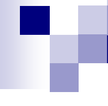
ORGANIZACIJA RAČUNALNIKOV

4. Paralelizem na nivoju ukazov

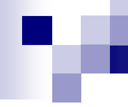
© 2024 Rozman - FRI



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 : $MIPS = \frac{\text{fcpe} \uparrow}{\downarrow \text{CPI } 10^6}$

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:

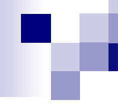
CISC	RISC
ADD (A),(B),(C)	LDR R1,(A)
	LDR R2,(B)
	ADD R3,R1,R2
	STR R3,(C)
CISC = 1MIPS	RISC=4MIPS

Sicer čas lahko enak !!!!

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

Glavna prednost cevovoda (poleg pohitritve) je

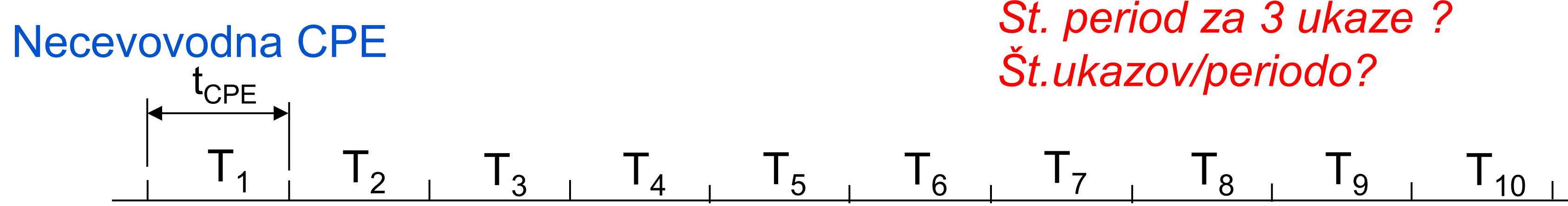
- ***TRANSPARENTNOST:***
  - ***ni posegov (v programe) !!!***



Izvrševanje treh ukazov pri necevovodni in cevovodni CPE s 5 stopnjami

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

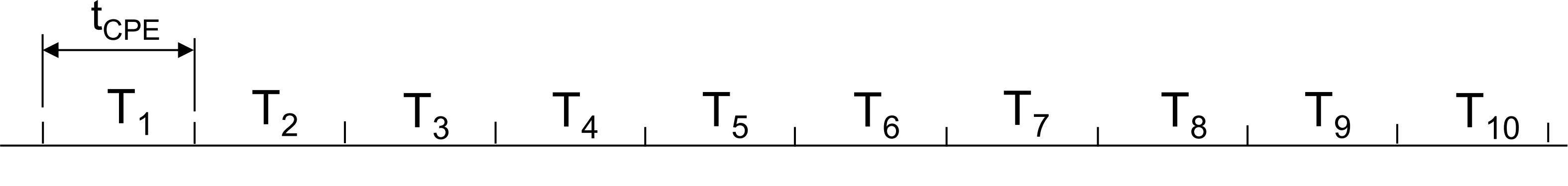
Necevovodna CPE



ukaz 1
ukaz 2

ukaz 3

Cevovodna CPE (5 stopenj)



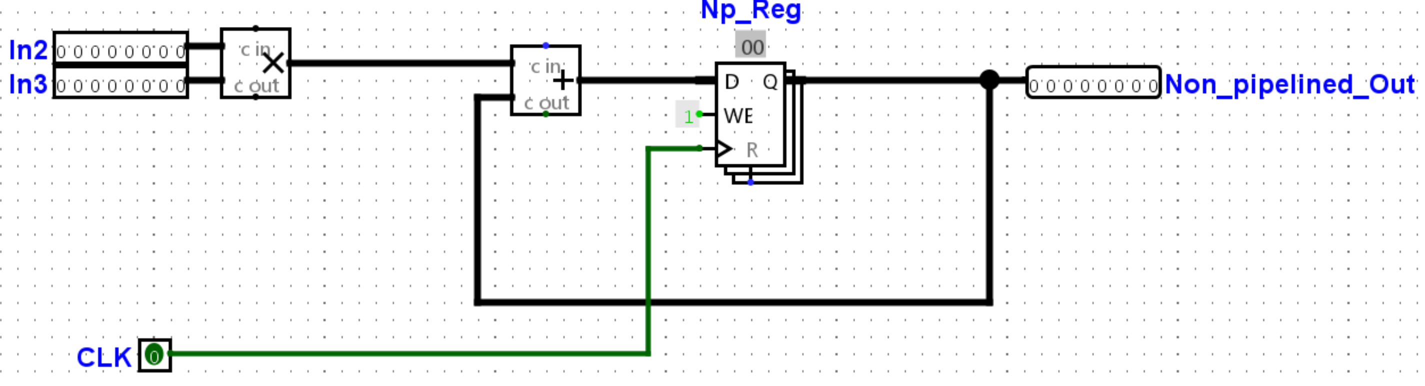
	T_1	T_2	T_3	T_4	T_5	T_6	T_7
ukaz 1	korak 1	korak 2	korak 3	korak 4	korak 5		
ukaz 2		korak 1	korak 2	korak 3	korak 4	korak 5	
ukaz 3			korak 1	korak 2	korak 3	korak 4	korak 5

© 2024 Rozman - FRI

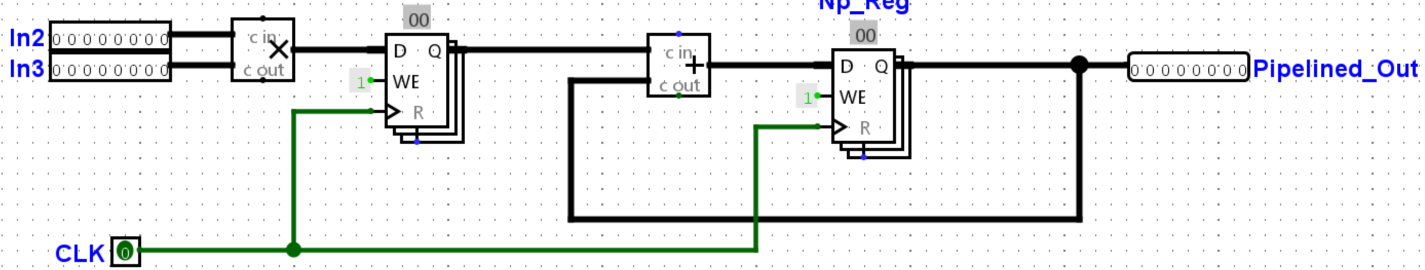
4.1 Zgradba cevovodne CPE

Ideja cevovodnega procesiranja – Logisim primer: SUM = A x B + SUM

Necevovodna izvedba



Cevovodna izvedba



Signal Name	Sign...
CLK	0
D Flip-Flop(130,150)	0
Non_pipelined_Out[7..0]	0
Pipelined_Out[7..0]	0

140,0 µs 160,0 µs 180,0 µs 200,0 µs 220,0 µs 240,0 µs 260,0 µs 280,0 µs 294999 ns

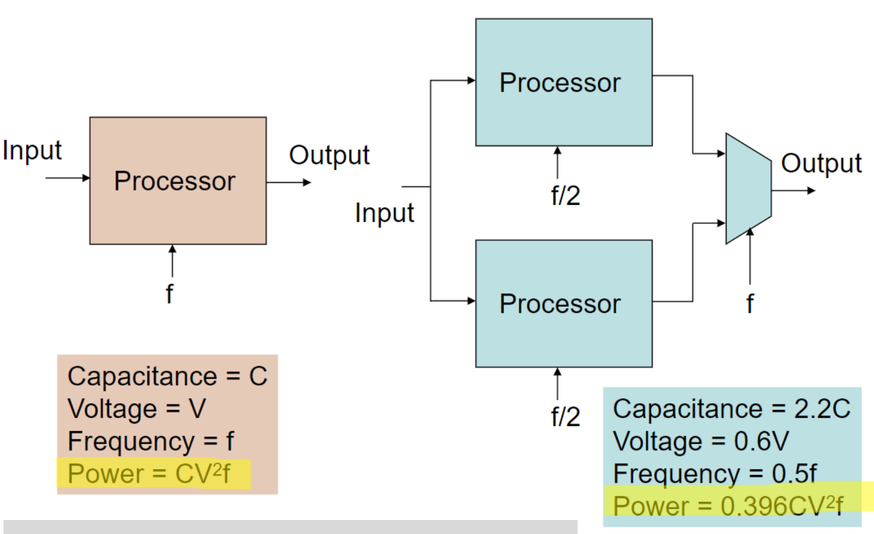
CLK: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

D Flip-Flop(130,150): 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

Non_pipelined_Out[7..0]: 0 3 5 1 2 4 -1 0

Pipelined_Out[7..0]: 3 5 1 2 4 -1 0

2.3.5 Pojav paralelizma



Approximate Trend

	n-parallel proc.	n-stage pipeline proc.
Capacitance	nC	C
Voltage	V/n	V/n
Frequency	f/n	f
Power	CV^2f/n^2	CV^2f/n^2
Chip area	n times	10-20% increase

G. K. Yeap, *Practical Low Power Digital VLSI Design*, Boston: Kluwer Academic Publishers, 1998.

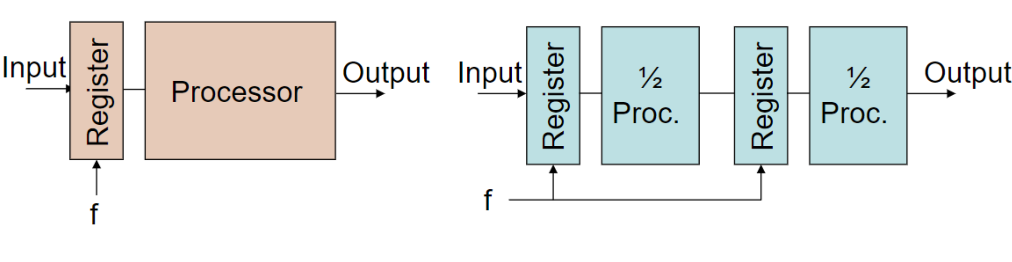
2/8/06 D&T Seminar 22

Poraba: *„… Dve jedri porabita manj kot eno dvakrat hitrejše …“*

Multi-Core Parallelism for Low-Power Design

Vishwani D. Agrawal
James J. Danaher Professor
Department of Electrical and Computer Engineering
Auburn University
http://www.eng.auburn.edu/~vagrawal
vagrawal@eng.auburn.edu

https://slideplayer.com/slide/5164867/

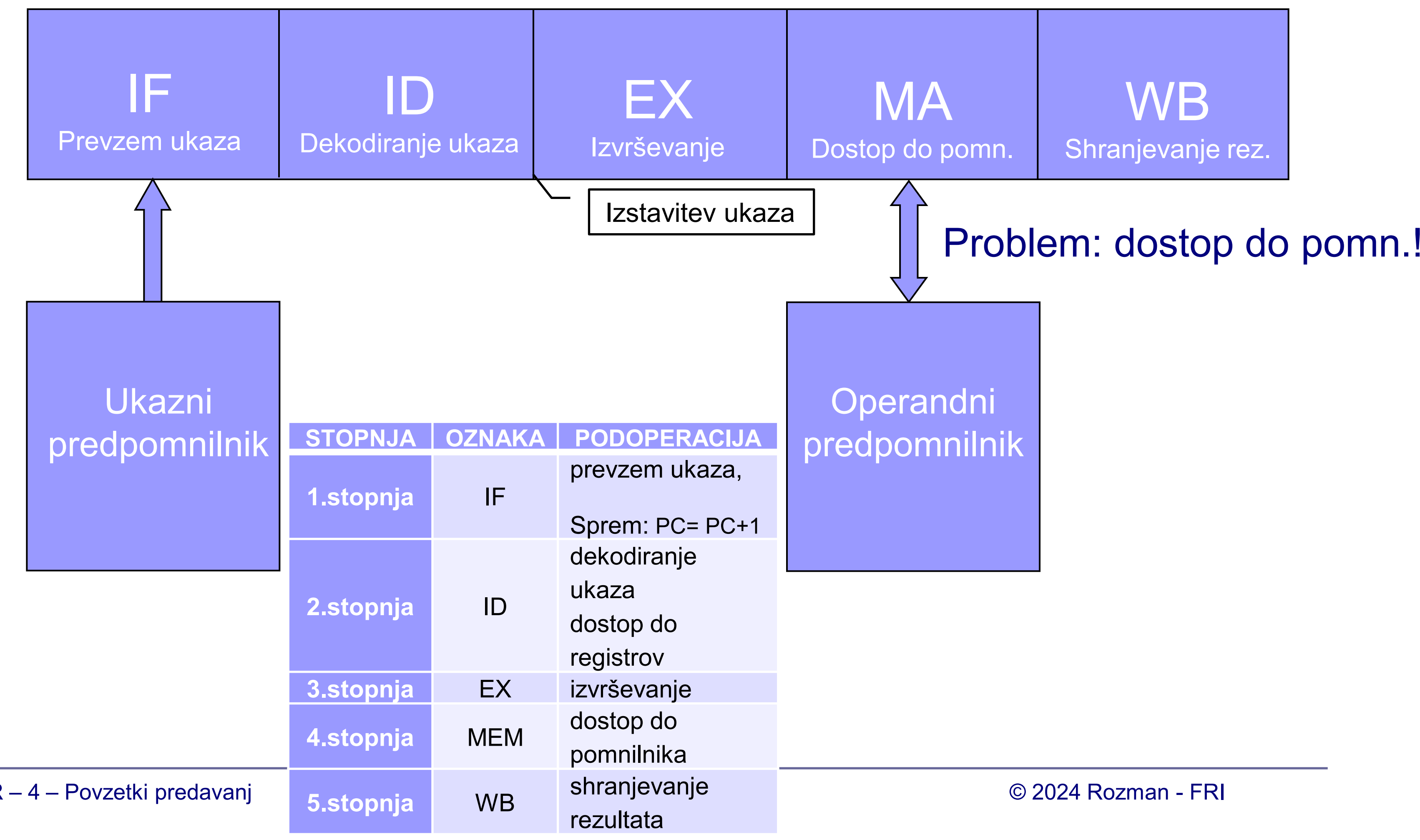


Capacitance = C
Voltage = V
Frequency = f
Power = CV^2f

Capacitance = 1.2C
Voltage = 0.6V
Frequency = f
Power = $0.432CV^2f$

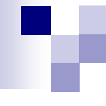
4.1 Zgradba cevovodne CPE

Primer 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

© 2024 Rozman - FRI



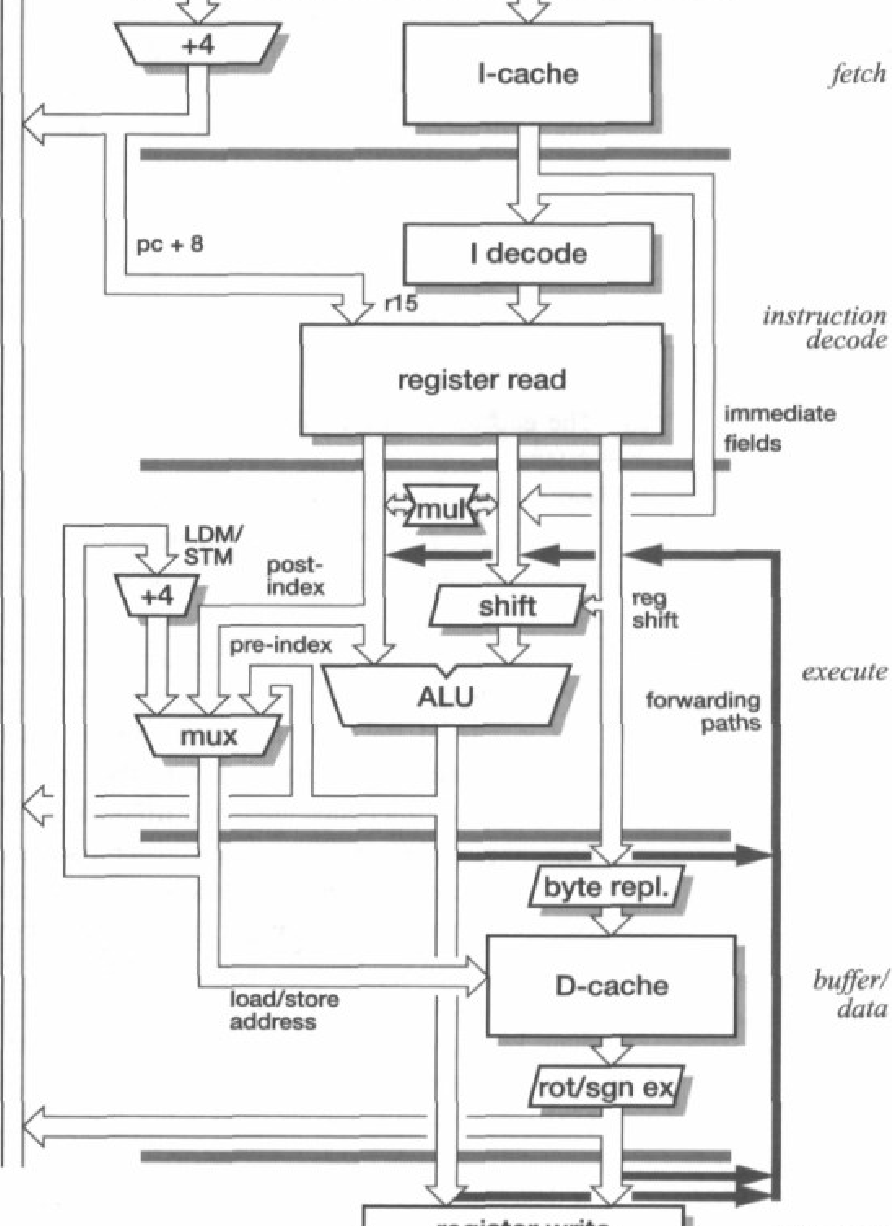
4.1 Zgradba cevovodne CPE

Primer cevovoda s 5 stopnjami (HiP):

BNE R4,LOOP --- --- --- ---

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

PREFETCH UNIT
DATA PROCESSING UNIT (+ FPU)
LOAD/STORE UNIT

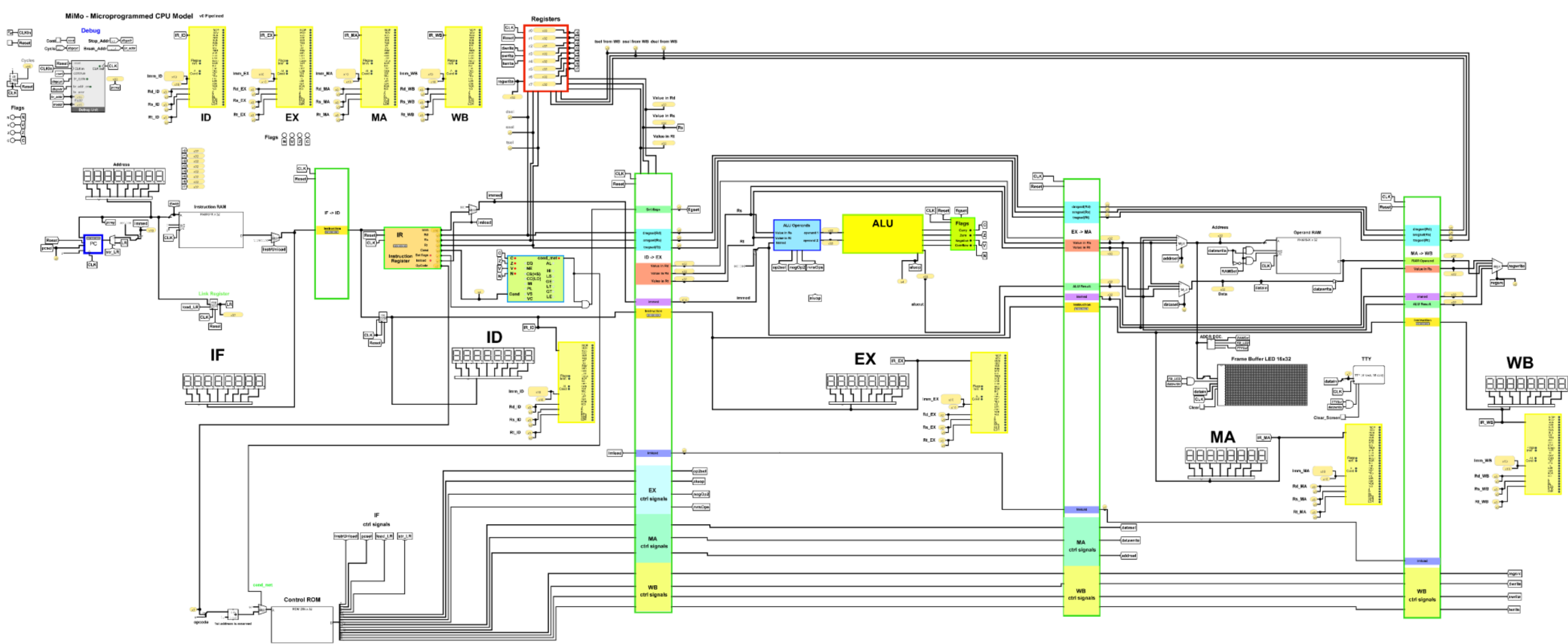
Update from DPU
Prefetch
BTAC
4 3 2 1
32-bit
32-bit
Fetch
Decode
Issue
#1 DECODE
#2 DECODE
from NVIC
64-bit instruction per cycle
Code memories

Execute
Load/Store 1
(2x 32b)
Load/Store 2
ALU 1 (Main)
ALU 2
MAC
(32b x 32b + 64b)
Branch
FPU

© 2024 Rozman - FRI

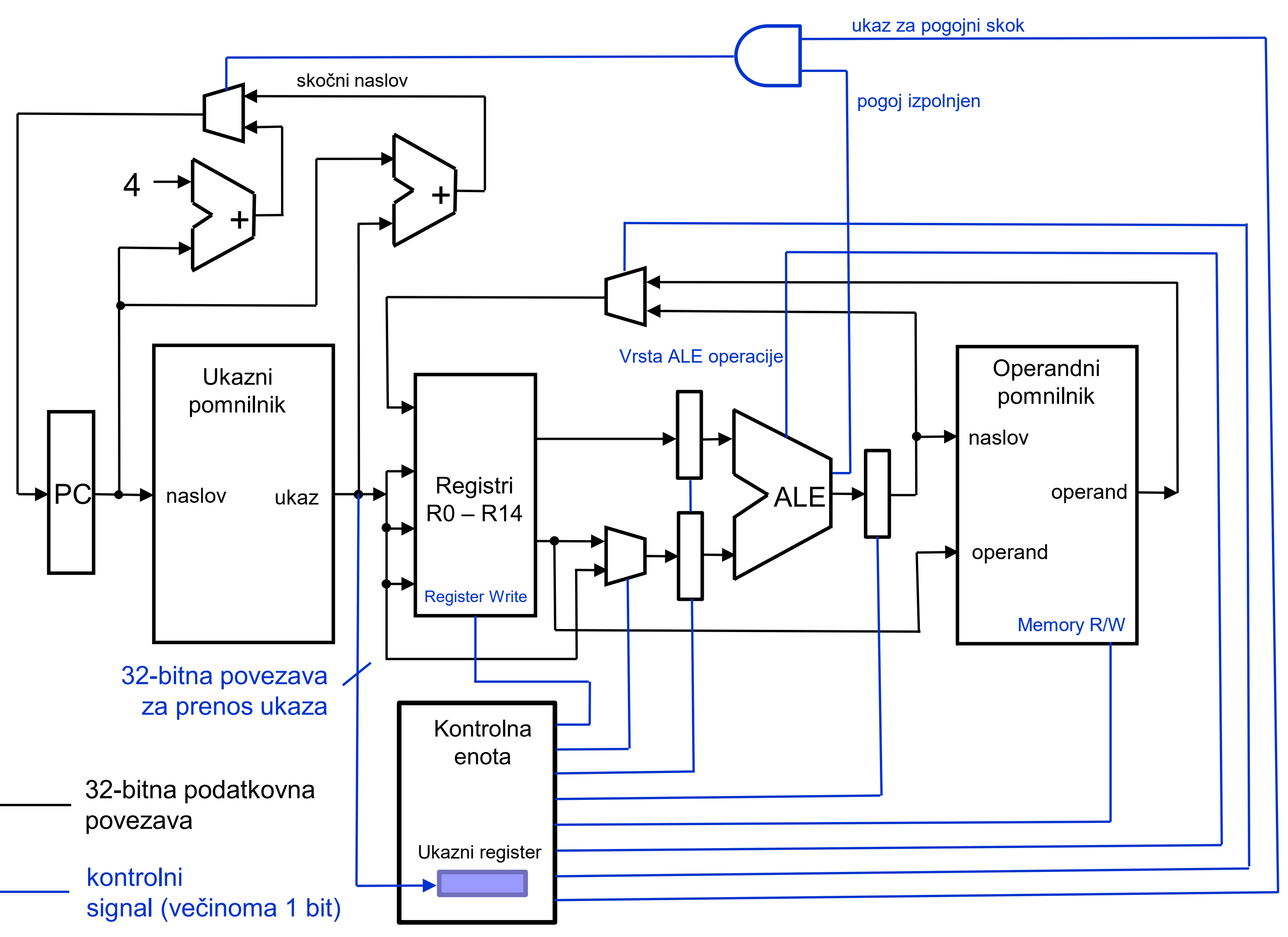
4.1 Zgradba cevovodne CPE

Primer 5 – stopenjskega cevovoda: MiMo v2:



