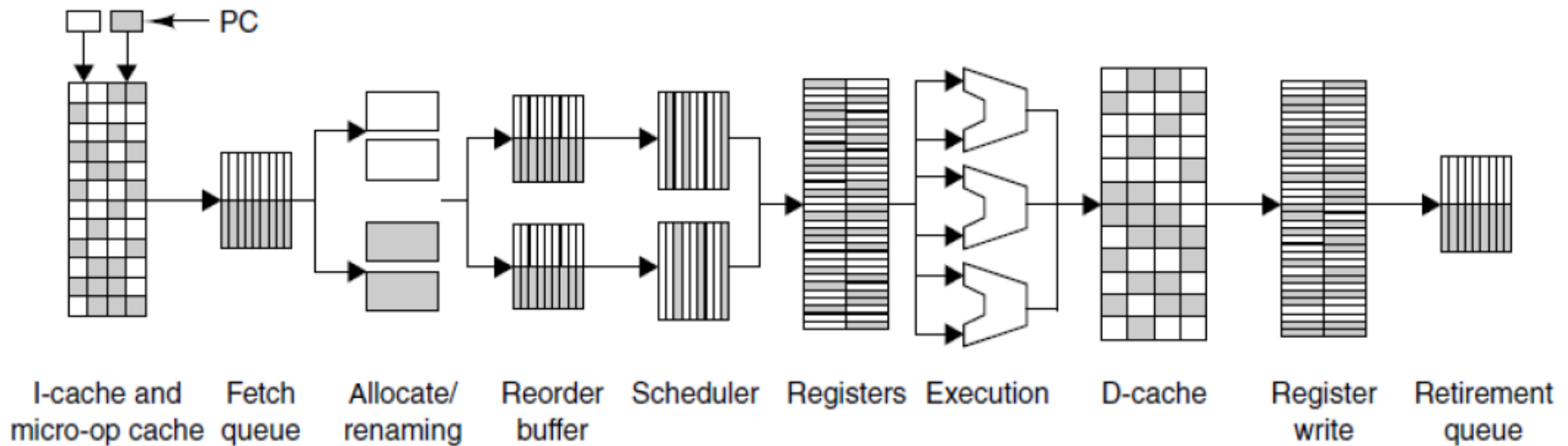
Istočasna
večnitost



12 urinih period

- Ker je večina današnjih procesorjev superskalarnih, je največkrat uporabljena istočasna večnitnost.
- Smatramo lahko, da istočasna večnitnost poveča zmogljivost superskalarnih procesorjev.
- Niti si delijo vire v procesorju na različne načine
 - viri so lahko fiksno razdeljeni med nitmi
 - vire si lahko delijo po pravilu “kdor prvi pride, prvi melje”
 - dodeljevanje virov je lahko dinamično – drugačna pravila

Istočasna večnitnost ali „Hyperthreading“ na Core i7



Niti si delijo vire v procesorju na različne načine

- viri so lahko fiksno razdeljeni med nitmi
- vire si lahko delijo po pravilu “kdor prvi pride, prvi melje”
- dodeljevanje virov je lahko dinamično – drugačna pravila

Istočasna večnitnost ali „Hyperthreading“ na Core i7 Primerjava „SMT“ vs „non-SMT“