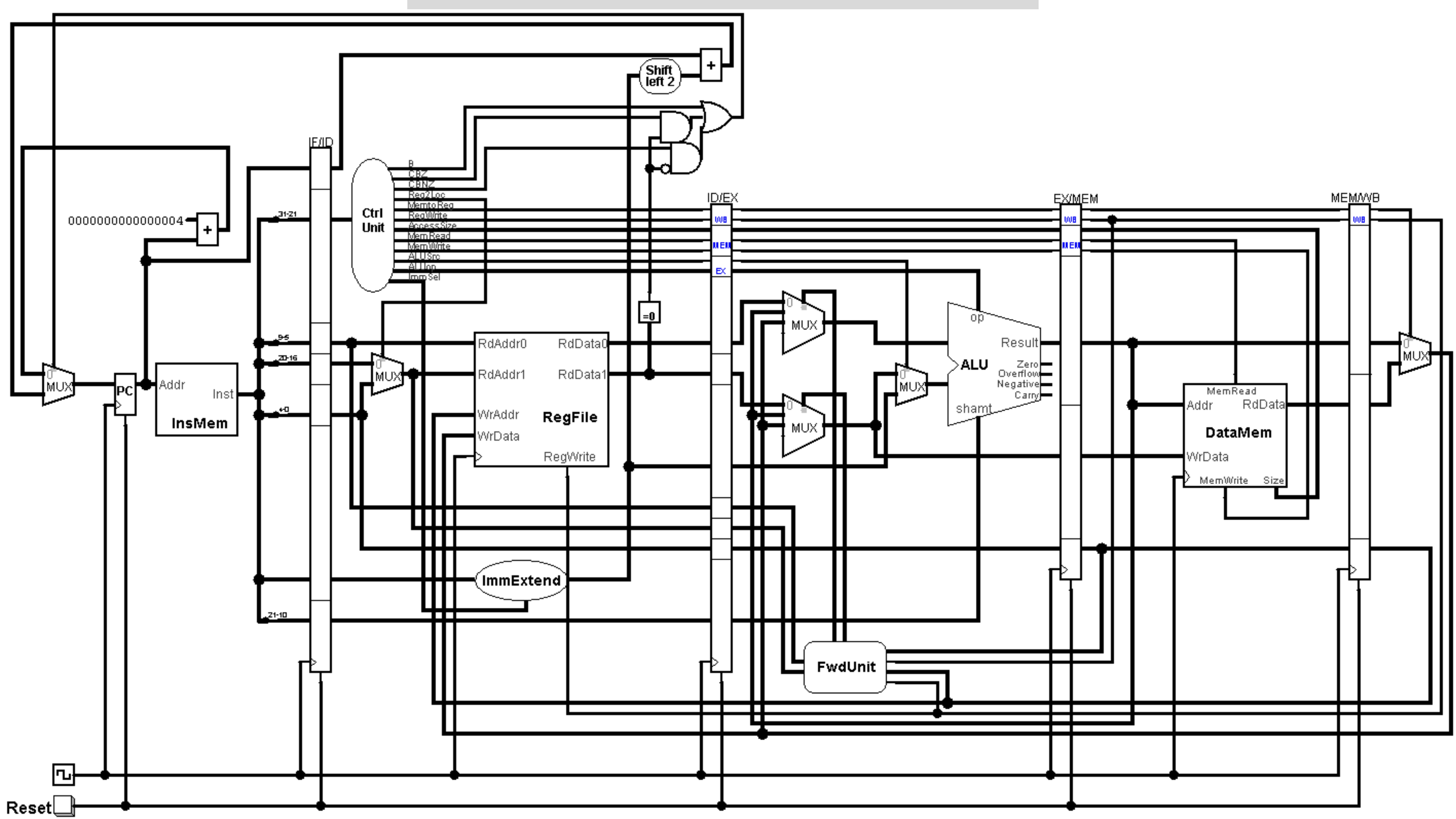
© 2024 Rozman - FRI

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

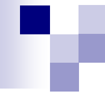


Poenostavljena različica ARM LEGv8 (Vir: [Patt], pogl. 4)

ARM V8 v Logisimu



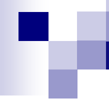
https://github.com/mkayaalp/computer-organization-logisim



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

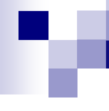
Problem dostopa do pomnilnika:

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

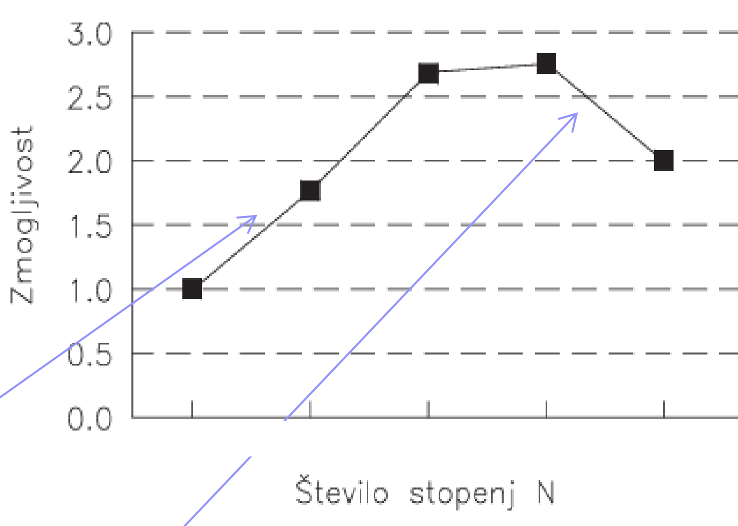
Problemi pri izvajanju ukazov

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

© 2024 Rozman - FRI



4.2 Cevovodne nevarnosti



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

Glede na vzrok ločimo:

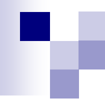
- **strukturne**
- **podatkovne (operandne)**
- **kontrolne**

S povečevanjem števila stopenj

- se povečuje število ukazov, ki se hkrati izvršujejo (manjšata se **CPI** in $\mathbf{t_{CPE}}$)
- 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

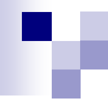
Odpravimo :

- **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“

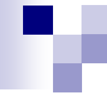
Imamo zaporedje dveh ukazov:

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

3 skupine podatkovnih nevarnosti :

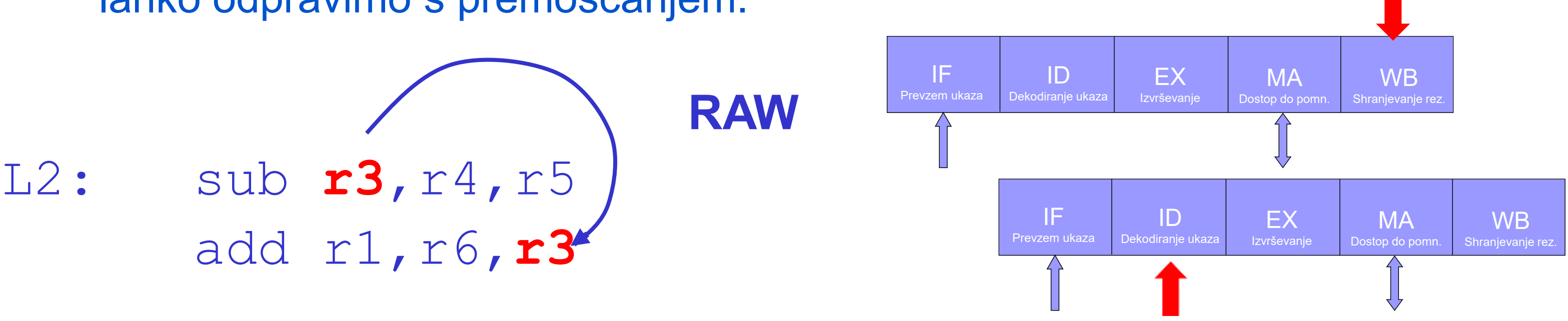
- **RAW – Read After Write**
- **WAR – Write After Read**
- **WAW – Write After Write**
- **~~RAR~~ ?**

 © 2024 Rozman - FRI



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.



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

Rešitve :

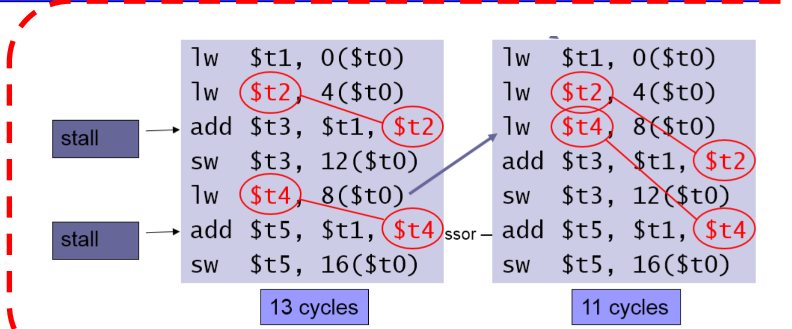
- *cevovodno razvrščanje (prevajalnik)*
- *zaklenitev cevovoda*
- *premoščanje*

© 2024 Rozman - FRI

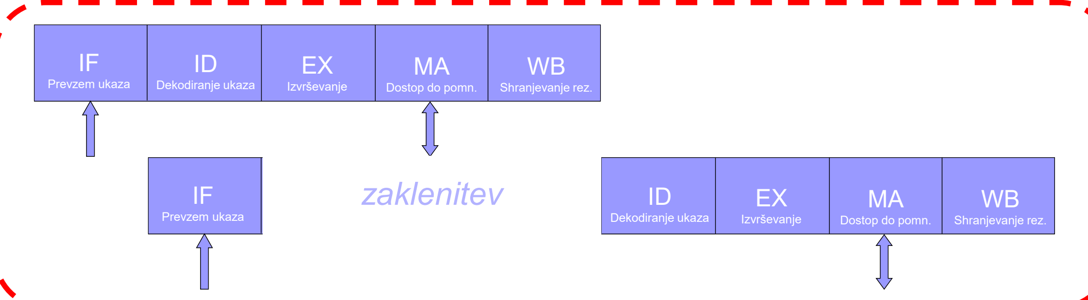
RAW (read after write) :

Rešitve :

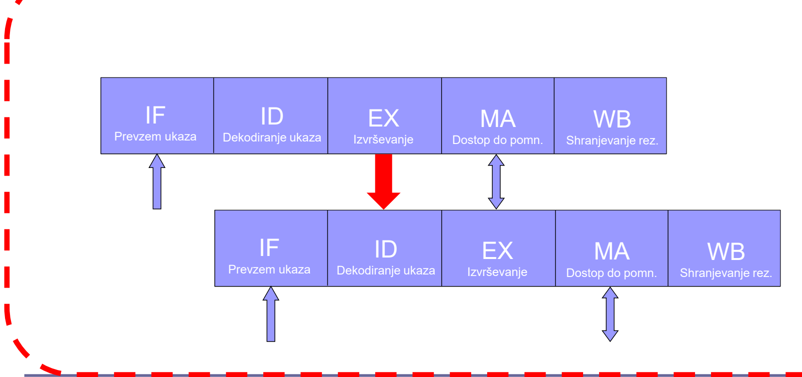
- *cevovodno razvrščanje (prevajalnik)*

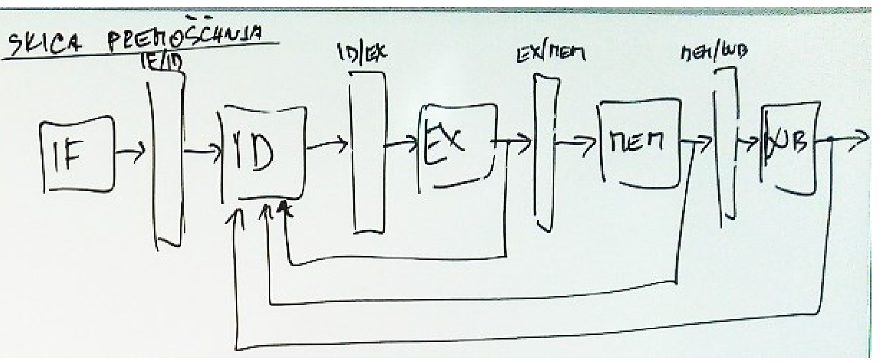


- *zaklenitev cevovoda*

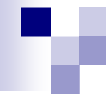


- *premoščanje*





© 2024 Rozman - FRI



RAW (read after write) : *Primer MiMo-Pipeline (v2) :*

Brez detekcije vs. zaklepanje

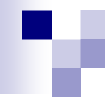
Ni detekcije nevarnosti

```
mov r1, #3  @cycle 5
mov r2, #3  @cycle 6
nop         @cycle 7
nop         @cycle 8
nop         @cycle 9
cmp r1, r2  @cycle 10
nop         @cycle 11
streq r1, #3@cycle 12
```

Zaklepanje

```
mov r1, #3  @cycle 5
mov r2, #3  @cycle 6
cmp r1, r2  @cycle 10
streq r1, #3@cycle 12
```

 © 2024 Rozman - FRI



RAW (read after write) : *Primer MiMo-Pipeline (v2) :*

zaklepanje vs. premoščanje

Zaklepanje

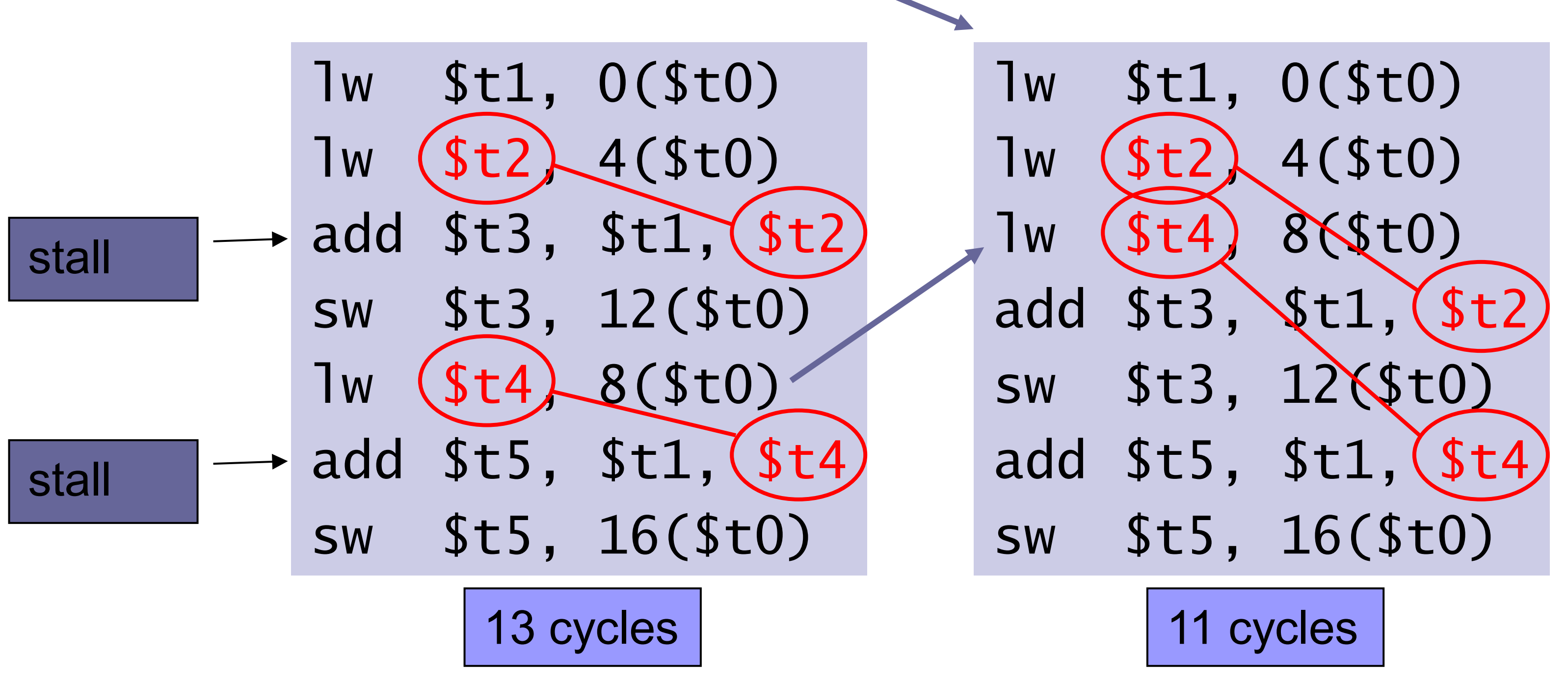
```
loop:                       @ stall
        mov r3, #3          @ 5, 22
        ldr r1, [r2]        @ 6, 23
        add r1, r1, #1      @ 10, 27
        add r7, r7, #1      @ 11, 28
        str r2, r1          @ 14
        r4, r3, r1          @ 15, 32
        add r5, r5, #1      @ 17, 34
        add r7, r7, #1      @ 18, 35
        add r6, r1, r4      @ 19, 36
        jne loop            @ 20, 37
```

Premoščanje

```
| forwarding
| 5, 17
| 6, 18
| 8, 20 (here one mandatory stall-get the value from MA)
| 9, 21 but left pipeline on 14), 31
| 10, 22
| 11, 23
| 12, 24
| 13, 25
| 14, 26
| 15, 27
```

Cevovodno razvrščanje

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



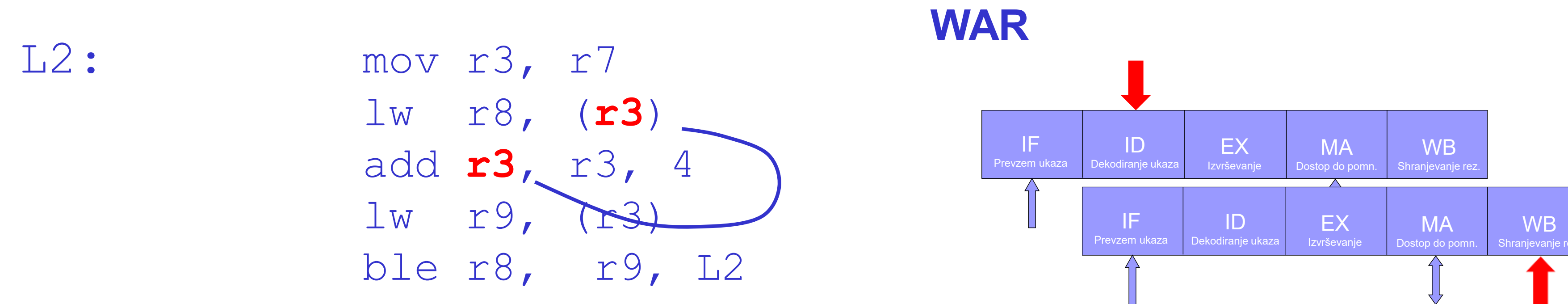
Potrebno znanje o delovanju konkretnega cevovoda !

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

© 2024 Rozman - FRI

WAR (write after read):

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

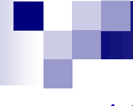


WAR (write after read) Ukaz *add* piše v register r3, preden ukaz *lw* prebere vsebino registra. Ukaz *lw* dobi tako novo vrednost namesto stare

Nastopa le :

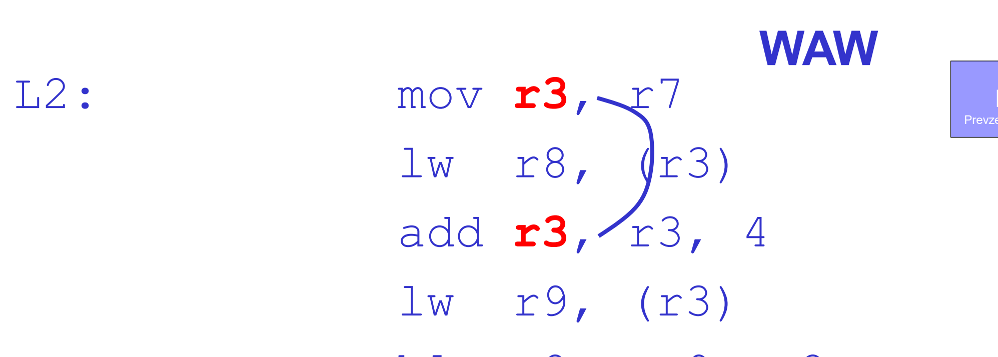
- v nekaterih izvedbah cevovodov in
- pri dinamičnem razvrščanjem ukazov

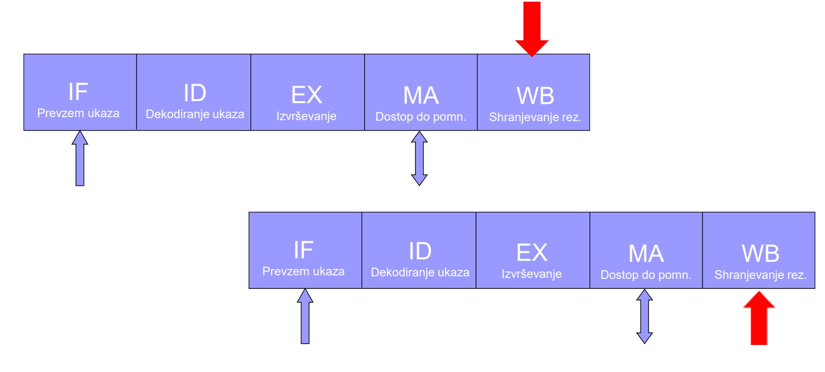
 © 2024 Rozman - FRI



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*.





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 razvrščanjem ukazov

 © 2024 Rozman - FRI

Analiza vpliva podatkovnih nevarnosti

MERITVE: 10 PROGR. (SPEC 92)

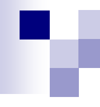
- PREDP.: • PREMOŠČANJE → ODPRAVI PODATK. NEV. ZAP. UKAZOV → RAZEN LOAD/STORE
  - • CEVOVODNO RAZVRŠČANJE
- → 24 % VSEH UKAZOV "LOAD"
  - → 19 % PRIDE DO NEVARNOSTI

ŠT. UKAZOV Z NEV. = 0'24 · 0'19 = 0'0456

$CPI_R = (1 - 0'0456) \cdot 1 + 0'0456 \cdot 2 = 1'0456$

→ SE POVEČA ZA 4'56 % (5 ST. CEV.)

© 2024 Rozman - FRI



4.2.3 Kontrolne nevarnosti

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

To se dogaja pri kontrolnih ukazih:

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

Slednji so **ugodnejši**, ker prej zvemo novi naslov.
Pri **pogojnih zvemo 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%!

© 2024 Rozman - FRI

4.2.3 Kontrolne nevarnosti

Izvajanje ukaza za pogojni skok

ukaz 1 - pogojni skok na ukaz 5

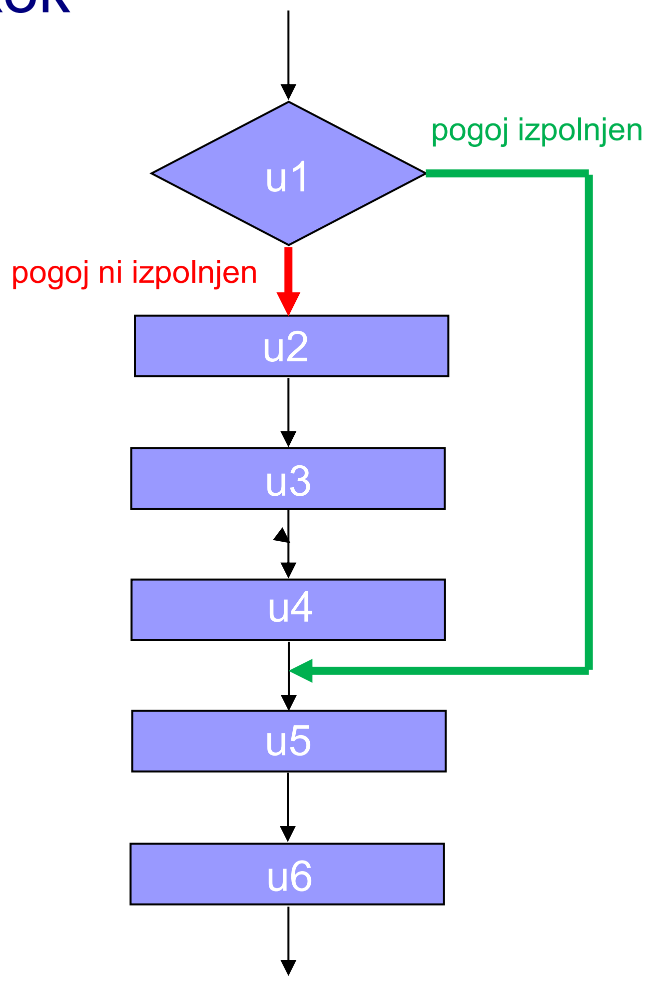
ukaz 2

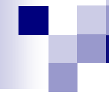
ukaz 3

ukaz 4

ukaz 5

ukaz 6





4.2.3 Kontrolne nevarnosti

Pogoj ni izpolnjen

ukaz 1 - pogojni skok na ukaz 5

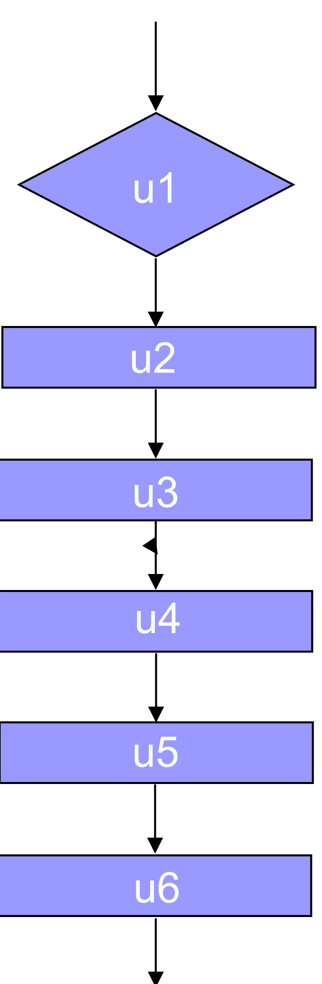
ukaz 2

ukaz 3

ukaz 4

ukaz 5

ukaz 6



4.2.3 Kontrolne nevarnosti

Pogoj je izpolnjen

ukaz 1 - pogojni skok na ukaz 5

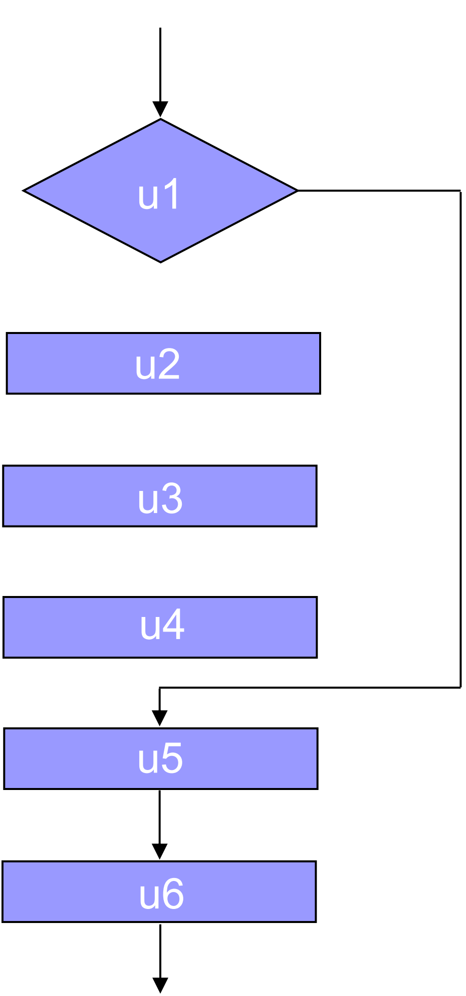
ukaz 2

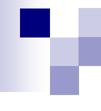
ukaz 3

ukaz 4

ukaz 5

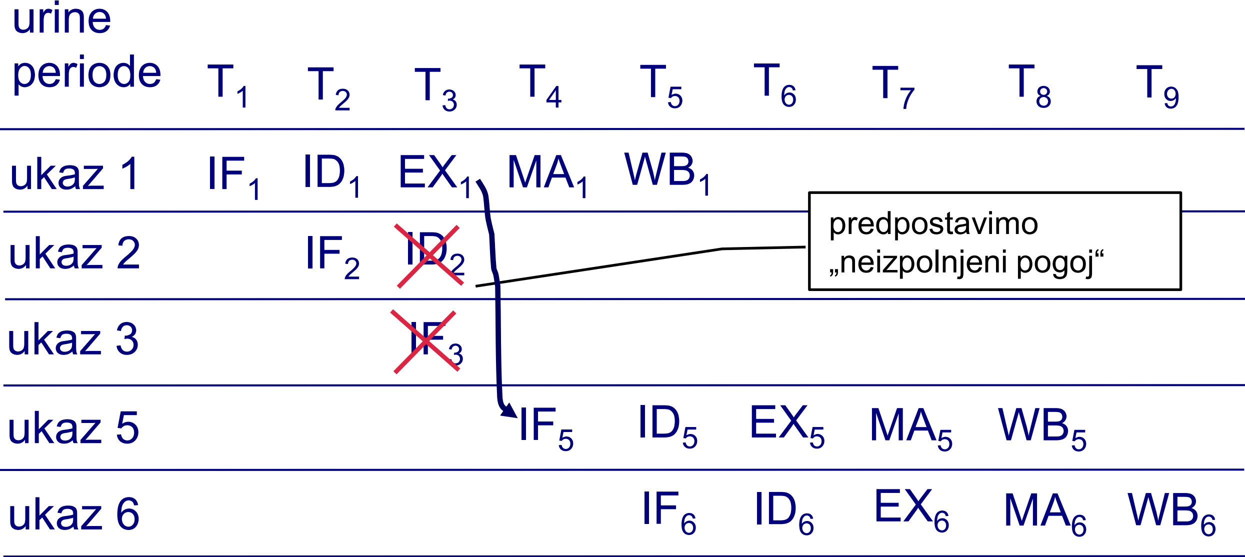
ukaz 6





- 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 :



© 2024 Rozman - FRI

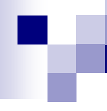
4.2.3.1.1 Statična predikcija neizpolnjenega pogoja („branch not-taken“)

-
  Primer ARM 9:
  - Statična napoved neizpolnjenega pogoja
  - Bxx (npr. BNE):
    - Pogoj xx neizpolnjen :
      - ?? t_{cpe}
    - Pogoj 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	~~ID_2~~						
ukaz 3			~~IF_3~~						
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“

© 2024 Rozman - FRI



4.2.3.1 Zmanjšanje zakasnitev pri kontrolnih nevarnosti

Pri gradnji cevovoda naredimo, da:

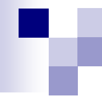
- se preverjanje pogoja za skok izvaja čim bolj na začetku cevovoda, saj je tako manj ukazov, ki bi jih morali zavreči
- se tudi izračun skočnega naslova izvaja čim bolj na začetku cevovoda

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 zakasnjenimi skoki („delayed branch“)
- dinamična predikcija (med delovanjem)



4.2.3.1.1 Statična predikcija z zakasnjenimi 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)

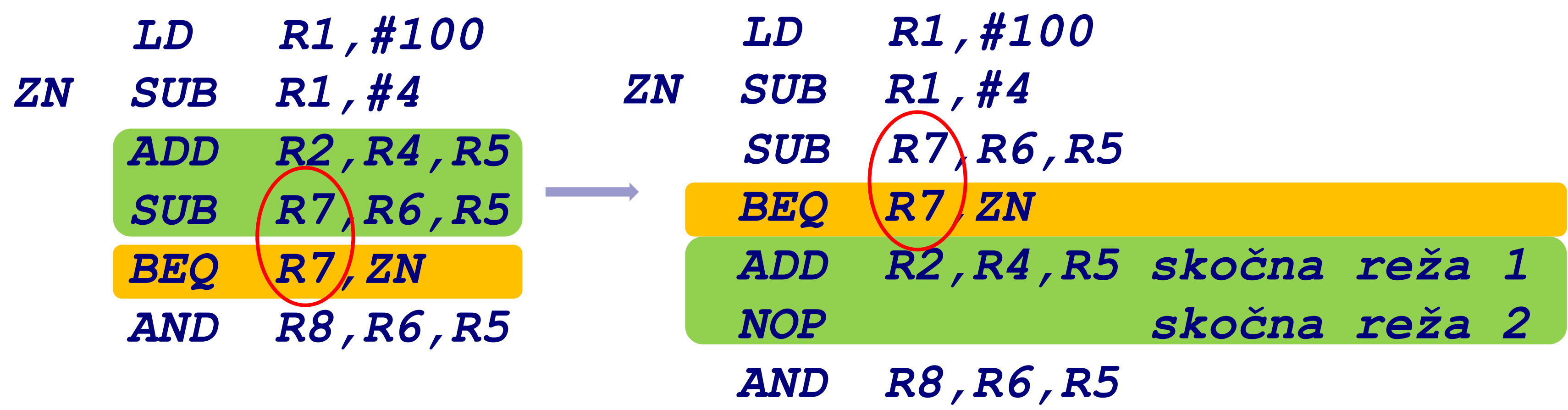
Vstavitev ukazov v skočne reže:

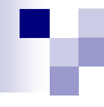
- ukazi ne smejo vplivati na izid skoka
- če ni primernih, vstavi NOP-e

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

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

- 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, ...





4.2.3.1.1 Statična predikcija z zakasnjenimi 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%!
- HiP: ob uporabi statične predikcije (zak.skoki) se CPI poveča le še za 8.7%!

 © 2024 Rozman - FRI

Primer pohitritve kode z uporabo cevovodnega razvrščanja, zakasnjenih 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	;delayed branch
SD	0(R1), F4	6	SD	8(R1), F4	6	;altered and interchanged with SUBI
SUBI	R1, R1,#8	7				
BENZ	R1, Loop	8				
stall		9				
9 clock cycles per element			**6 clock cycles per element**			

Vir: https://web.archive.org/web/20201019124610/http://web.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/loopUnrolling.html

© 2024 Rozman - FRI

- Vir: https://web.archive.org/web/20201019124610/http://web.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/loopUnrolling.html
  - Primer pohitritve kode z uporabo cevovodnega razvrščanja, zakasnjenih 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

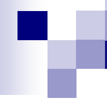
Loop:	LD	F0, **0**(R1)	1	
	stall		2	
	ADDD	F4, F0, F2	3	
	stall		4	
	stall		5	
	SD	**0**(R1), F4	6	;drop SUBI &BNEZ
	LD	F6, -**8**(R1)	7	
	stall		8	
	ADDD	F8, F6, F2	9	
	stall		10	
	stall		11	
	SD	-**8**(R1), F8	12	;drop SUBI &BNEZ
	LD	F10,-**16**(R1)	13	
	stall		14	
	ADDD	F12,F10,F2	15	
	stall		16	
	stall		17	
	SD	-**16**(R1), F12	18	;drop SUBI &BNEZ
	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**

Scheduled

Loop:	LD	F0, **0**(R1)	1	
	LD	F6, -**8**(R1)	2	
	LD	F10,-**16**(R1)	3	
	LD	F14,-**24**(R1)	4	
	ADDD	F4, F0, F2	5	
	ADDD	F8, F6, F2	6	
	ADDD	F8, F6, F2	7	
	ADDD	F16, F14, F2	8	
	SD	**0**(R1), F4	9	
	SD	-**8**(R1), F8	10	
	SD	-**16**(R1), F12	11	
	SUBI	R1, R1, #32	12	
	BENZ	R1, Loop	13	
	SD	**8**(R1), F16	14	;8-32=-24