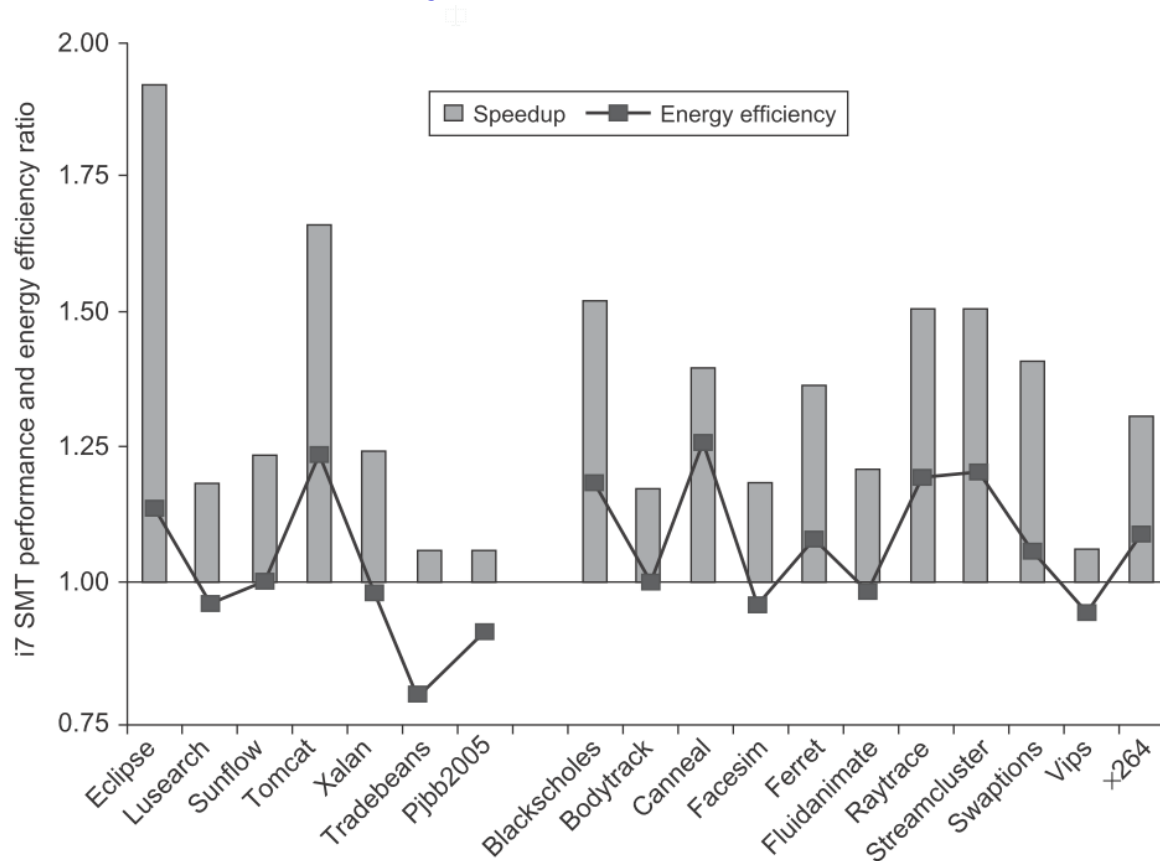


Figure 3.33: The speedup from using multithreading on one core on an i7 processor averages **1.28** for the Java benchmarks and **1.31** for the PARSEC benchmarks

Primer (kaže tudi na slabosti večnitnosti):

- Dve niti, ki za dobro delovanje potrebujeta vsaka $\frac{3}{4}$ predpomnilnika.
 - Če se izvajata ločeno:
 - delujeta dobro, z malo zgrešitvami v predpomnilniku, ki pa povzročajo čakanje.
 - Če tečeta istočasno:
 - je pri vsaki veliko število zgrešitev v predpomnilniku in je delovanje lahko precej slabše kot brez večnitnosti

Intelovi procesorji :

- Intel je prvič uporabil istočasno večnitnost („hyper-threading“) v 2-nitnem procesorju Pentium 4.
- Realizacija je zahtevala za 5% več tranzistorjev, po Intelovih podatkih pa se je hitrost povečala za 15 do 30%. (pri nekaterih programih pa tudi zmanjšala).
- Pri Core (Intel Core2) mikroarhitekturi je Intel večnitnost opustil, pri mikroarhitekturi Nehalem (Intel Core iX) pa je zopet uvedel istočasno večnitnost z dvema nitma.

Naslednji korak izkoriščanja paralelizma na nivoju ukazov so **večjedrni procesorji**:

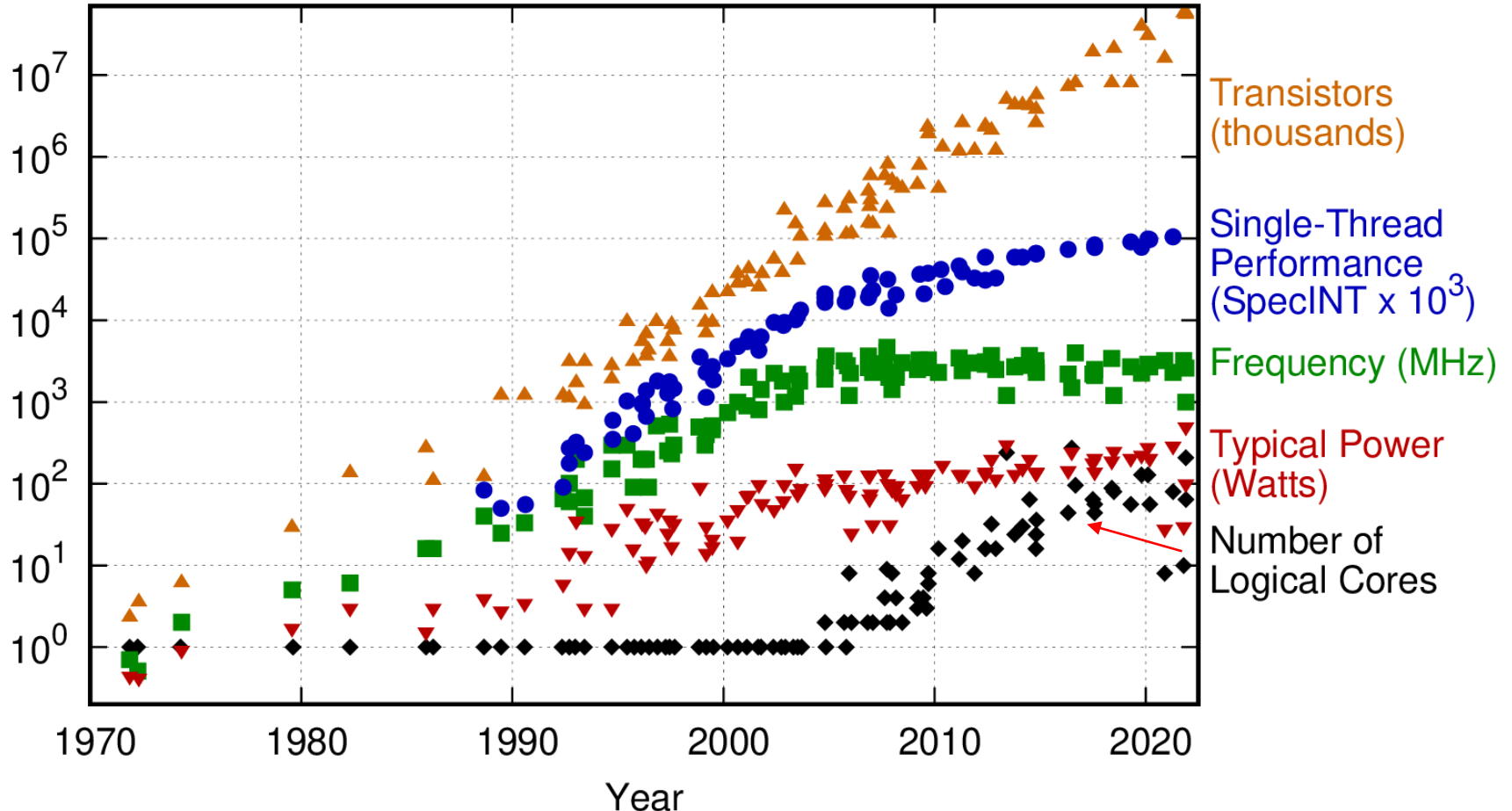
- procesi se izvajajo bolj ločeno
- si ne delijo več naprav !!!!
- imajo na čipu več neodvisnih CPE (jeder)
- niti tečejo na svojih CPE -> prava paralelnost

Večjedrnost :

- korist za proizvajalce :
 - ceneje duplicirati
 - »uporabniki naj se naučijo paralelnega programiranja«
- manj koristi za uporabnike:
 - »raje imam CPE z IPC=4, kot pa 8-jedrni procesor«

4.9 Pregled dosedanjega razvoja

50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

Vir: <https://raw.githubusercontent.com/karlrupp/microprocessor-trend-data/master/50yrs/50-years-processor-trend.png>