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



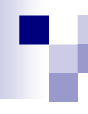
4.2.3.1.2 Dinamična predikcija skokov

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
```

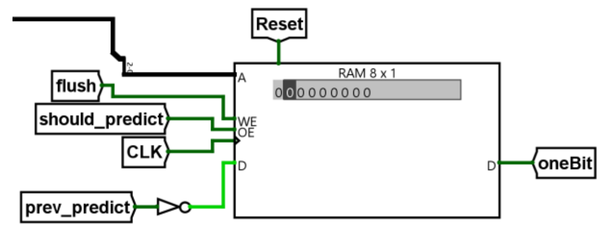
LSB	Predictor
⋮	⋮
111100	0
⋮	⋮

- 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

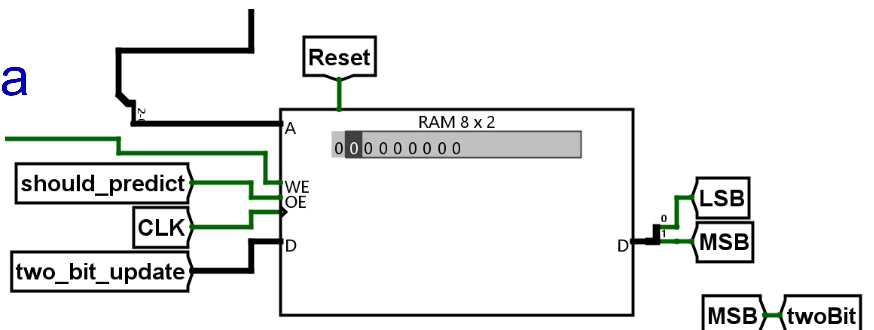
1 bit prediction table



Zanka:

- dve obvezni zgrešitvi: na začetku in koncu
- zato boljša 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:

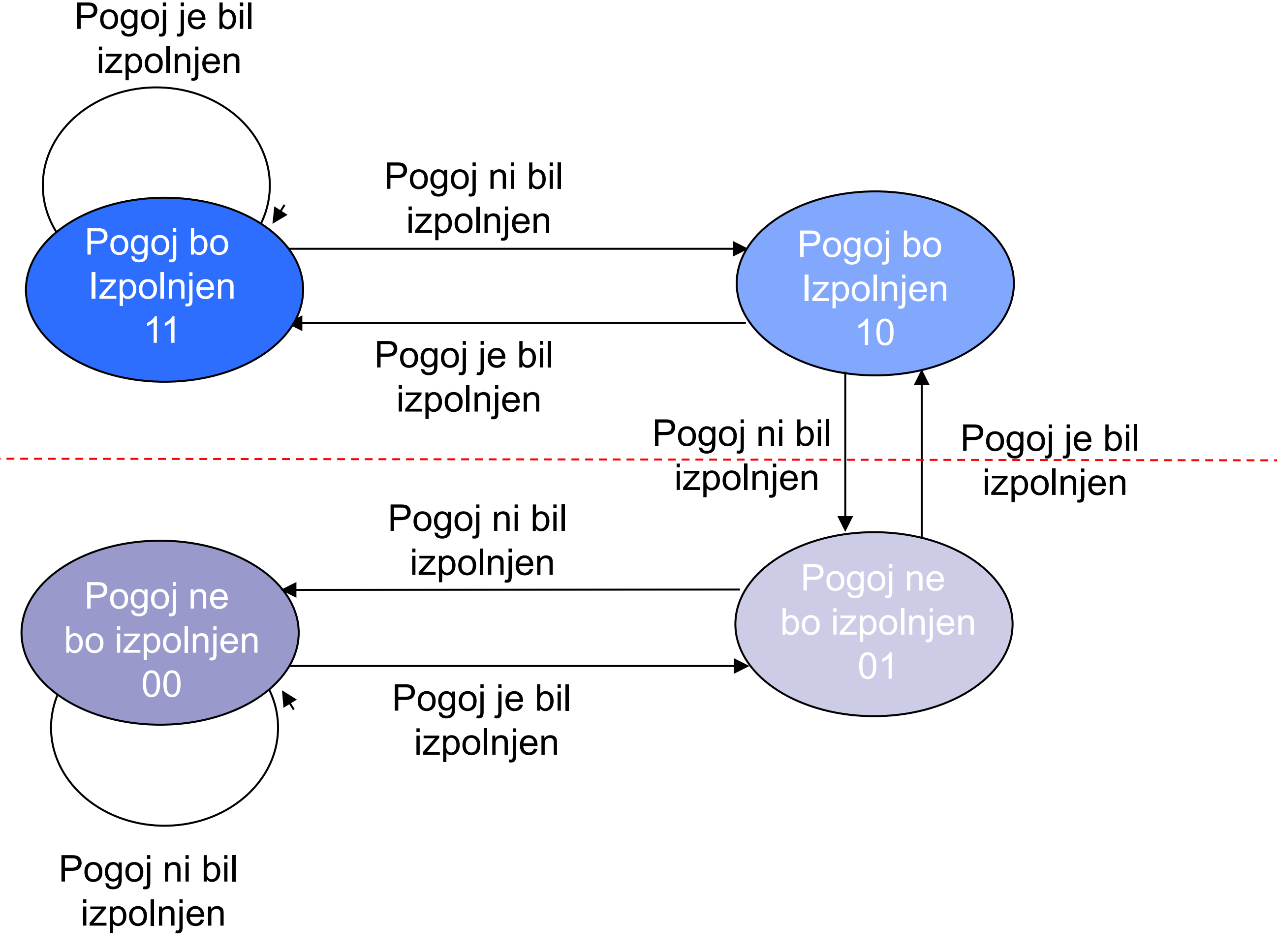
- prevzem ukaza v skladu s predikcijo
- če pogoj izpolnjen, se vrednost poveča za 1 (maks.vr. 3)
- če pogoj ni izpolnjen, se vrednost zmanjša za 1 (min.vr. 0)

Meritev na programih:

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

© 2024 Rozman - FRI

Prehajanje stanj pri 2-bitnem prediktorju



© 2024 Rozman - FRI

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 tabele 0,1,2,3
- glede na izpolnjenost pogoja zadnjih dveh skokov se uporabi različna napoved (ena izmed tabel 0,1,2,3).

spodnji naslovni biti skočnega ukaza

štiri 2–bitne prediktorske tabele

0 1 2 3

X X

XX (vrednost, ki se uporabi za napoved)

0 1

2–bitna globalna zgodovina

Delovanje:

- prevzem ukaza v skladu s predikcijo
- v izbrani 2-bitni tabeli:
  - če pogoj izpolnjen, se vrednost poveča za 1 (maks.vrednost 3)
  - če pogoj ni izpolnjen, se vrednost zmanjša za 1 (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

© 2024 Rozman - FRI

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

Primerjava na skupini programov

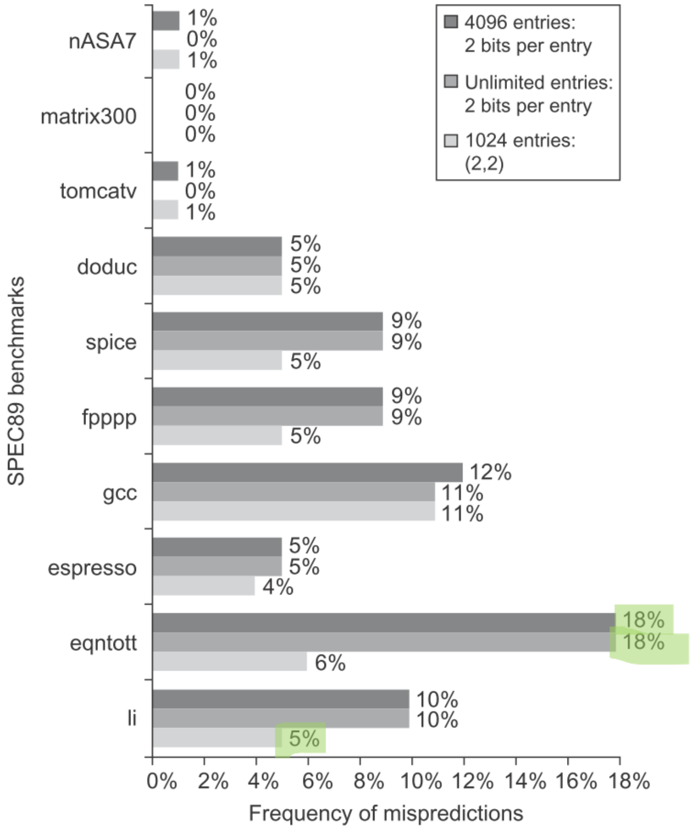
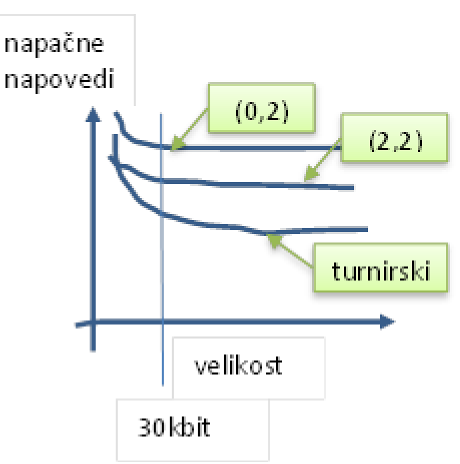


Figure 3.3 Comparison of 2-bit predictors. A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating 2-bit predictor with unlimited entries and a 2-bit predictor with 2 bits of global history and a total of 1024 entries. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks would show similar differences in accuracy.

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 – novejše nadgradnje ideje, z združevanjem napovedi različnih prediktorjev

Primerjava enostavnejših prediktorjev

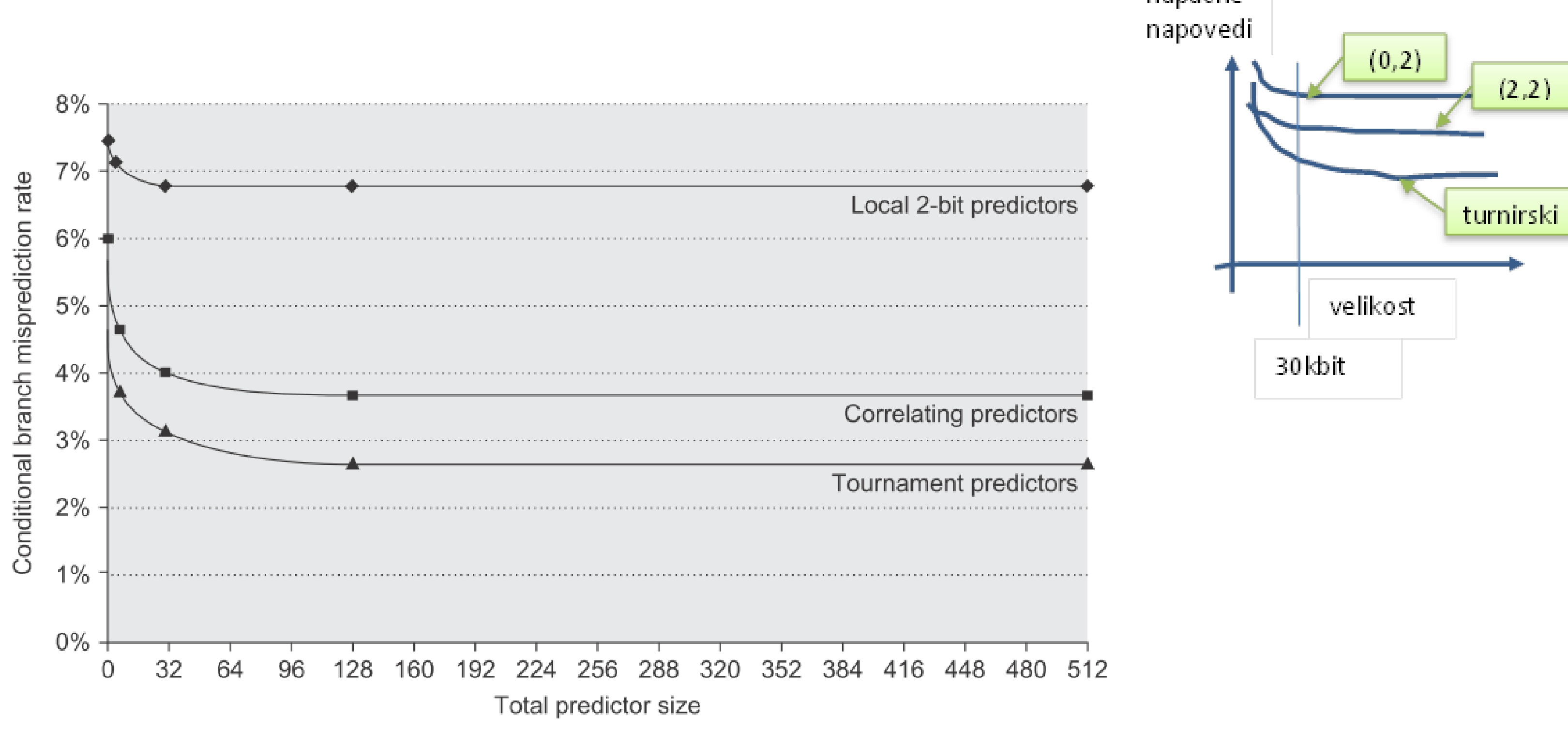
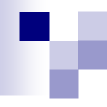


Figure 3.6 The misprediction rate for three different predictors on SPEC89 versus the size of the predictor in kilobits. The predictors are a local 2-bit predictor, a correlating predictor that is optimally structured in its use of global and local information at each point in the graph, and a tournament 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

Nevronske mreže (MLP)

- AMD Ryzen, Exynos

G-Share (korelacijski)

TAGE (hibridni)

4.2.3.1.2.5 Prediktorji – trenutno stanje

Aktualni:

- „gshare“ prediktor (korelacijski)
  - kombinira „lokalno“ in „globalno“

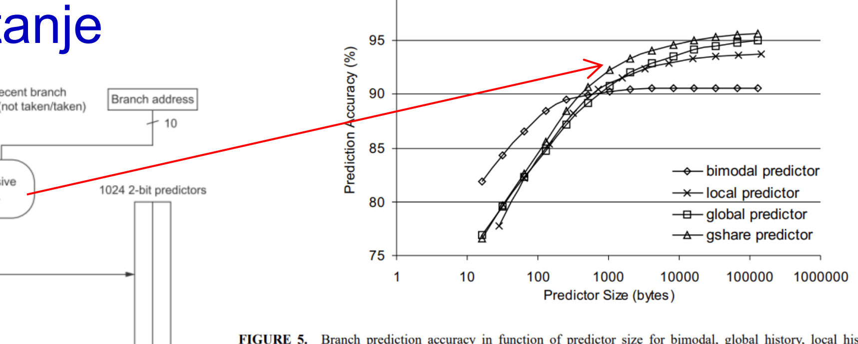


Figure 3.4 A gshare predictor with 1024 entries, each being a standard 2-bit predictor.

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

- TAGE prediktor (hibridni)
  - več dolžin zgodovine

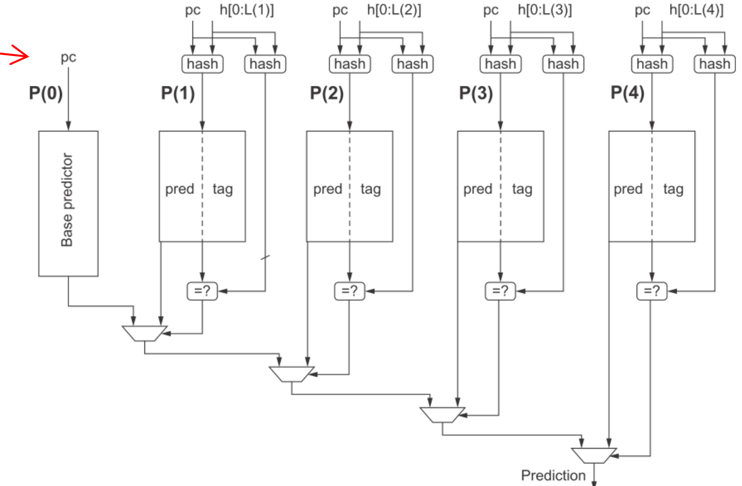
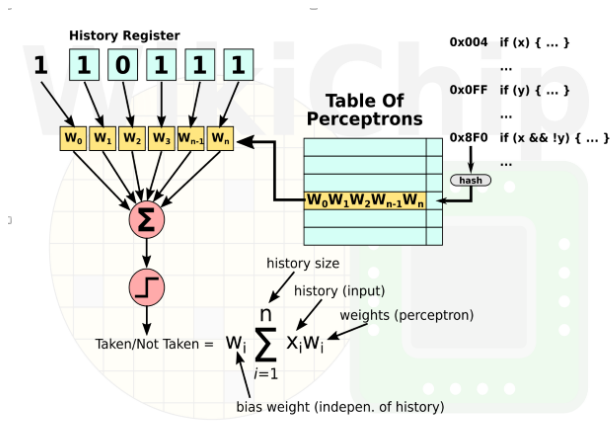


Figure 3.7 A five-component tagged hybrid predictor has five separate prediction tables, indexed by a hash of the branch address and a segment of recent branch history of length 0–4 labeled “h” in this figure. The hash can be as simple as an exclusive-OR, as in gshare. Each predictor is a 2-bit (or possibly 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 prediktorji ali druge hibridne kombinacije
- AMD: hibridni (Zen 2)
  - Perceptron (hiter) + TAGE (boljša napoved)
  - Perceptron hitro napove, TAGE kasneje preveri

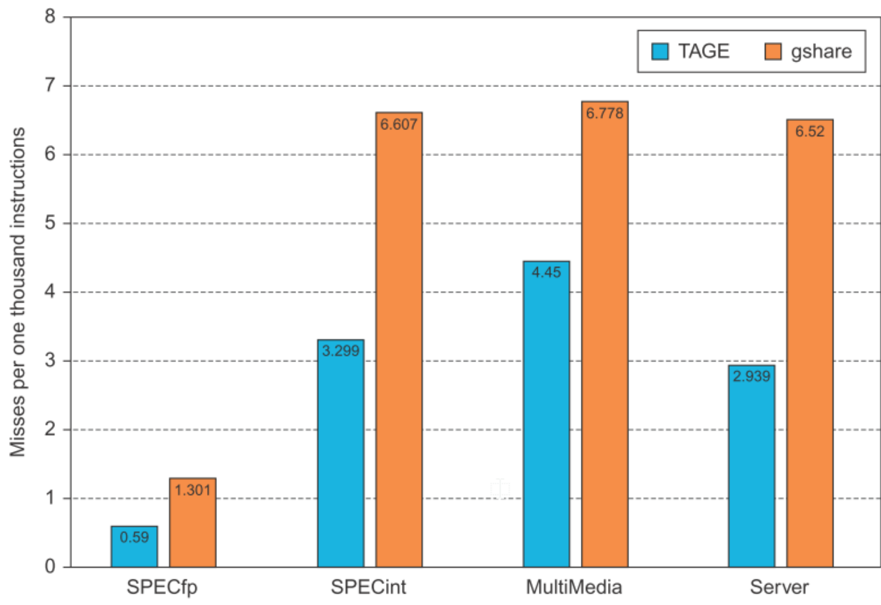


Figure 3.8 A comparison of the misprediction rate (measured as mispredicts per 1000 instructions executed) for tagged hybrid versus gshare. Both predictors use the same total number of bits, although tagged hybrid uses some of that storage for tags, while gshare contains no tags. The benchmarks consist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

© 2024 Rozman - FRI

4.2.3.1.2.5 Prediktorji – trenutno stanje

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.

The Zen 2 has three levels of branch target buffer (BTB). The three levels of BTB have 16, 512, and 7168 entries, respectively, with a latency of 0, 1, and 4 clock cycles, respectively.

The Zen 3 has a level-1 BTB with 1024 entries and a level-2 BTB with 6656 entries. It has an additional predictor for indirect jumps with 1536 entries. The Zen 4 has a level-1 BTB with 2*1536 entries and a level-2 BTB with 2*7168 entries.

The average misprediction penalty was measured to approximately 18 clock cycles for Zen 1-3. The misprediction penalty varies from 15 - 18 in Zen 4.

The return stack buffer has 32 entries.

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 - 2023. Last updated 2023-05-26.

3.8 Branch prediction in Intel Haswell, Broadwell, Skylake, and other Lakes

The branch predictor appears to have been redesigned in the Haswell and later Intel processors, but the design is undocumented.

Reverse engineering has revealed that the branch prediction is using several tables of local and global histories of taken branches [Yavarzadeh, 2023].

These observations may indicate that there are two branch prediction methods: a fast method tied to the µop cache and the instruction cache, and a slower method using a branch target buffer.

BTB organization

The organization of the branch target buffer is unknown. It appears to be reasonably big.

Misprediction penalty

The branch misprediction penalty varies a lot. It was measured to 15 - 20 clock cycles.

Prediction of function returns

The return stack buffer has 16 entries for near returns in Haswell, Broadwell, and Skylake, and 22 in Ice Lake and Tiger Lake.

4.2.3.1.2.5 Prediktorji – trenutno stanje

6.375 Complex Digital Systems

Francis Wang, Shana Mathew

Multi-tiered branch predictor for pipelined processors

Some modern processors employ a tiered strategy to branch prediction. There is a hierarchy of prediction structures where each predictor is slower, larger, and more accurate than the last. Closest to the fetch stage is the Branch Target Buffer (BTB). The BTB is very tightly coupled to the fetch stage and it predicts the next address before the instruction is even fetched from memory. When an instruction is identified as a branch, the first level branch predictor kicks in and redirects the instruction flow if it believes the BTB to be in error. As the instruction goes through the pipeline, more sophisticated branch predictors are consulted and the instruction flow is redirected as is appropriate. These branch predictors may take several cycles to compute their results but make use of longer history records or are specialized to recognize certain program behavior such as loops or function returns. The combination of small and fast predictors and large and slow predictors allow designers to strike a good compromise between latency and accuracy.

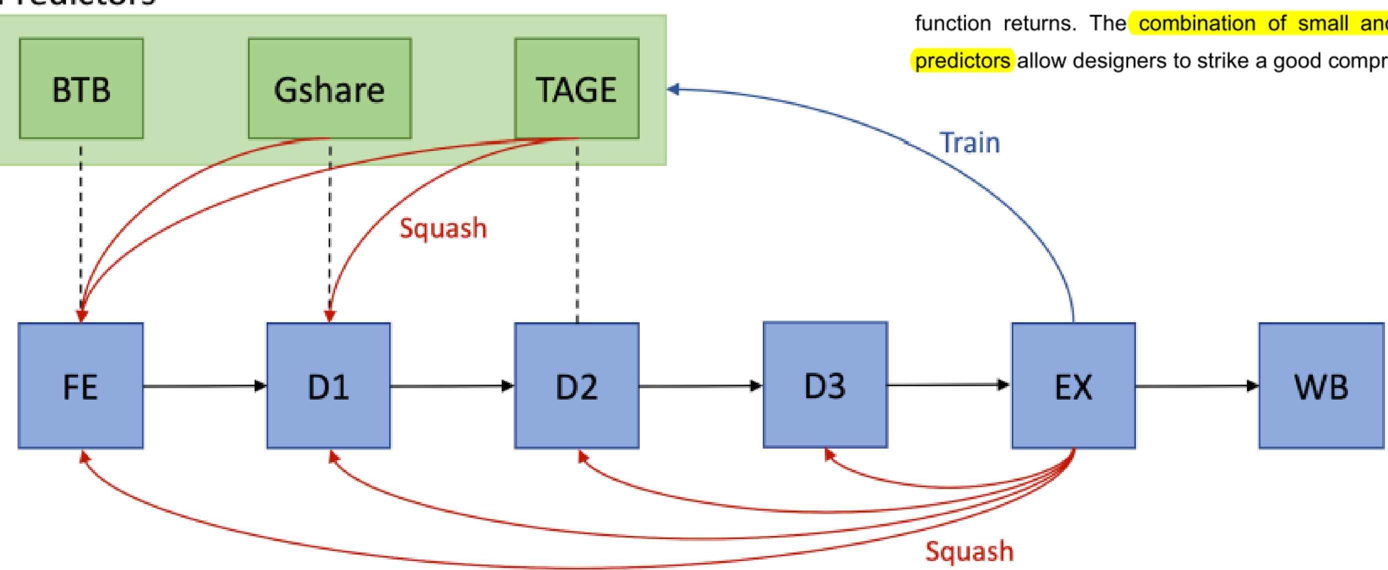


Figure 1. Pipeline diagram with predictors

4.2.3.1.2.5 Prediktorji – pogled naprej

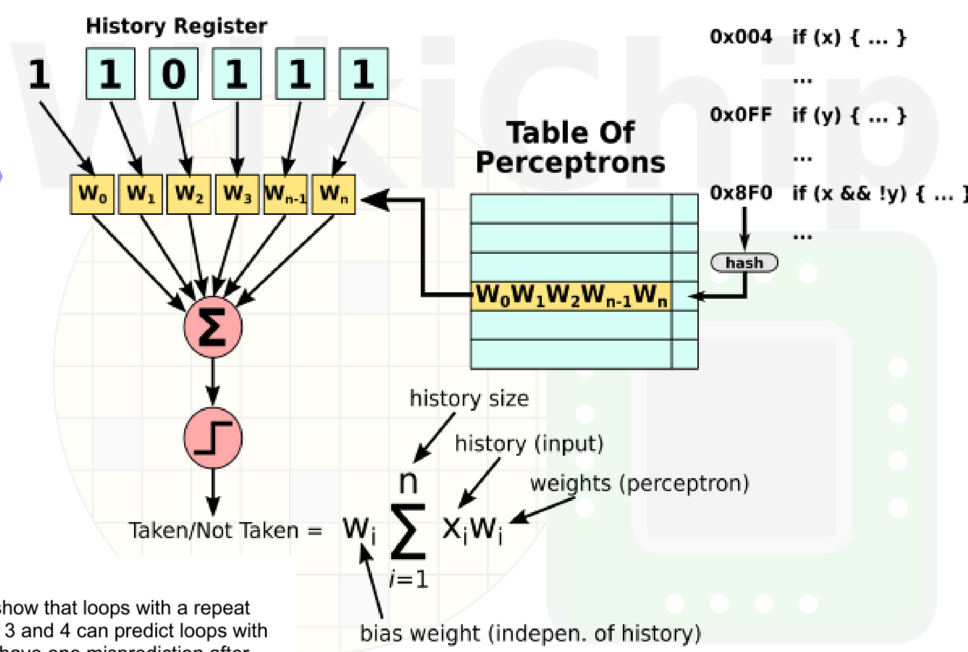
3.
The microarchitecture of Intel, AMD, and VIA CPUs
An optimization guide for assembly programmers and compiler makers

By Agner Fog. Technical University of Denmark.
Copyright © 1996 - 2023. Last updated 2023-05-26.

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

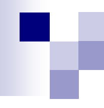
Potenciali:

- 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.



4.2.3.1.3 Skočni predpomnilnik

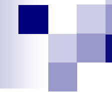
Za napoved skoka poleg napovedi pogoja rabimo še napoved skočnega naslova (sicer še vedno skočna zakasnitev):

- 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:

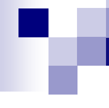
- prevzem ukaza in
- dostop do skočnega predpomnilnika:
    - če zadetek, se prebere skočni naslov in prenese v PC.
    - če zgrešitev, predpostavimo neizpolnjen pogoj

 © 2024 Rozman - FRI



4.2.3.1.3 Skočni predpomnilnik

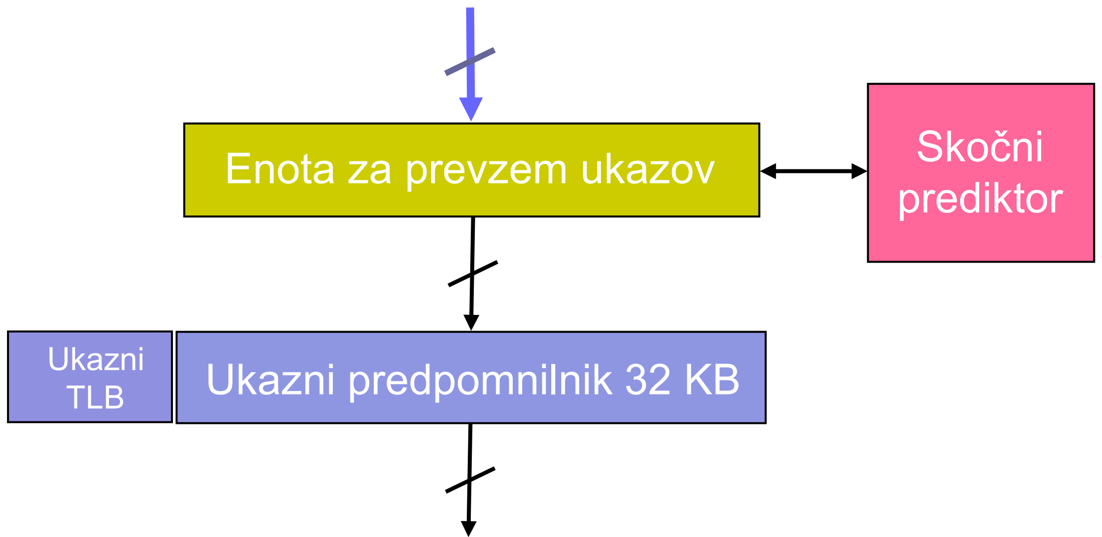
Skočni predpomnilnik



4.2.3.1.3 Skočni predpomnilnik

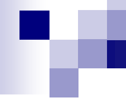
Če sta napoved izida skoka ali skočni naslov napačna, se mora ukaze, ki so bili v cevovodu, zavreči (zamenjati z mehurčki)

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

© 2024 Rozman - FRI



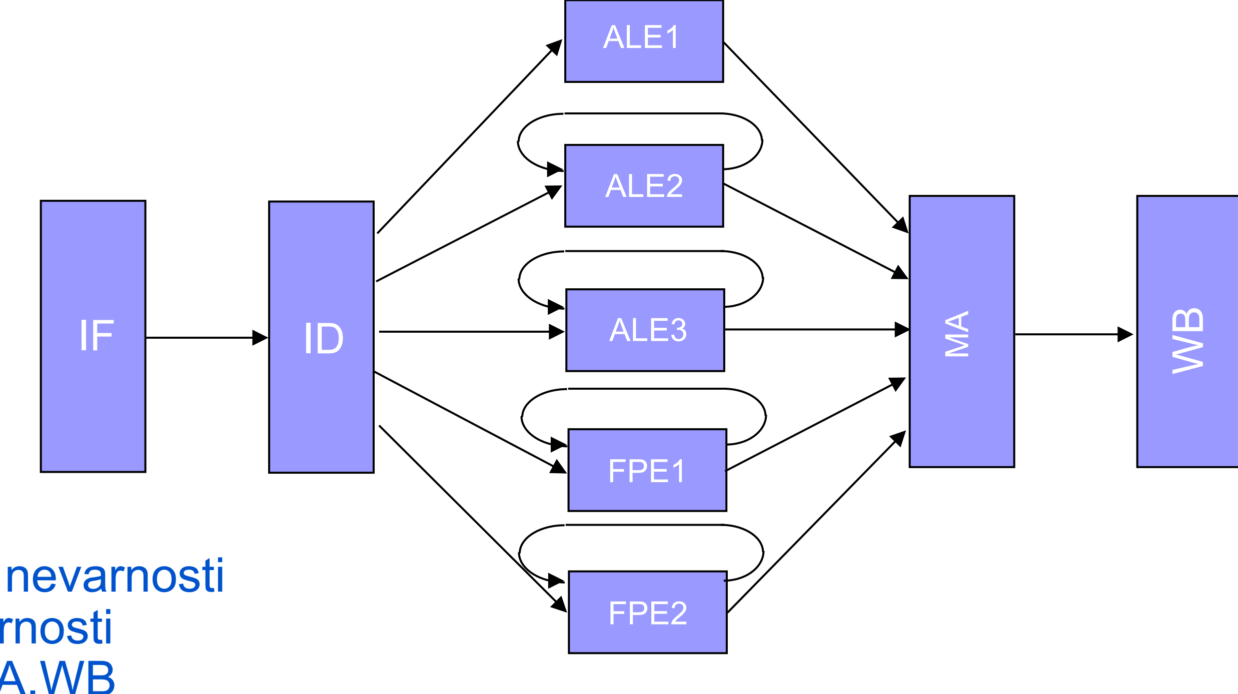
4.3 Izvajanje operacij, ki trajajo več urinih period

- 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)

© 2024 Rozman - FRI

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

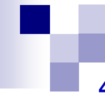
- Primer: Izvrševanje v prvi funkcijski enoti (ALE1) traja eno urino periodo, v vseh ostalih pa več urinih period.



Večperiodne FE:

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

© 2024 Rozman - FRI



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

IF → ID

A

A1 → A2

IM1 → IM2 → IM3 → IM4

FM1 → FM2 → FM3 → FM4 → FM5 → FM6

FD1 → FD2 → - - - - - → FD11 → FD12

→ MA → WB

Rešitev za boljšo izkoriščenost FE enot -> dinamično razvrščanje ukazov

 © 2024 Rozman - FRI

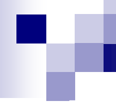
Problemi cevovodne realizacije FE:

- 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
FLD F4,0(R2)	IF	ID	EX	MEM	WB												
FMUL F0,F4,F6		IF	ID	○	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
FADD F2,F0,F8			IF	○	ID	○	○	○	○	○	○	A1	A2	A3	A4	MEM	WB
FST 0(R2),F2					IF	○	○	○	○	○	○	ID	EX	○	○	○	MEM

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

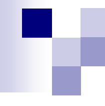


4.4 Odpravljanje podatkovnih nevarnosti z dinamičnim razvrščanjem ukazov

- 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:

```
IMUL    R2,R3,R4    R2←R3*R4
ADD     R6,R2,R5    R6←R2+R5   RAW
IDIV    R7,R8,R5    R7←R8/R5
```

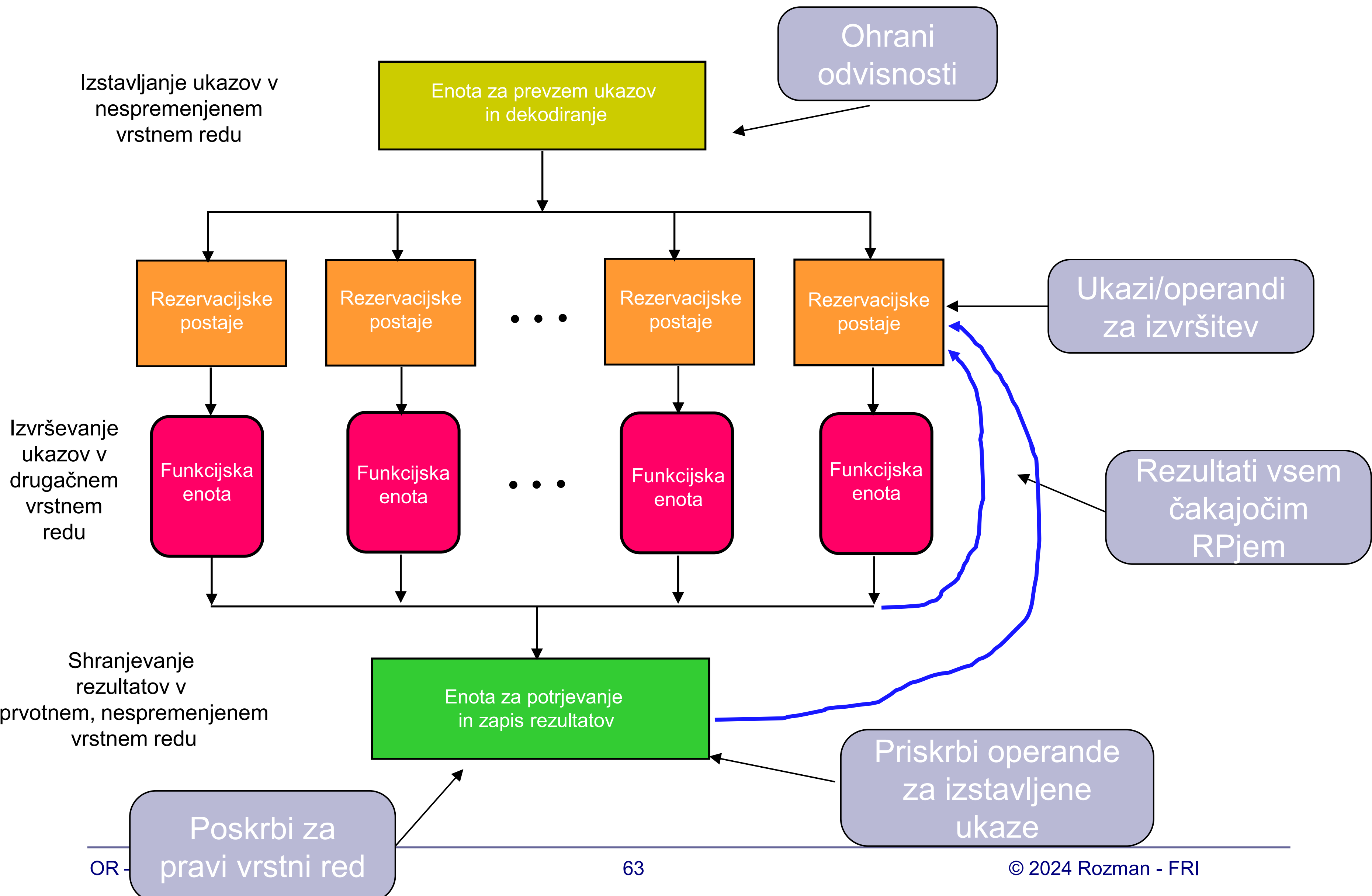


Realizacija:

- 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“)

© 2024 Rozman - FRI


Ohrani odvisnosti
Izstavljanje ukazov v nespremenjenem vrstnem redu
Enota za prevzem ukazov in dekodiranje
Rezervacijske postaje
Rezervacijske postaje
• • •
Rezervacijske postaje
Rezervacijske postaje
Ukazi/operandi za izvršitev
Izvrševanje ukazov v drugačnem vrstnem redu
Funkcijska enota
Funkcijska enota
• • •
Funkcijska enota
Funkcijska enota
Rezultati vsem čakajočim RPjem
Shranjevanje rezultatov v prvotnem, nespremenjenem vrstnem redu
Enota za potrjevanje in zapis rezultatov
Priskrbi operande za izstavljene ukaze
Poskrbi za pravi vrstni red

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

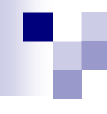
Name	Busy	Op	Vj	Vk	Qj	Qk	A	Result
Load0	true	L.D		Regs[R2]				
Load1	true	L.D		Regs[R3]				
Add0	true	ADD.D		Regs[F8]				
Add1	true	SUB.D			Load1	Load0		
Add2	true	ADD.D			Add1	Load1		
Mult0	true	MUL.D		Regs[F4]	Load1			
Mult1	true	DIV.D			Mult0	Load0		
Store0	true	S.D			Add2		Regs[R2] + 32	
Store1	false							

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/

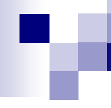


Slabosti dinamičnega razvrščanja ukazov:

- 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….

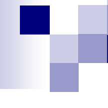


4.5 Špekulativno izvrševanje ukazov

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.

Primera:

- 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).

© 2024 Rozman - FRI

Procesor s špekulativnim izvrševanjem ukazov običajno izvaja kombinacijo omenjenih pristopov:

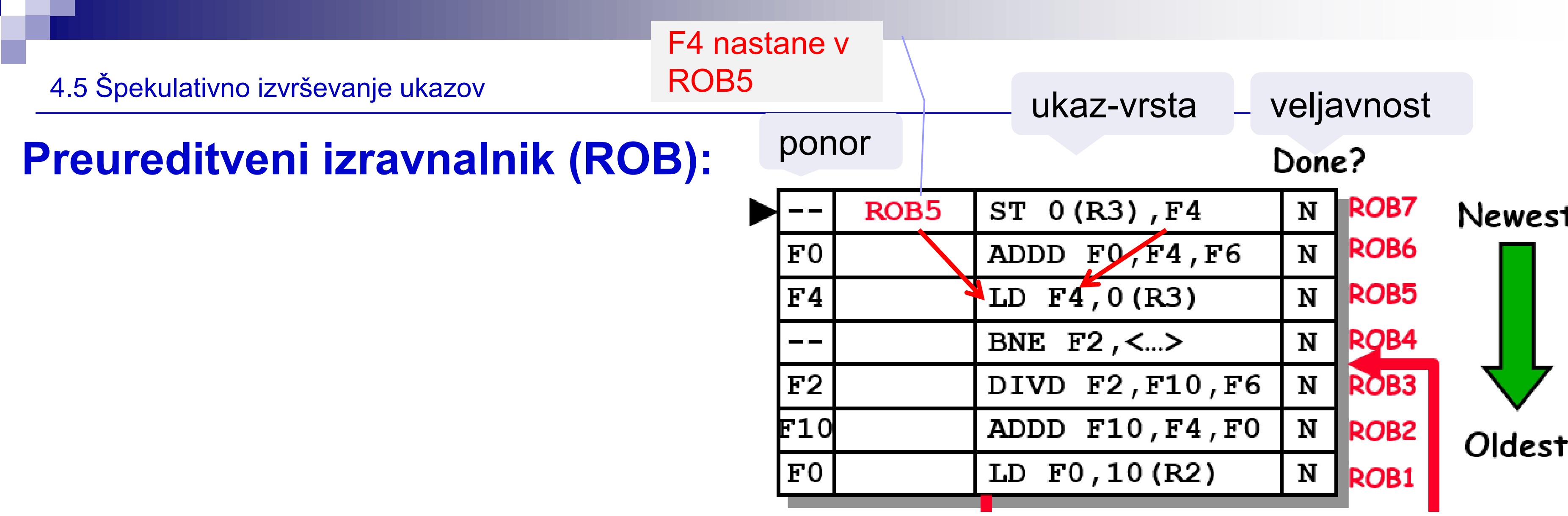
- Dinamično predikcijo skokov, pri kateri se kljub skokom izberejo ukazi, ki gredo v izstavljanje.
- Dinamično razvrščanje ukazov.
- Špekulativno izvrševanje ukazov brez predhodnega čakanja na preverjanje pravilnosti skočne predikcije in z možnostjo izničenja vpliva napačno izvršenih ukazov pri napačni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

© 2024 Rozman - FRI

Preureditveni izravnalnik (ROB):



- 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

© 2024 Rozman - FRI

4.5 Špekulativno izvrševanje ukazov

Čakalna vrsta ukazov

Ukazno vodilo UV

Skupno podatkovno vodilo SPV

Operandno vodilo OV

Rezervacijske postaje | Rezervacijske postaje | . . . | Rezervacijske postaje | Rezervacijske postaje

Funkcijska enota | Funkcijska enota | . . . | Funkcijska enota | Funkcijska enota

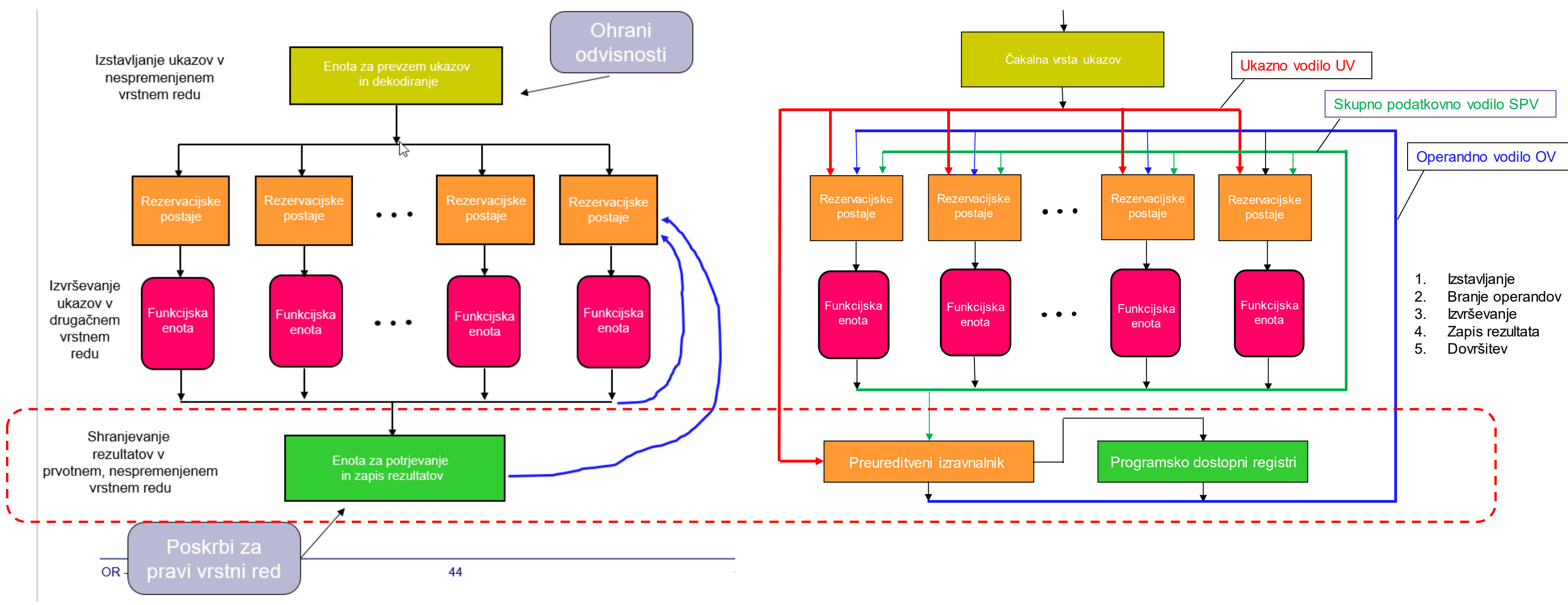
Preureditveni izravnalnik

Programsko dostopni registri

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

© 2024 Rozman - FRI

4.5 Špekulativno izvrševanje ukazov – primerjava z dinamičnim razvrščanjem ukazov



© 2024 Rozman - FRI

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

 © 2024 Rozman - FRI

- **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.

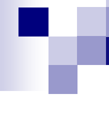
- **Zapis rezultata (SPV):**
  - Ko je na izhodu funkcijske enote rezultat ukaza, se prenese (SPV):
    - v vse rezervacijske postaje, ki čakajo nanj
    - in v preureditveni izravnalnik, v ukaz ki čaka na ta rezultat.

- **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 sprosti prostor za novi ukaz

 © 2024 Rozman - FRI

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«): l. 2019	352	>125 (???)	„..352 entries might be equivalent to as much as 525 µOPs…“
Intel Core iX (»Golden Cove«): l. 2021	512	?	

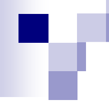


Spletna demonstracija delovanja PI :

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

Prednosti špek. izvrševanja ukazov:

- *Ukaz #4 (DIV.D) se začne izvajati 1. cikel pred MUL.D, ki čaka na rezultata obeh LD ukazov*
- *Ukaz DIV.D konča izvedbo že v 45. ciklu (brez ŠI „in-order“ bi končal v 56. ciklu)*



4.6 Večizstavitveni procesorji

Z znanimi metodami:

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

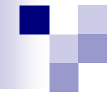
se CPI (povprečno število urinih period za izvedbo ukaza) približa 1.

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.

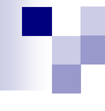
$$IPC = \frac{1}{CPI}$$

© 2024 Rozman - FRI



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.

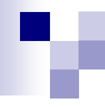


Strojno ugotavljanje medsebojnih podatkovnih odvisnosti je zahtevno za realizacijo:

- 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

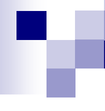
Izraz superskalarni procesor:

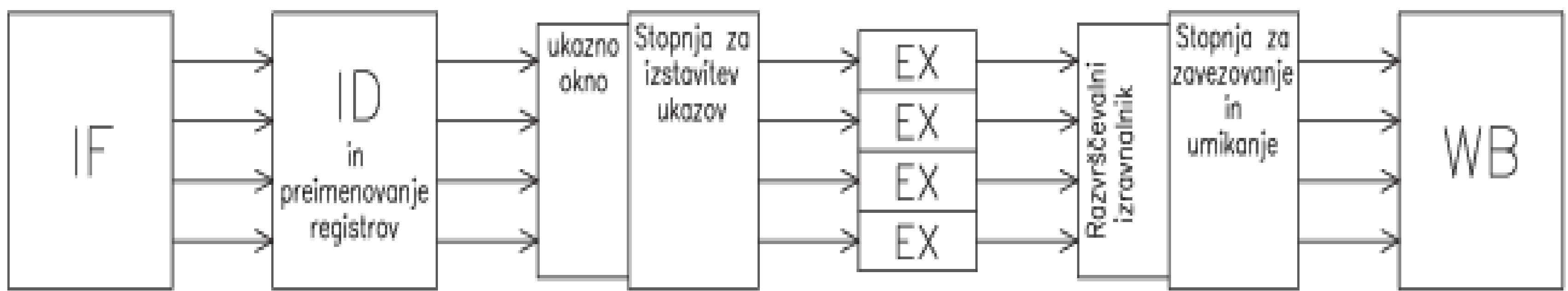
- prvič uporabljen pri IBM projektu America, ki je bil osnova za serijo procesorjev IBM RS/6000 (Power1),
- nekateri viri: „da je bil že superračunalnik CDC 6600 leta 1963 superskalarni“.

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.





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.

© 2024 Rozman - FRI

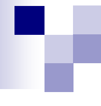
Lastnosti nekaterih superskalarnih procesorjev

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

 © 2024 Rozman - FRI



- 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.

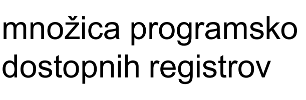
Primer: Pentium III: :

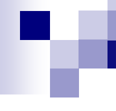
- PI porabi 27% celotne energije !!!
- „Preveč zapleteno, preveč porabe!“

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

razširjena množica registrov

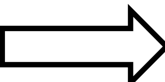




Preimenovanje registrov – osnovni pristop

Želimo izvesti 2 operaciji :
- M[1024] + 2 -> M[1032]
- M[2048] + 4 -> 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

© 2024 Rozman - FRI

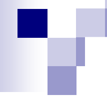
Razreši WAW/WAR
RAW ostanejo

Preimenovalna tabela

Prvotni ukazi	r1	r2	r3	Prosti reg.	Preimenovanje reg.
	l1	l2	l3	l4,l5,l6,l7	
add r1,r2,r3	l4	l2	l3	l5,l6,l7	add l4,l2,l3
sub r3,r2,r1	l4	l2	l5	l6,l7	sub l5,l2,l4
mul r1,r2,r3	l6	l2	l5	l7	mul l6,l2,l5
div r2,r1,r3	l6	l7	l5		div l7,l6,l5

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

© 2024 Rozman - FRI



4.6.2 VLIW procesorji - „Very Long Instruction Word“

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

Značilnosti:

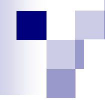
- t.i. **dolgi ukazi**:

Ukaz za funkc. enoto 1	Ukaz za funkc. enoto 2	Ukaz za funkc. enoto 3	…	Ukaz za funkc. enoto n

  - 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

l. 1997: HP, Intel predlagata EPIC:

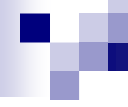
- „Explicitly Parallel Instruction Computing“
- Intel: Itanium 1, 2

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.“

© 2024 Rozman - FRI



4.7 Primeri izvedbe cevovodov

Tipični primeri izvedbe cevovodov:

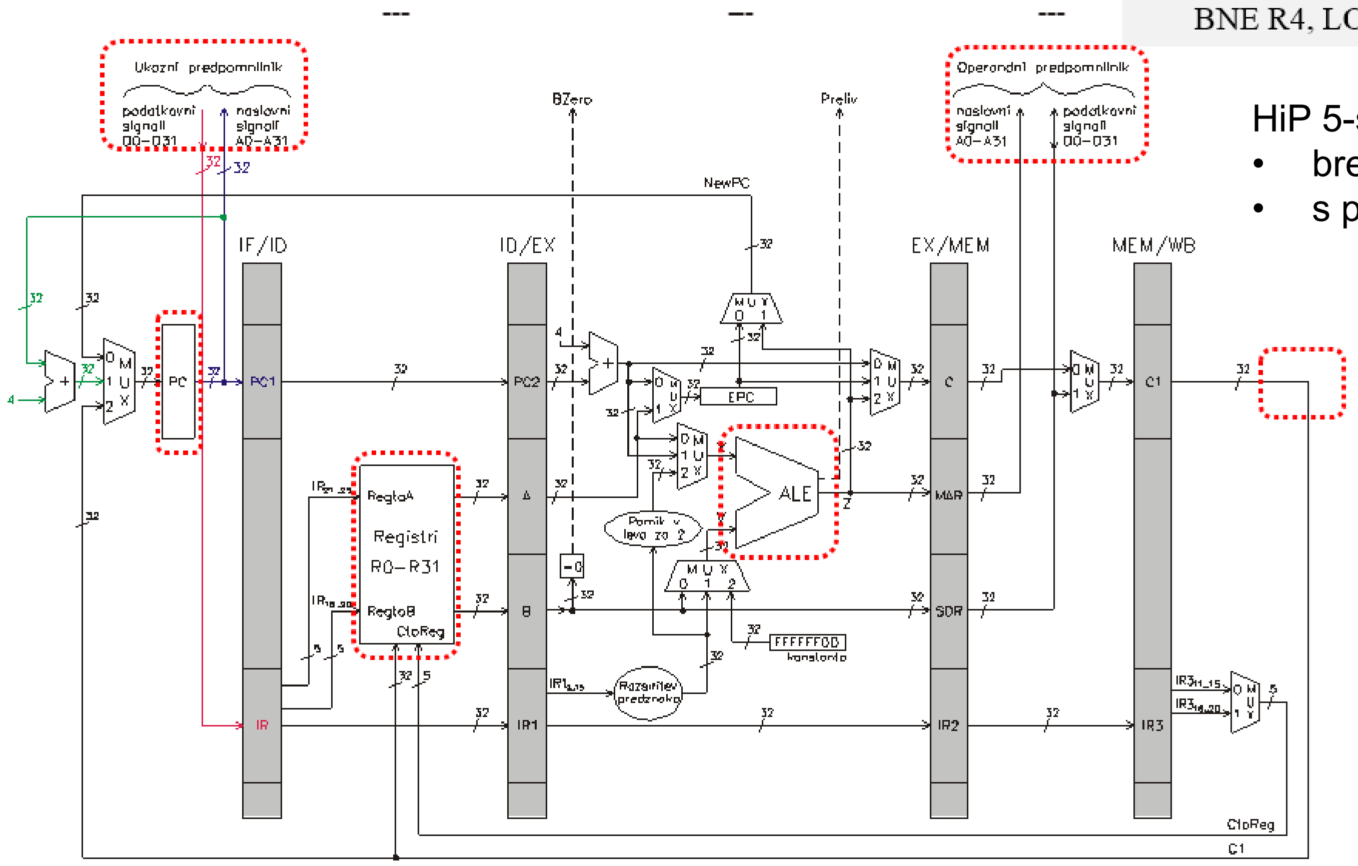
- HiP 5-stopenjski cevovod
- ARM9TDMI 5-stopenjski cevovod (FRI SMS)
- MiMo v2 – cevovodna različica
- ARM Cortex A8 cevovod
- ARM Cortex M7 (ST-H7) cevovod
- Core i7 cevovod
- AMD Zen mikro-arhitekture

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
```



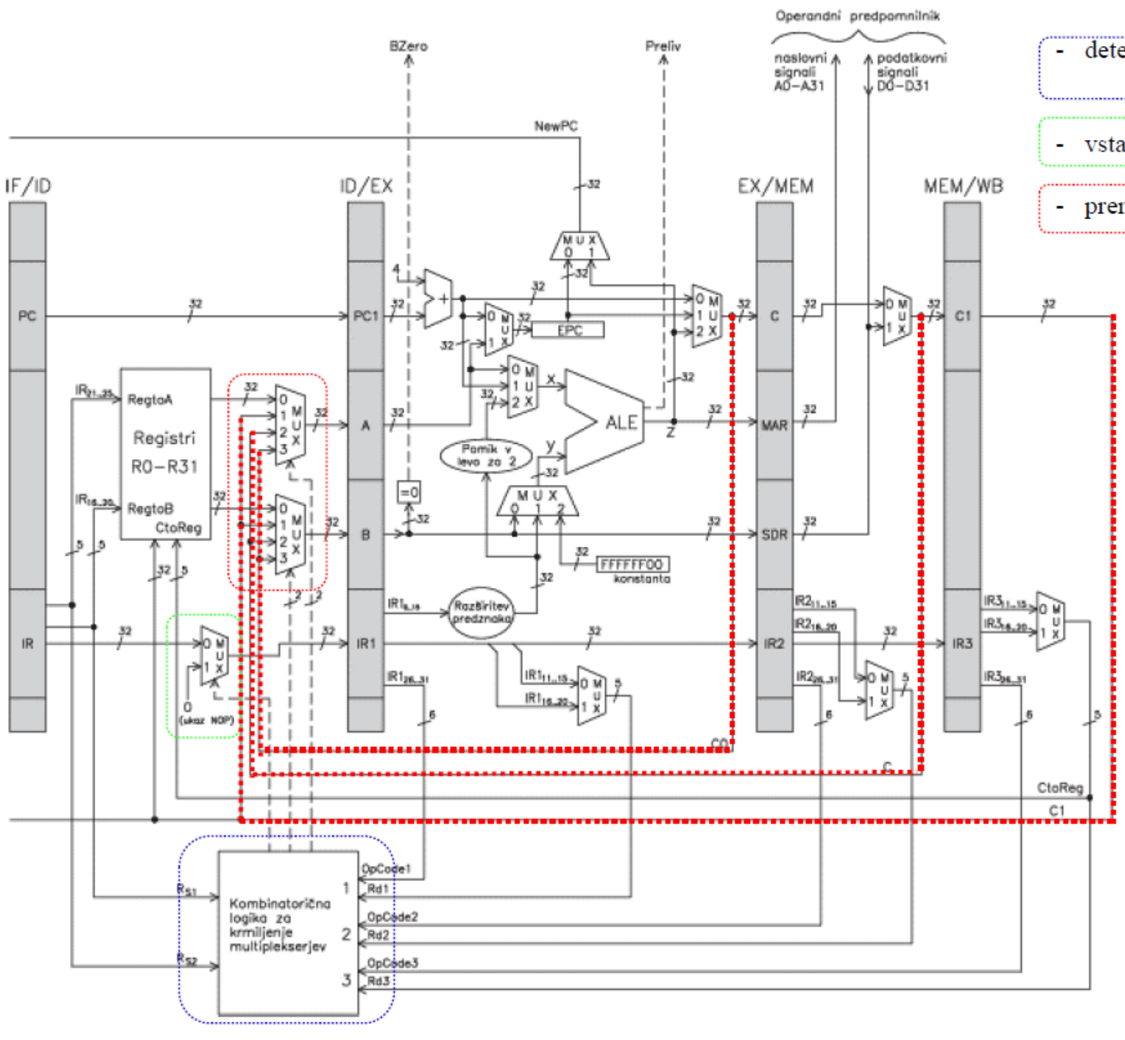
HiP 5-stopenjski cevovod :

- brez premoščanja 19 t_{cpe}
- s premoščanjem 11 t_{cpe}

4.7.1 HiP 5-stopenjski cevovod

HiP_cevovod_OR_v3_premoscanja.pdf (premoščanja)

HIP – shema z logiko za podatkovne nevarnosti in premoščanje



- detekcija nevarnosti
  - primerjalnik
- vstavljanje mehurčkov
- premoščanje

HiP 5-stopenjski cevovod :

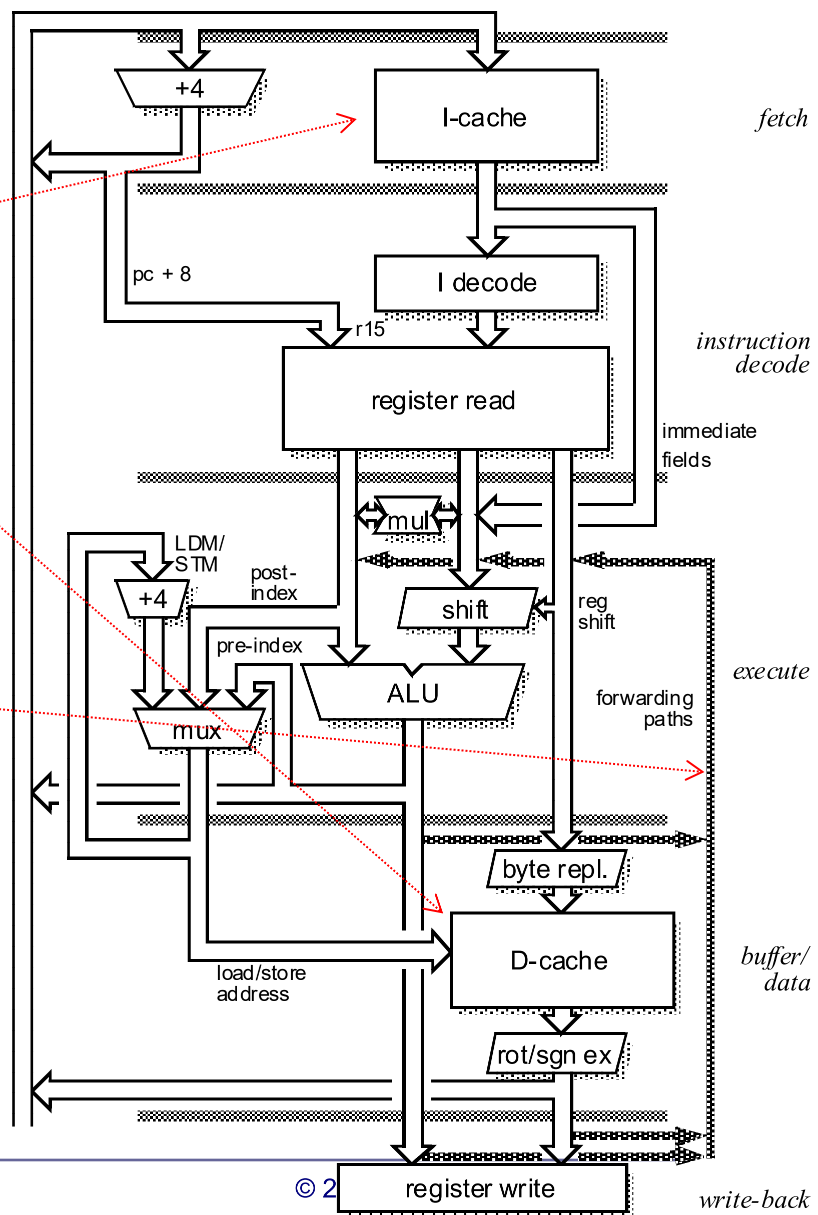
- brez premoščanja 19 t_{cpe}
- s premoščanjem 11 t_{cpe}

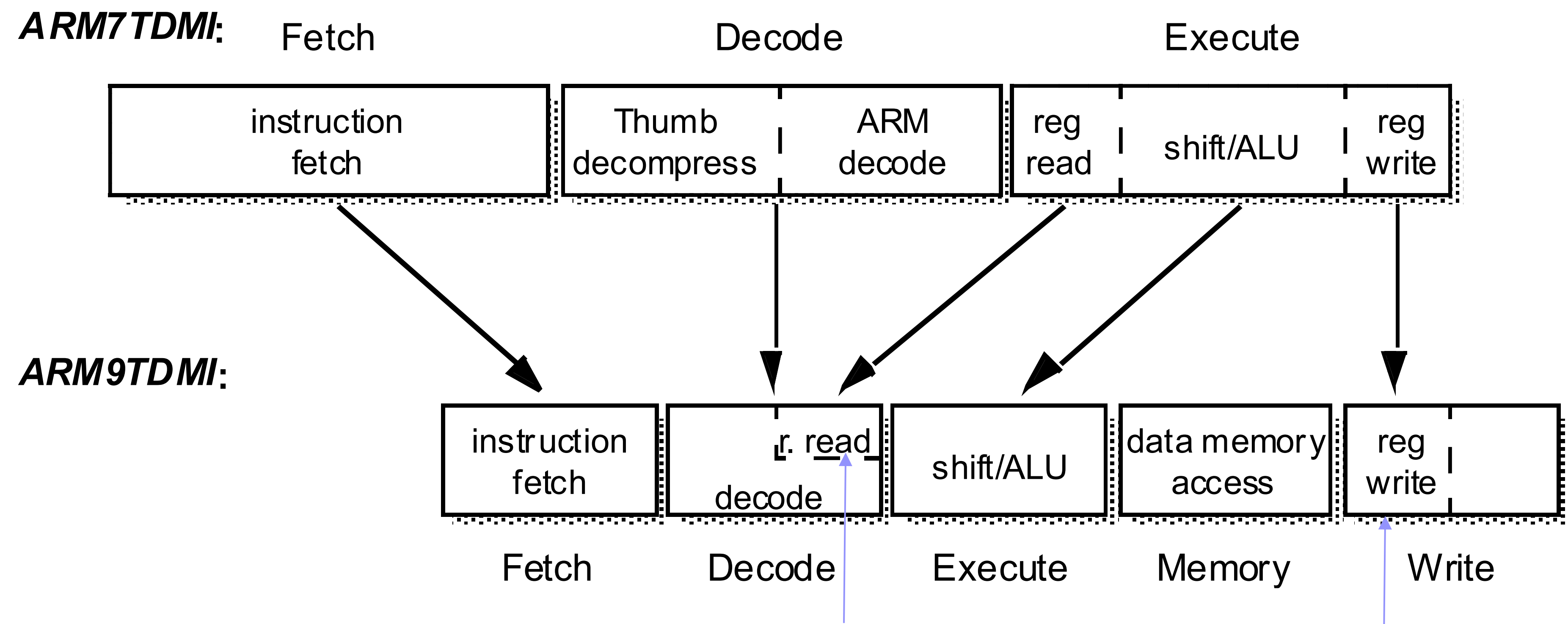
4.7.2 ARM9TDMI 5-stopenjski cevovod (FRI SMS)

- 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

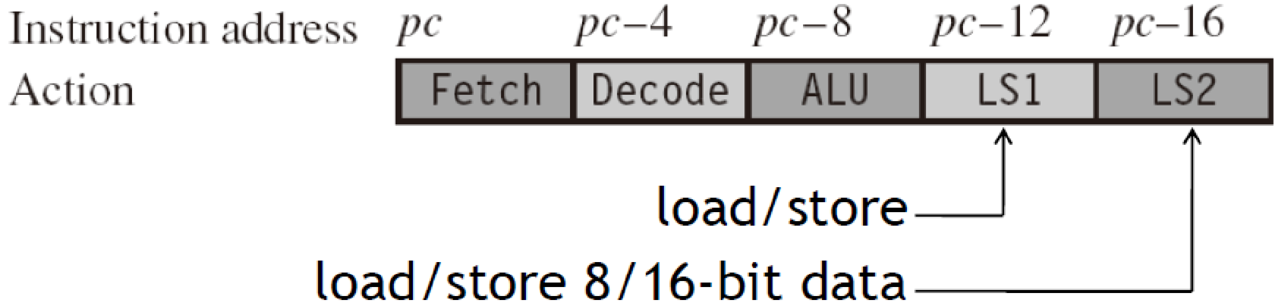




- ARM9TDMI podatki:

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)

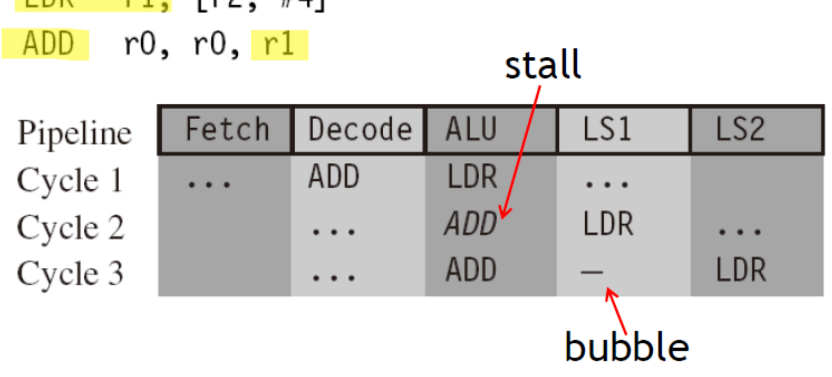


- No hazard, 2 cycles

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

- One-cycle interlock

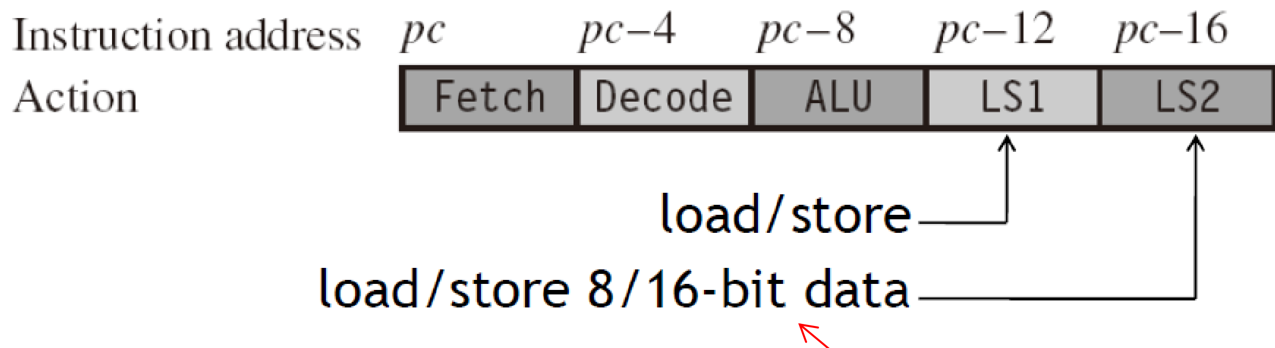
```
LDR   r1, [r2, #4]
ADD   r0, r0, r1
```



- Premoščanje reši podatkovno nevarnost
- LDR : dobi rezultat šele v LS1 !

© 2024 Rozman - FRI

4.7.2 ARM9TDMI 5-stopenjski cevovod (FRI SMS)



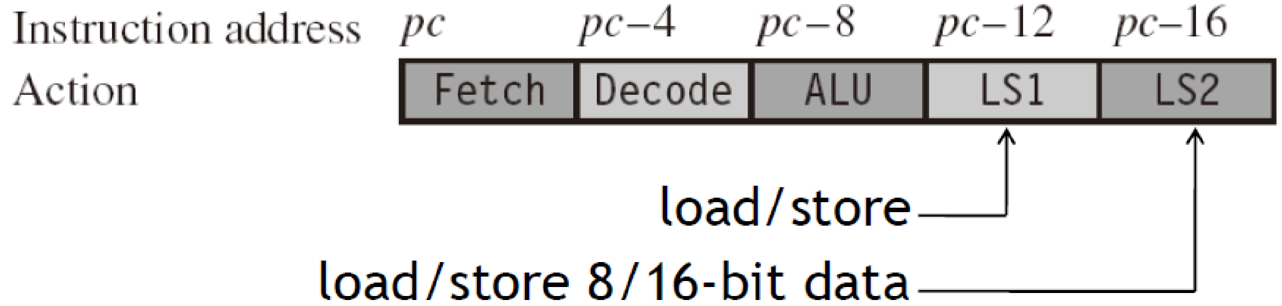
- One-cycle interlock, 4 cycles

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

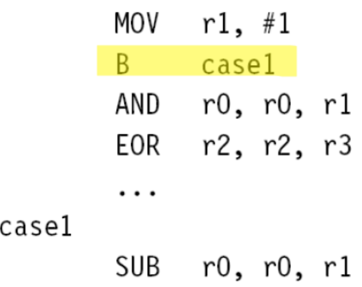
Pipeline	Fetch	Decode	ALU	LS1	LS2
Cycle 1	EOR	ADD	LDRB	...	
Cycle 2	...	EOR	ADD	LDRB	...
Cycle 3		...	*EOR*	ADD	LDRB
Cycle 4		...	EOR	—	ADD

- LDRB : dobi rezultat šele v LS2 !

4.7.2 ARM9TDMI 5-stopenjski cevovod (FRI SMS)



- Brach takes 3 cycles due to stalls

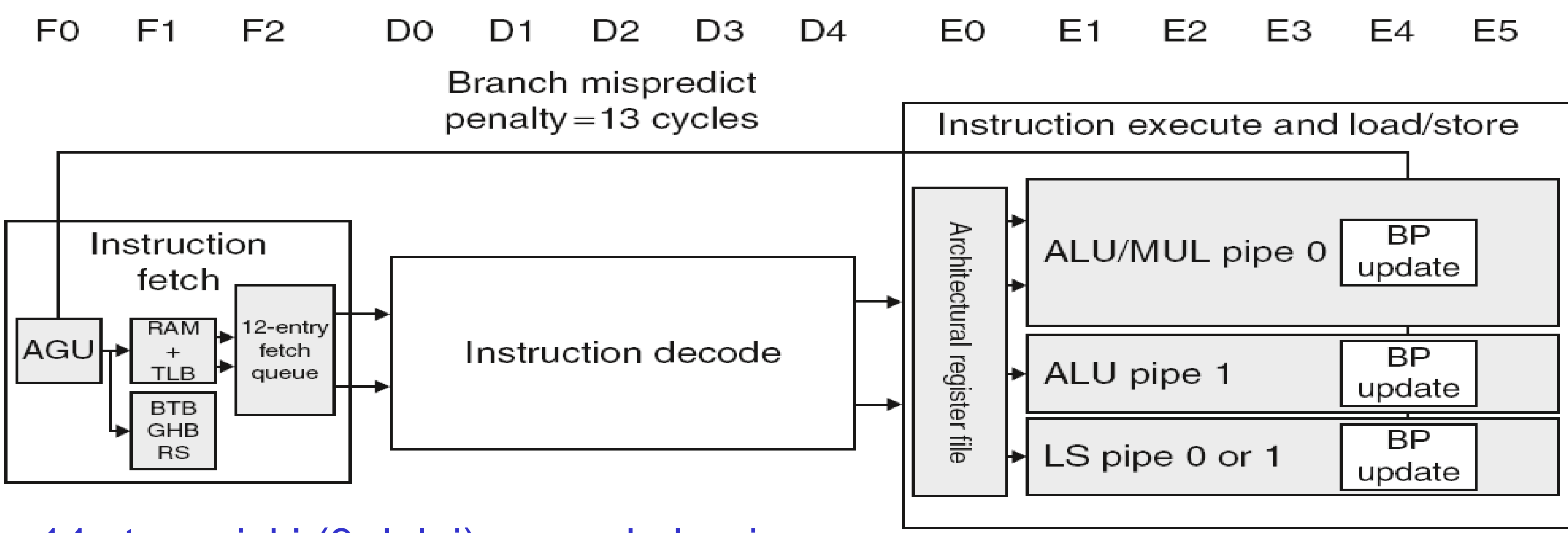


Pipeline	Fetch	Decode	ALU	LS1	LS2
Cycle 1	AND	B	MOV	...	
Cycle 2	~~EOR~~	~~AND~~	B	MOV	...
Cycle 3	SUB	—	—	B	MOV
Cycle 4	...	SUB	—	—	B
Cycle 5		...	SUB	—	—

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

© 2024 Rozman - FRI

4.7.3 ARM Cortex-A8 Cevovod (poenostavljen)

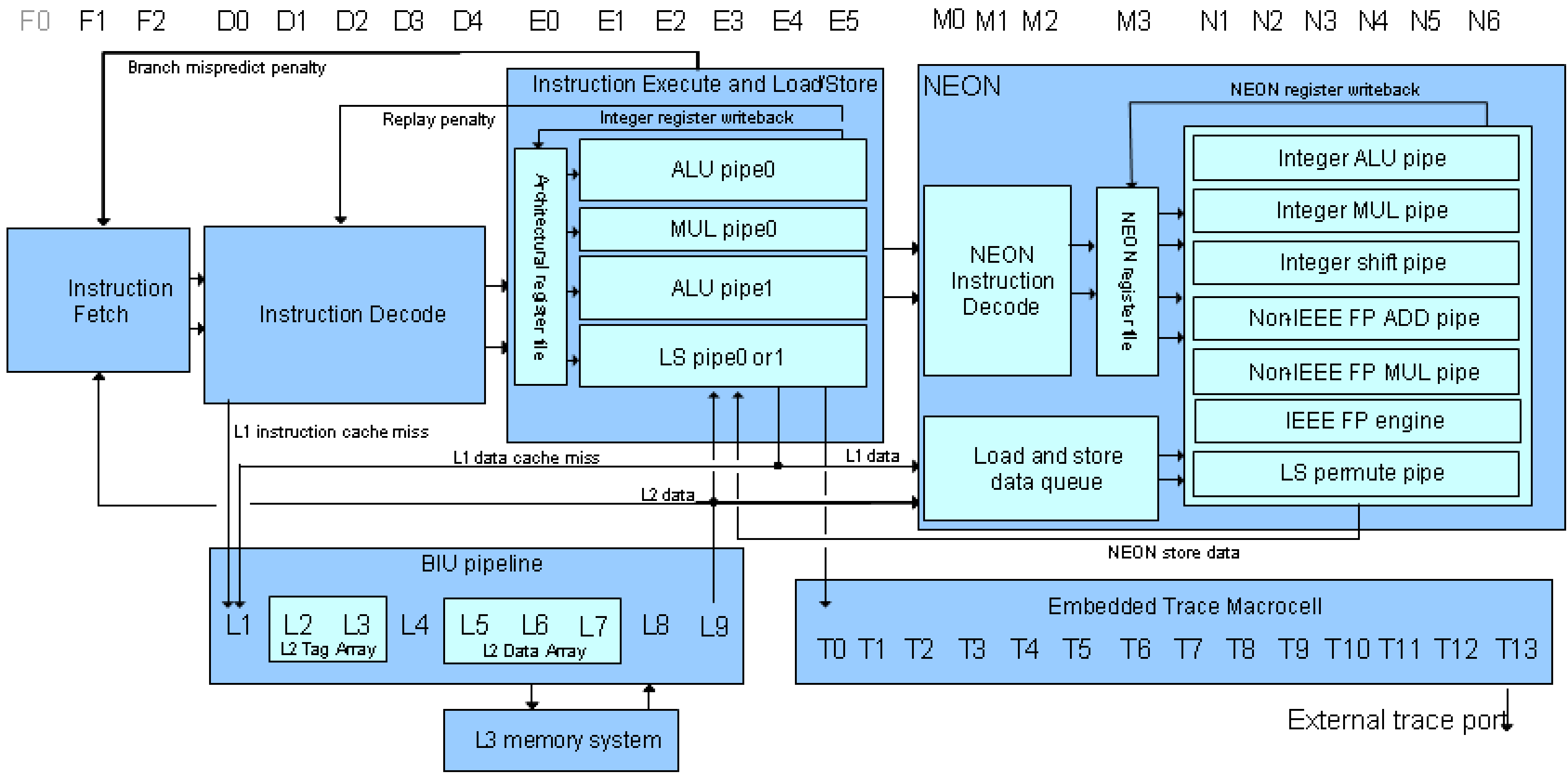


- 14 stopenjski (3 delni) superskalarni cevovod
  - dinamični več-izstavitveni
  - statični „in-order“ cevovod
    - pomembna vloga prevajalnika !
- odprava kontrolnih nevarnosti:
  - 2-stopenjska predikcija
- odprava podatkovnih nevarnosti:
  - polno premoščanje med EX cevovodi.

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

© 2024 Rozman

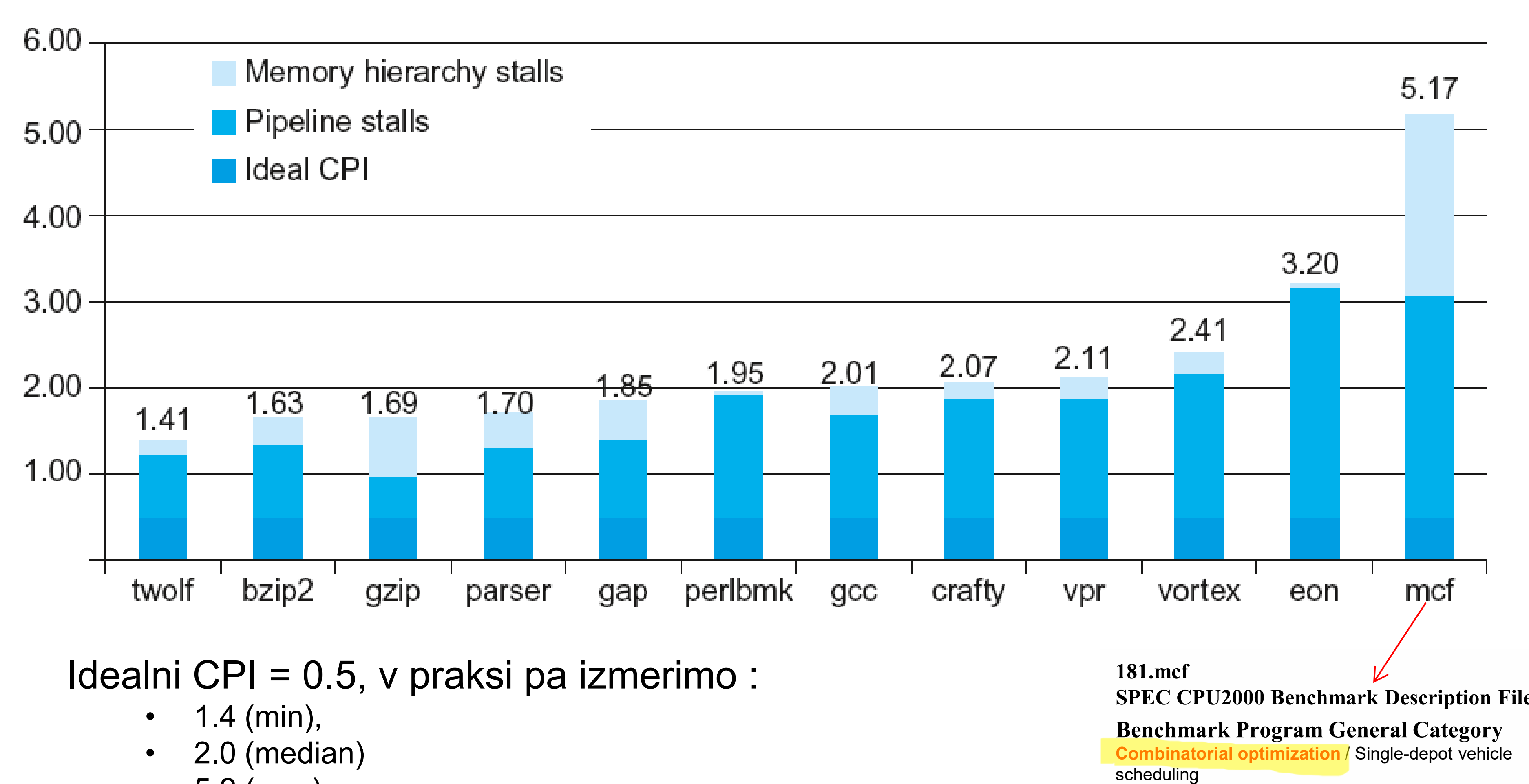
4.7.3 ARM Cortex-A8 Cevovod v celoti



4.7.3 ARM Cortex-A8 Cevovod – Minnespec Benchmarks

Meritve CPI na Minnespec Benchmarks:

- poenostavljen SPEC2000 (bistveno manjša količina vhodnih podatkov)



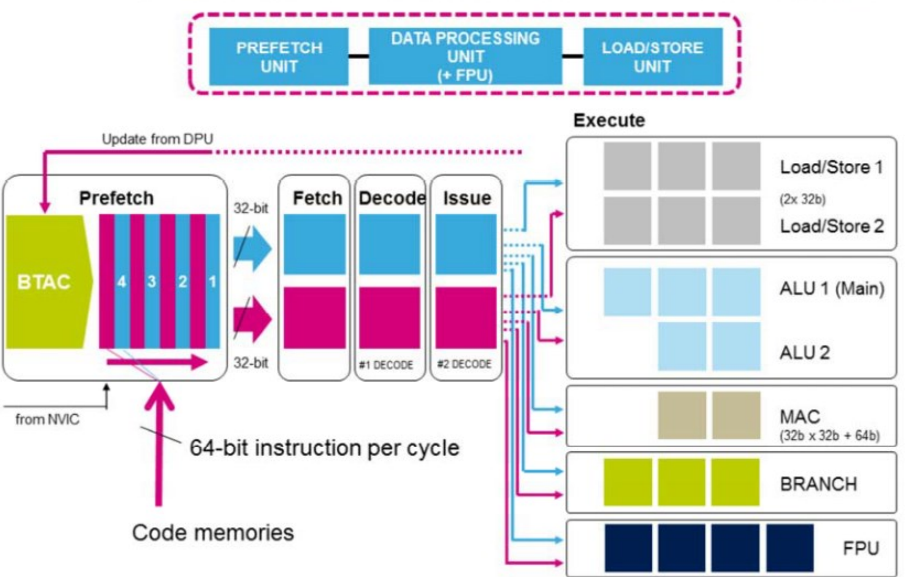
Idealni CPI = 0.5, v praksi pa izmerimo :

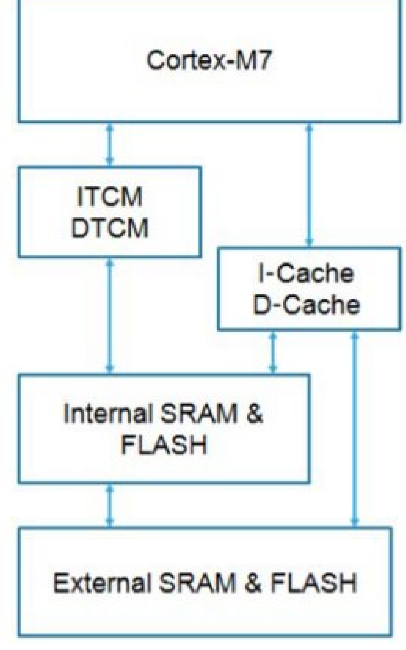
- 1.4 (min),
- 2.0 (median)
- 5.2 (max)…

4.7.4 ARM Cortex M7 mikroarhitektura (STM32H750)

STM32H7- ARM® Core

ARM Cortex®-M7 Core
Revision 1.0





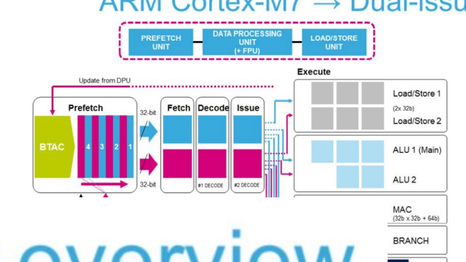
Značilnosti:

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

© 2024 Rozman

4.7.4 ARM Cortex M7 mikroarhitektura (STM32H750)

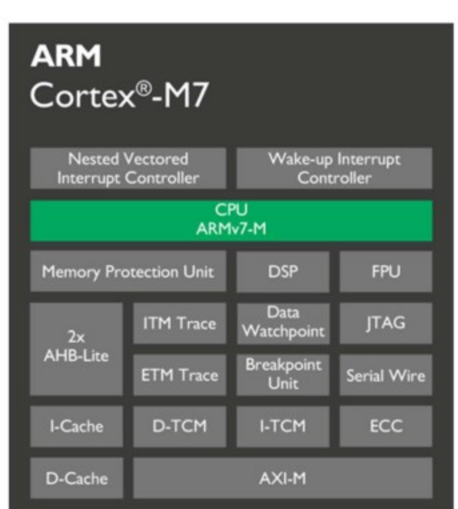
ARM Cortex-M7 → Dual-issue



Cortex-M7 processor overview

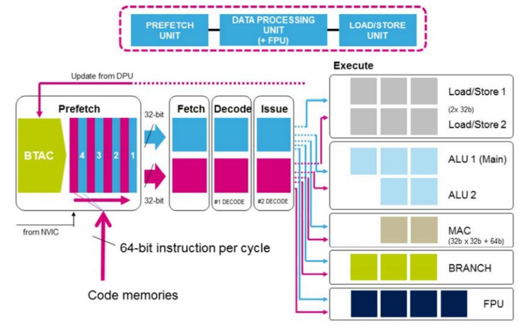
- ARMv7E-M architecture
- Harvard architecture, 6-stage pipeline
- Dual-issue superscalar architecture!
- DIV in 12 cycles (max.), SIMD instructions
- Memory Protection Unit (MPU)
- Single- and Double-precision Floating Point Unit (FPU)

⇒ Included in current STM32 products based on ARM Cortex®-M7



One step closer to DSPs	One step closer to Real-Time processors
Load and store in parallel with arithmetic operations	Tightly Coupled Memories
Zero overhead loops	AXI-M interface with Cache memory

ARM Cortex-M7 → Dual-issue



4.7.4 ARM Cortex M7 mikroarhitektura (STM32H750)

```
N=50:
first:
r0  81      @ DWT_CYCCNT (difference in cycles end-start)

r2  33      @ DWT_CPICNT - less stalls on further executions
r3  0       @ DWT_EXCCNT
r4  0       @ DWT_SLPCNT
r5  5       @ DWT_LSUCNT - 4 load,stores extra in code before disabling counters
r6  52      @ DWT_FOLDCNT

r8  95      @ Calculted num. of. instr.

second and further:
r0  56      @ DWT_CYCCNT (difference in cycles end-start)

r2  8       @ DWT_CPICNT - more stalls on first execution
r3  0       @ DWT_EXCCNT
r4  0       @ DWT_SLPCNT
r5  5       @ DWT_LSUCNT - 4 load,stores extra in code before disabling counters
r6  52      @ DWT_FOLDCNT

r8  95      @ Calculted num. of. instr.
```

```
// Register Addresses

    .equ     DWT_BASE,       0xE0001000 // DWT Base address

    .equ     DWT_CTRL,       0x00 // DWT_CTRL   reg (RM0433, pp.3209)
    .equ     DWT_CYCCNT,     0x04 // increments on each clock cycle when the processor is not halted in debu
    .equ     DWT_CPICNT,     0x08 // additional cycles required to execute multi-cycle instructions, and ins
    .equ     DWT_EXCCNT,     0x0C // count the total cycles spent in interrupt processing (cycles spent perf
    .equ     DWT_SLPCNT,     0x10 // count the total number of cycles during which the processor is sleeping
    .equ     DWT_LSUCNT,     0x14 // counts the total number of cycles that the processor is processing an I
                                  // For example, an LDR that takes two cycles to complete increments this c
                                  // Equivalently, an LDR that stalls for two cycles (and so takes four cycl
    .equ     DWT_FOLDCNT,    0x18 // count the total number of folded instructions (cycles saved by instruct
                                  // This counts 1 for each instruction that takes 0 cycles.

// If the processor configuration includes the DWT profiling counters, the instruction count can be calcula

// instructions executed = DWT_CYCCNT - DWT_CPICNT - DWT_EXCCNT - DWT_SLEEPCNT - DWT_LSUCNT + DWT_FOLDCNT
```

```
//-------------- subs,bne -----------------
//       ldr r5,=N
// ------------- odsek kode ----------------
// tloop:  subs r5,r5,#1
//         bne  tloop
// ------------- konec kode ----------------
/*
// Timings - usually in second or more repet
```

N	DWT_CYCCNT(1st)	DWT_CYCCNT(1st) Upd. 12/2024
50	56 (78)	56 (81)
100	106 (128)	106 (131)
200	206	
500	506	
1000	1006	
64000	64006 (64028)	

https://github.com/LAPSyLAB/ORLab-STM32H7/tree/main/DWT_Cycles_Measurements

© 2024 Rozman

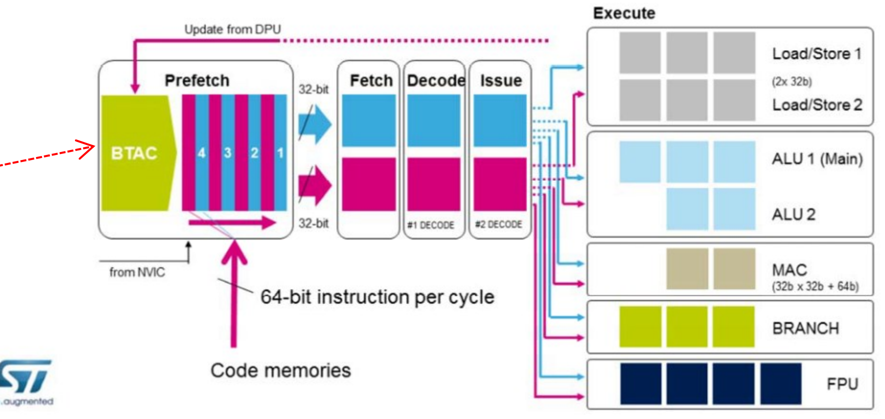
4.7.4 ARM Cortex M7 mikroarhitektura (STM32H750)

ARM Cortex-M7 → Dual-issue

PREFETCH UNIT — DATA PROCESSING UNIT (+ FPU) — LOAD/STORE UNIT

The prefetch unit (PFU) provides one 64-bit instruction per cycle to the **D**ata **P**rocessing **U**nit (DPU).
It includes:

- a buffer of 4 entries of 64 bits each to enable fetching ahead of the DPU
- and **B**ranch **T**arget **A**ddress **C**ache (BTAC) for single cycle branch prediction



The Cortex®-M7 core has a 6-stage dual-issue pipeline for efficient operation. It brings the ability to process 2 instructions in parallel if certain criteria are fulfilled.

When an instruction reaches the Issue stage, it is split into micro-operations and based on the needed operation and registers used. It is then issued to the appropriate blocks further in the processing pipe.

The data processing unit (DPU) is split into several pipes:

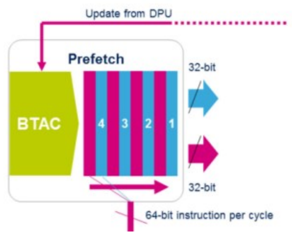
- Two ALUs, with one ALU capable of executing SIMD operations,
- Single MAC pipeline with one MAC per cycle capability,
- One floating point pipe supporting single and double precision operation.

© 2024 Rozman

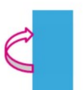
4.7.4 ARM Cortex M7 mikroarhitektura (STM32H750)

STM32H7

- Conditional loops need increment/decrement + branch execution
- On Cortex-M7: 1 cycle needed thanks to branch prediction and superscalar dual-issue architecture

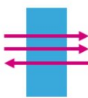


1. Superscalar → 2 instructions in 1 cycle



2. Branch in 1 cycle

3. Cache system to compensate slow memories



4. High system bandwidth for every application

As a branch can also be dual issued, it can be executed in parallel with computation.

Branch Target Address Cache or BTAC predicts whether the branch can be taken or not and reacts accordingly. It remembers the conditions and, based on the processing, it predicts the next address to fetch.

Forwarding of flags from the DPU to the PFU allows early resolution of direct branches in the decoder and first execution stages of pipeline.

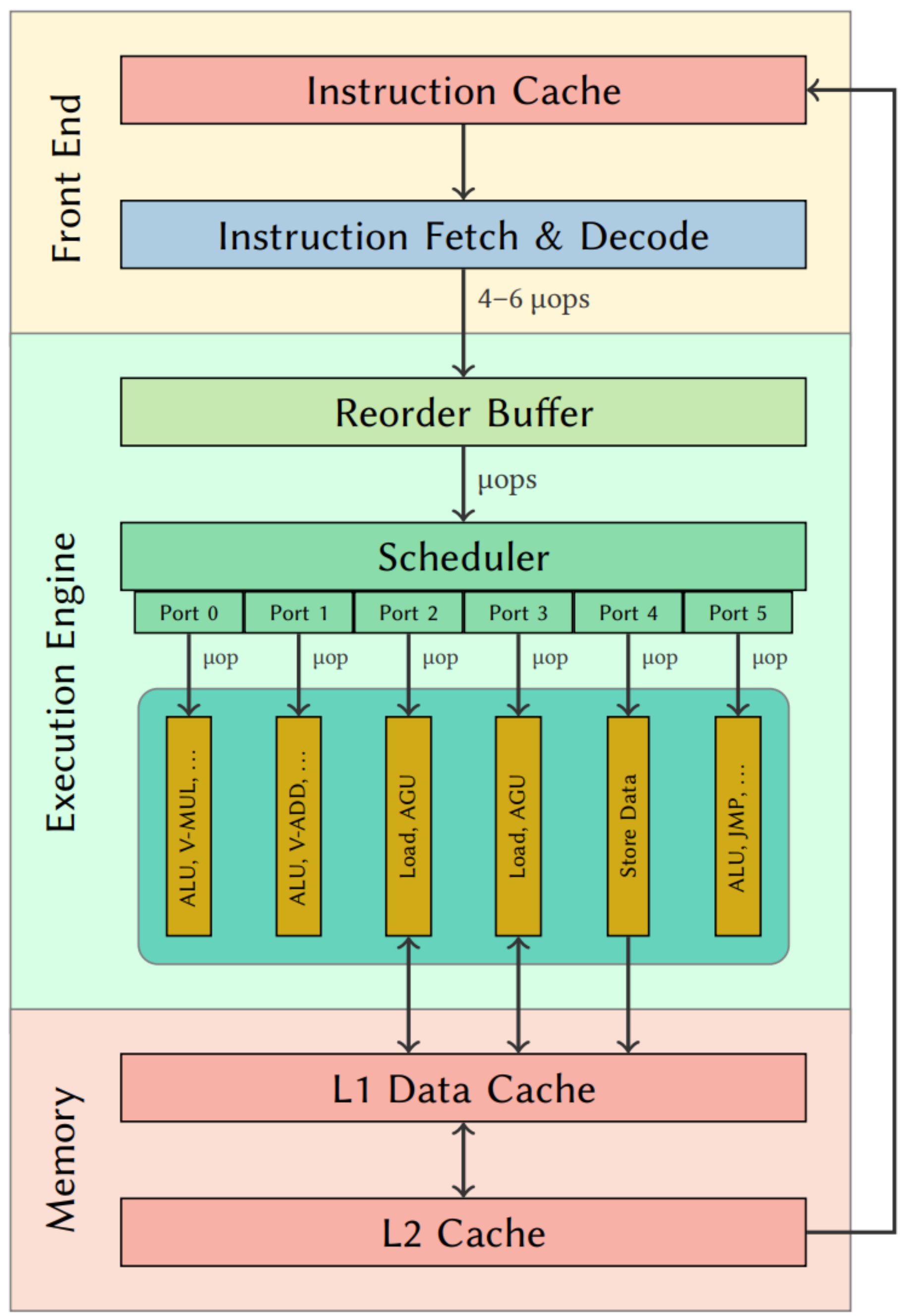
Tightly coupled memories (TCM)

- Critical code
- Interrupt service routines
- Highly deterministic

- Frequently used data
- Stack/Heap
- DSP coefficients

© 2024 Rozman

4.7.5 Core i7 superskalarni cevovod - poenostavljen



1.Instruction Fetch (16bajtov)

2.Predecode Stage (bajti->x86 ukaze)

3.μ-op decode (x86 ukazi -> μ-op)

5.Izstavitev μ-op -> ROB in RP

6.Izvedba μ-op

7.Dovršitev

© 2024 Rozman

4.7.5 Core i7 superskalarni cevovod

14 stopenjski superskalarni cevovod:

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

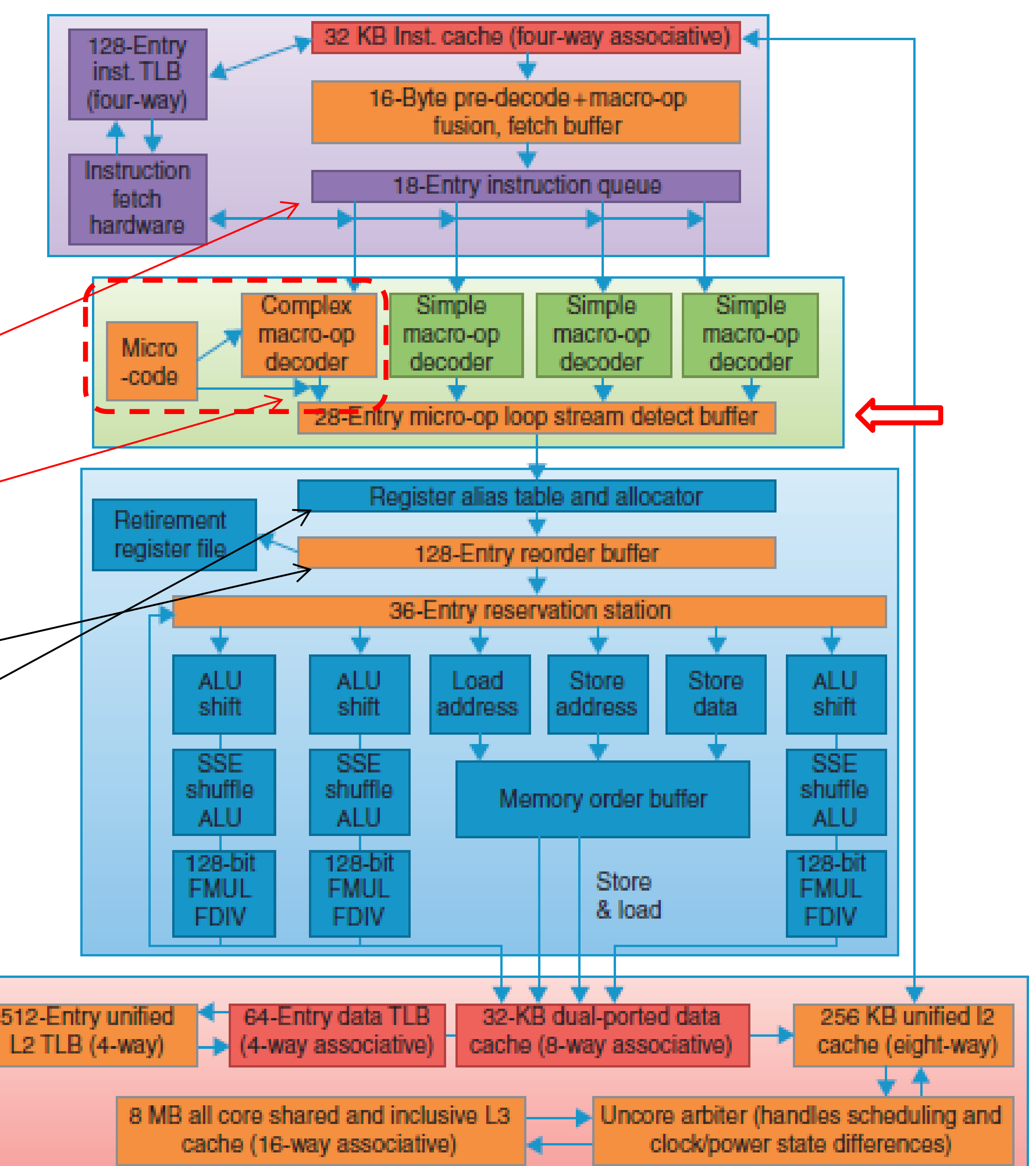
Osnovni problem :

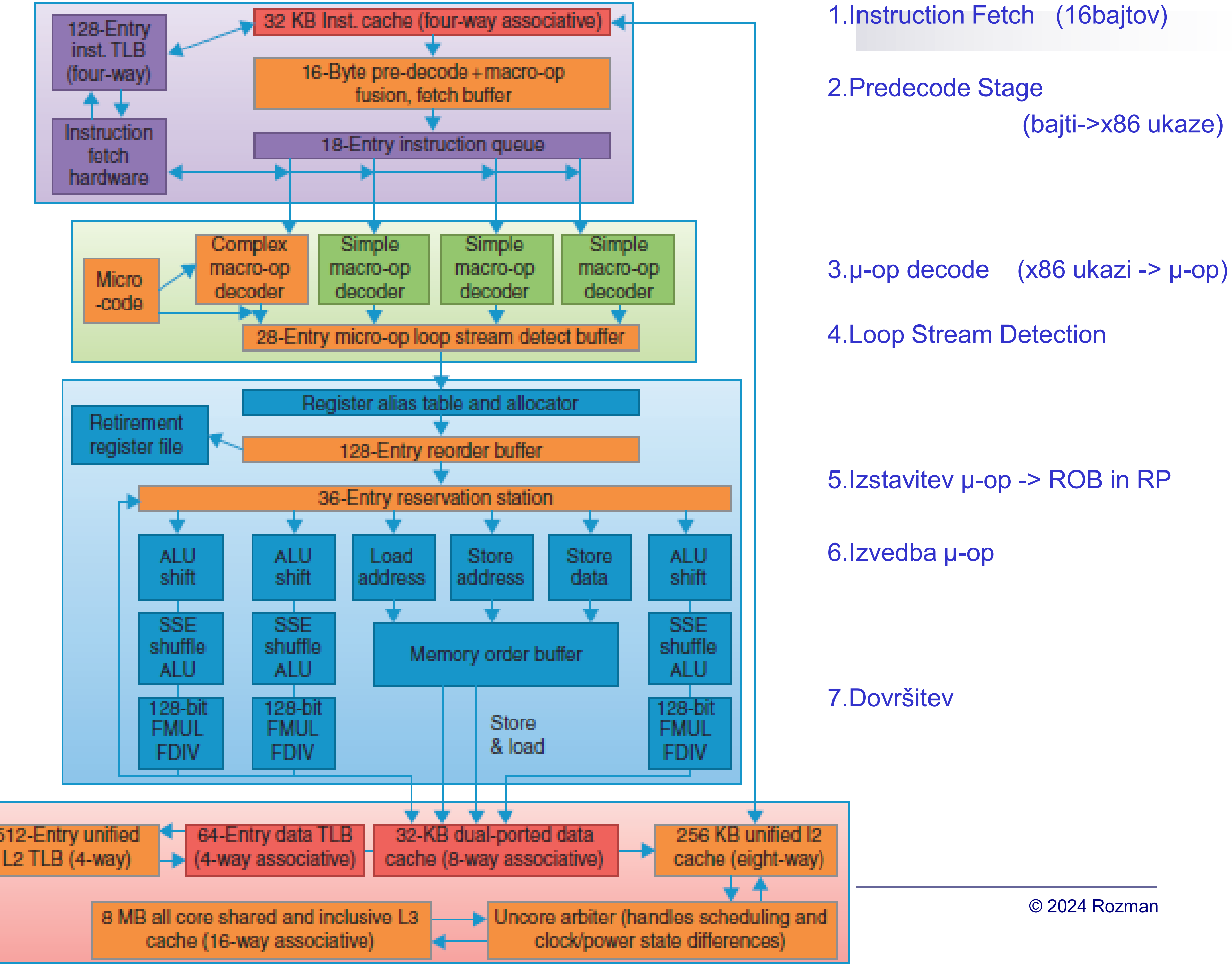
CISC ukazi

-> μ-operacije

-> cevovod

- značilnosti:
  - PI (ROB)
  - ekspl. preimenovanje reg.
- „kazni“ v t_{cpe} ciklih:
  - 15 napačno napovedan skok
  - 10 predp. L1 zgrešitev
  - 35 predp. L2 zgrešitev
  - 135 predp. L3 zgrešitev





© 2024 Rozman

4.7.5 Core i7 superskalarni cevovod

Resource	i7 920 (Nehalem)	i7 6700 (Skylake)	Sunny Cove
Micro-op queue (per thread)	28	64	140 (skupna)
Reservation stations	36	97	>125 (?)
Integer registers	NA	180	
FP registers	NA	168	
Outstanding load buffer	48	72	
Outstanding store buffer	32	56	
Reorder buffer	128	256	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.

© 2024 Rozman

4.7.5 Core i7 superskalarni cevovod

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 - 2023. Last updated 2023-05-26.

12.6 Macro-op fusion

The processor can fuse an arithmetic or logic instruction and a subsequent conditional jump instruction into a single compute-and-branch µop in certain cases. This works slightly different from previous processors. The fusion is not done by the decoders but immediately after the decode stage. The compute-and-branch µop is not split in two at the execution units but executed as a single µop by the branch unit.

First instruction	can pair with these, and the inverse	cannot pair with
cmp	jz, jc, jb, ja, jl, jg	js, jo, jp
add, sub, inc, dec	jz, jc, jb, ja, jl, jg	js, jo, jp
adc, sbb	none	
neg, not	none	
test	all	
and	all	
or, xor	none	
shift, rotate	none	

Table 12.1. Instruction fusion

8.4 Micro-op fusion

There are two cases of µop fusion: read-modify instructions and write instructions. A read-modify instruction needs one µop for reading a memory operand and another µop for doing a calculation with this operand. For example, `ADD EAX,[MEM]` needs one µop for reading `MEM` and one for adding this value to `EAX`. These two µops can be fused into one. A write instruction needs one µop for calculating the address and one for writing to that address. For example, `MOV [ESI+EDI],EAX` needs one µop for calculating the address `[ESI+EDI]` and one for storing `EAX` to this address. These two µops are fused together.

4.7.5 Core i7 superskalarni cevovod

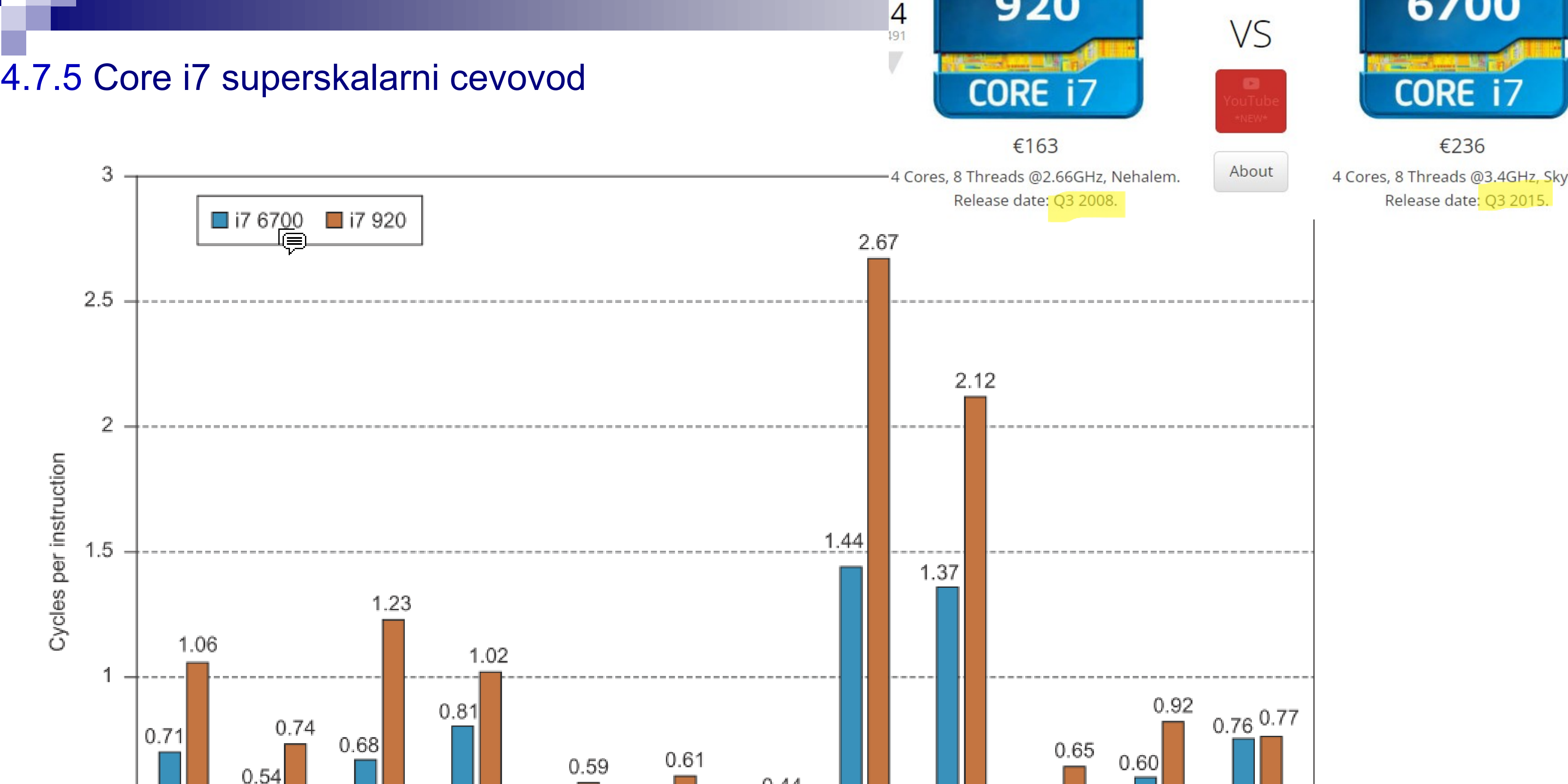
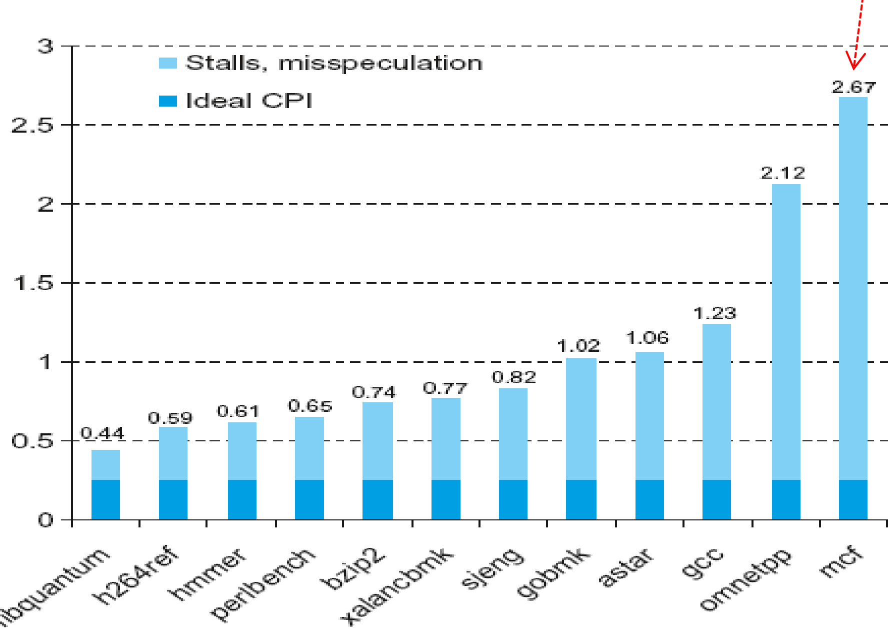
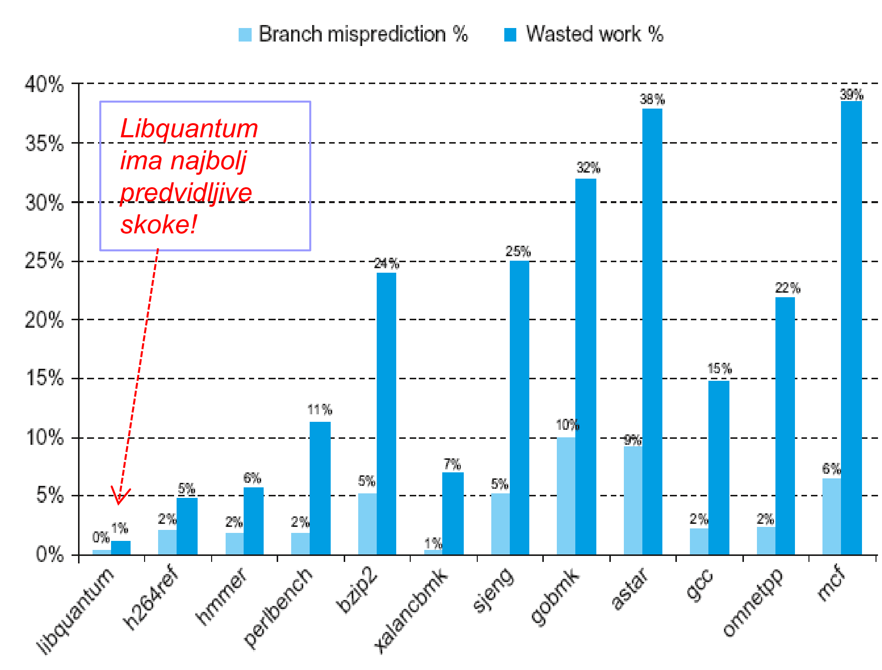


Figure 3.40 The CPI for the SPECCPUint2006 benchmarks on the i7 6700 and the i7 920. The data in this section were collected by Professor Lu Peng and PhD student Qun Liu, both of Louisiana State University.

Meritve na SPEC2006 :





Idealni CPI = 0.25, v praksi pa izmerimo :

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

Branch misprediction:

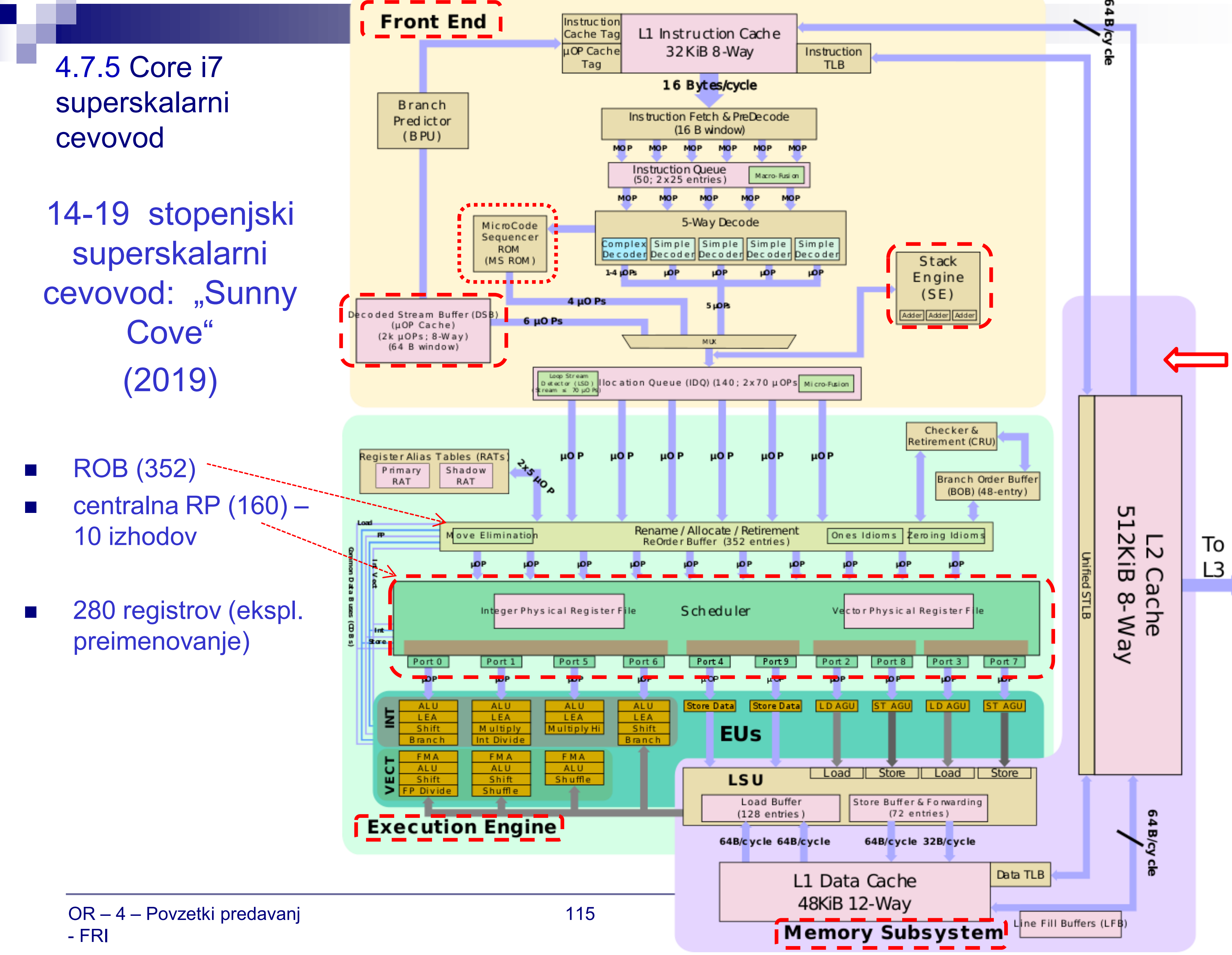
delež napačnih skočnih napovedi

0% (min), 2% (median) 10% (max)

Wasted work:

delež zavrženih μ-operacij

1% (min), 18% (median) 39% (max)



4.7.5 Core i7 superskalarni cevovod
14-19 stopenjski superskalarni cevovod: „Sunny Cove“
(2019)
ROB (352)
centralna RP (160) – 10 izhodov
280 registrov (ekspl. preimenovanje)
Front End
Instruction Cache Tag
µOP Cache Tag
L1 Instruction Cache
32KiB 8-Way
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
MOP
Instruction Queue (50; 2x25 entries)
Macro-Fusion
5-Way Decode
Complex Decoder
Simple Decoder
MicroCode Sequencer ROM (MS ROM)
1-4 µOPs
µOP
4 µOPs
5 µOPs
6 µOPs
Decoded Stream Buffer (DSB) (µOP Cache) (2k µOPs; 8-Way) (64 B window)
MUX
Stack Engine (SE)
Adder
Allocation Queue (IDQ) (140; 2x70 µOPs)
Micro-Fusion
Register Alias Tables (RATs)
Primary RAT
Shadow RAT
2x5 µOP
Checker & Retirement (CRU)
Branch Order Buffer (BOB) (48-entry)
Move Elimination
Rename / Allocate / Retirement
ReOrder Buffer (352 entries)
Ones Idioms
Zeroing Idioms
Integer Physical Register File
Scheduler
Vector Physical Register File
Port 0
Port 1
Port 5
Port 6
Port 4
Port 9
Port 2
Port 8
Port 3
Port 7
INT
ALU
LEA
Shift
Branch
Multiply
Int Divide
Multiply Hi
VECT
FMA
FP Divide
Shuffle
Store Data
LD AGU
ST AGU
EUs
Execution Engine
LSU
Load
Store
Load Buffer (128 entries)
Store Buffer & Forwarding (72 entries)
64B/cycle
32B/cycle
L1 Data Cache
48KiB 12-Way
Data TLB
Line Fill Buffers (LFB)
Memory Subsystem
Unified STLB
L2 Cache
512KiB 8-Way
To L3
64B/cycle
OR – 4 – Povzetki predavanj - FRI
115

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

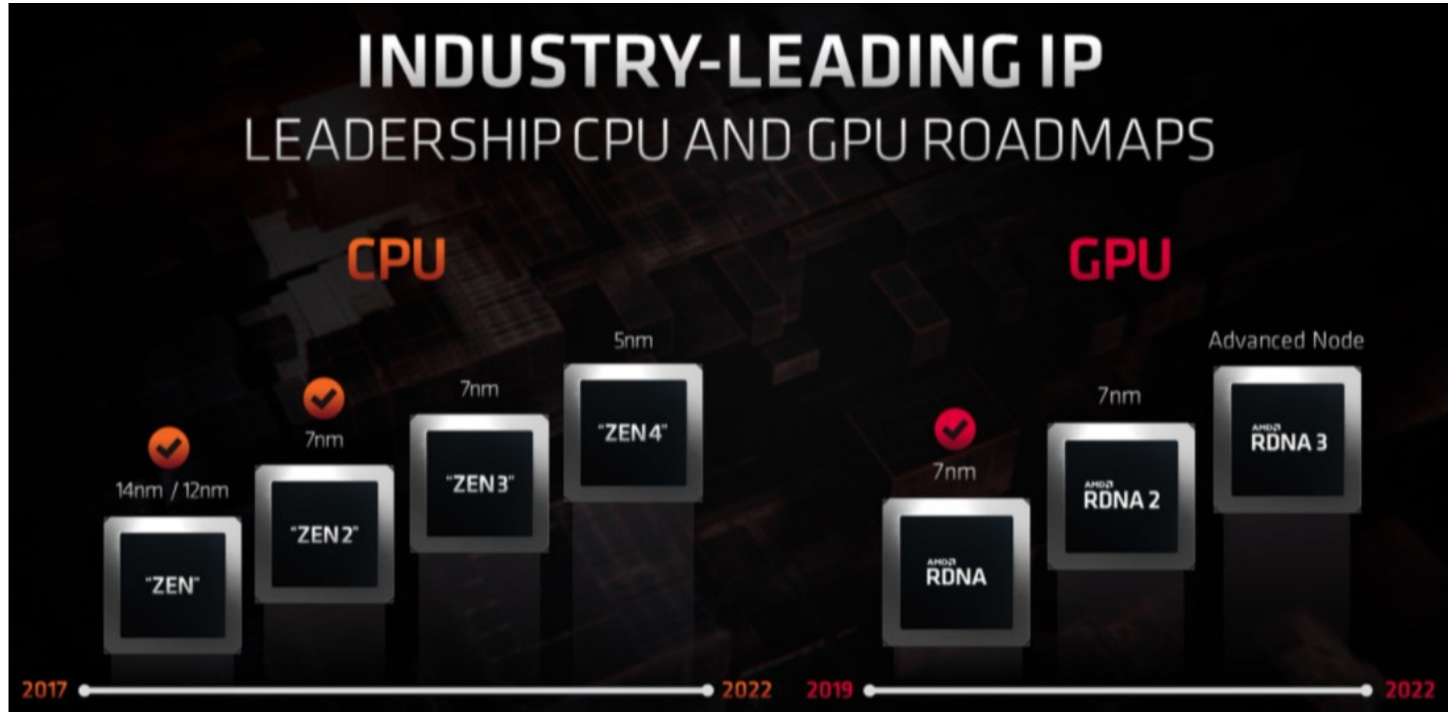
Vir: http://www.hep.wisc.edu/~pinghc/x86AssmTutorial.htm

© 2024 Rozman

4.7.5 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
1st level caches/core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D	16 KiB I, 16 KiB D
2nd level caches/core	128-1024 KiB	256 KiB	
3rd level caches (shared)	-	2- 8 MB	

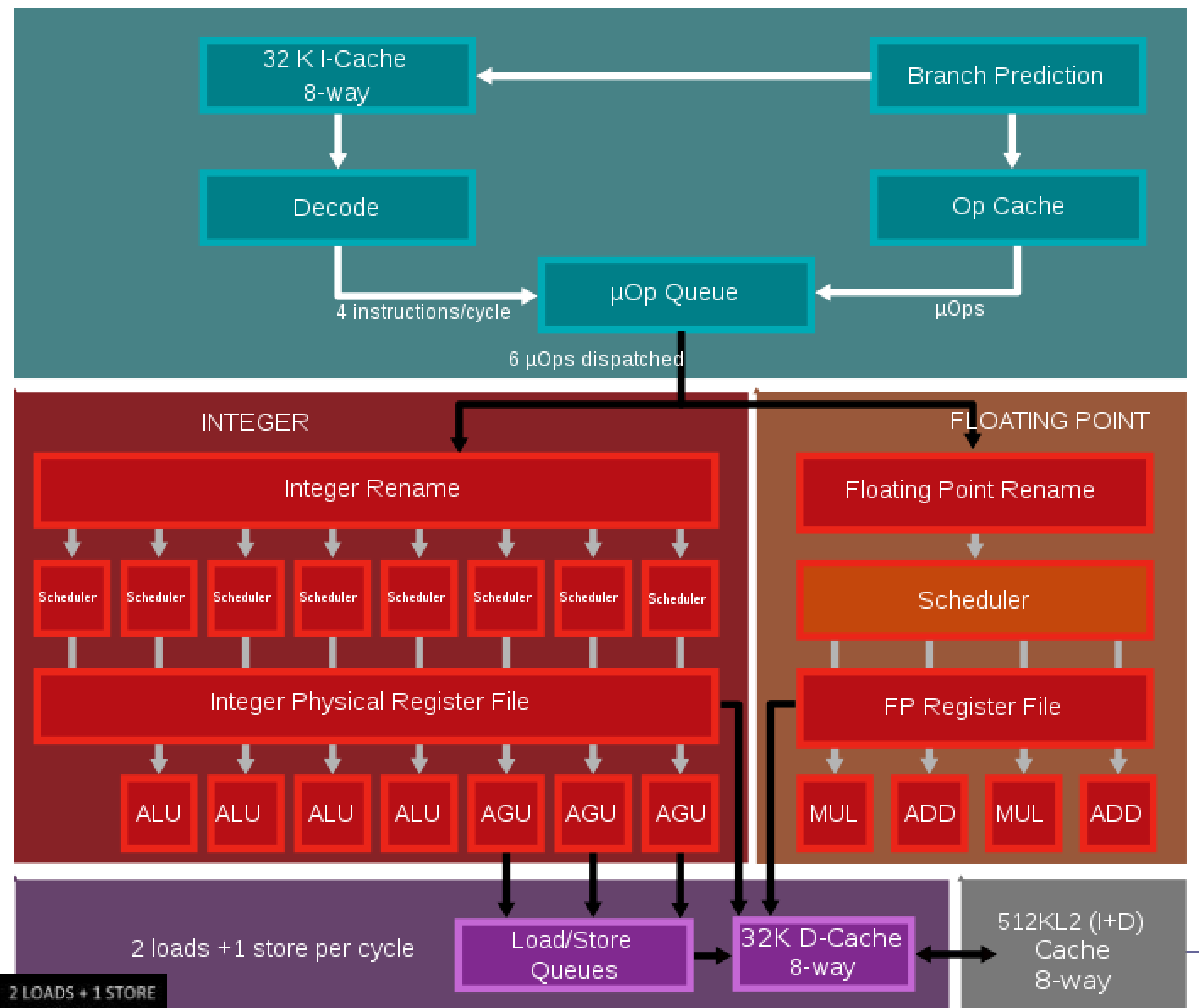
4.7.6 AMD „Zen“ mikroarhitekture



Novosti v „Zen“:

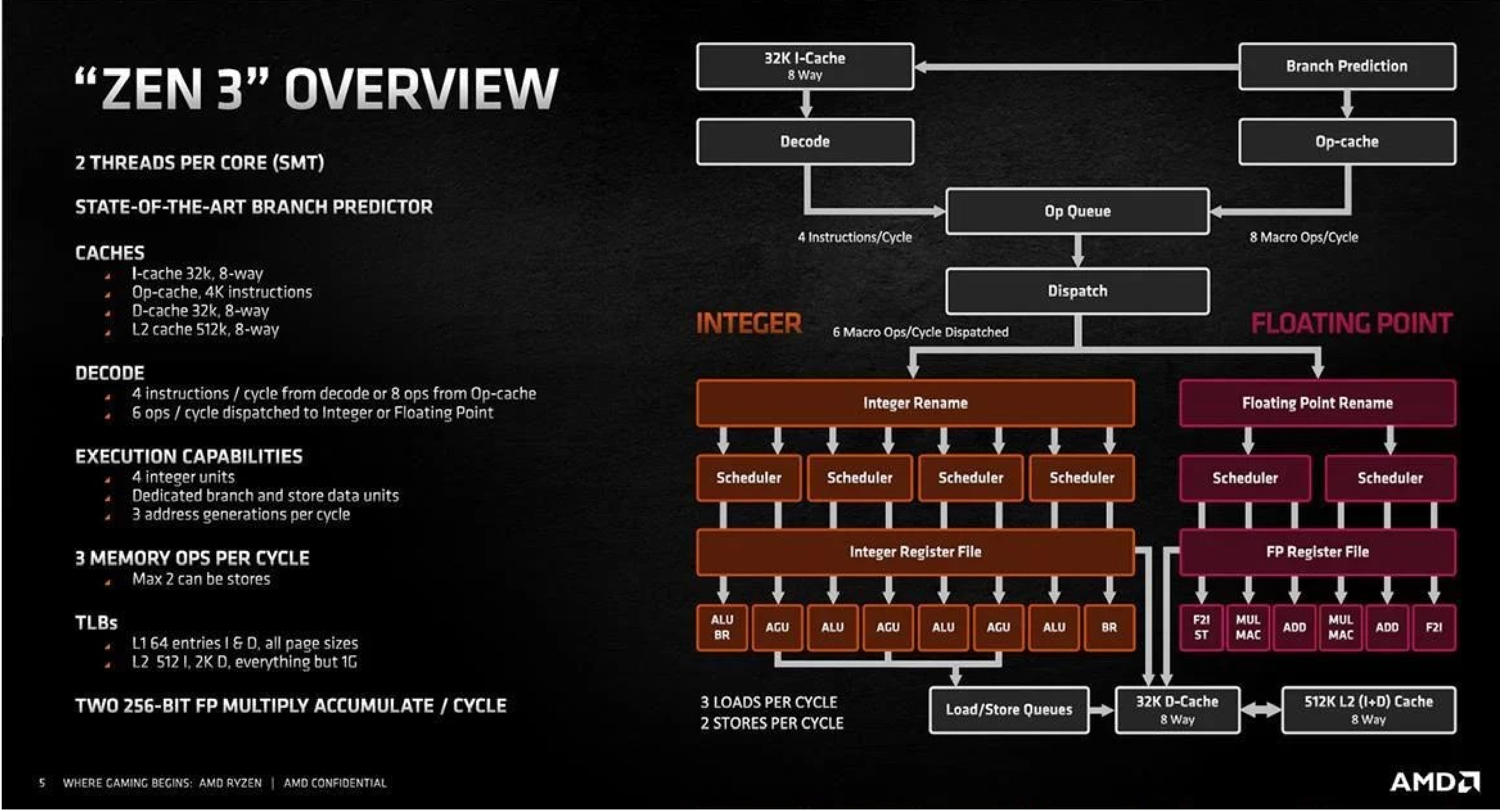
- „Design from scratch“
- 40% višji IPC od predhodnika
- SMT
- Izboljšan predpomnilniški sistem
- Evolucija: Zen+, Zen2, Zen 3, Zen 4, Zen 5

4.7.5 Zen 2 mikroarhitektura – poenostavljen model

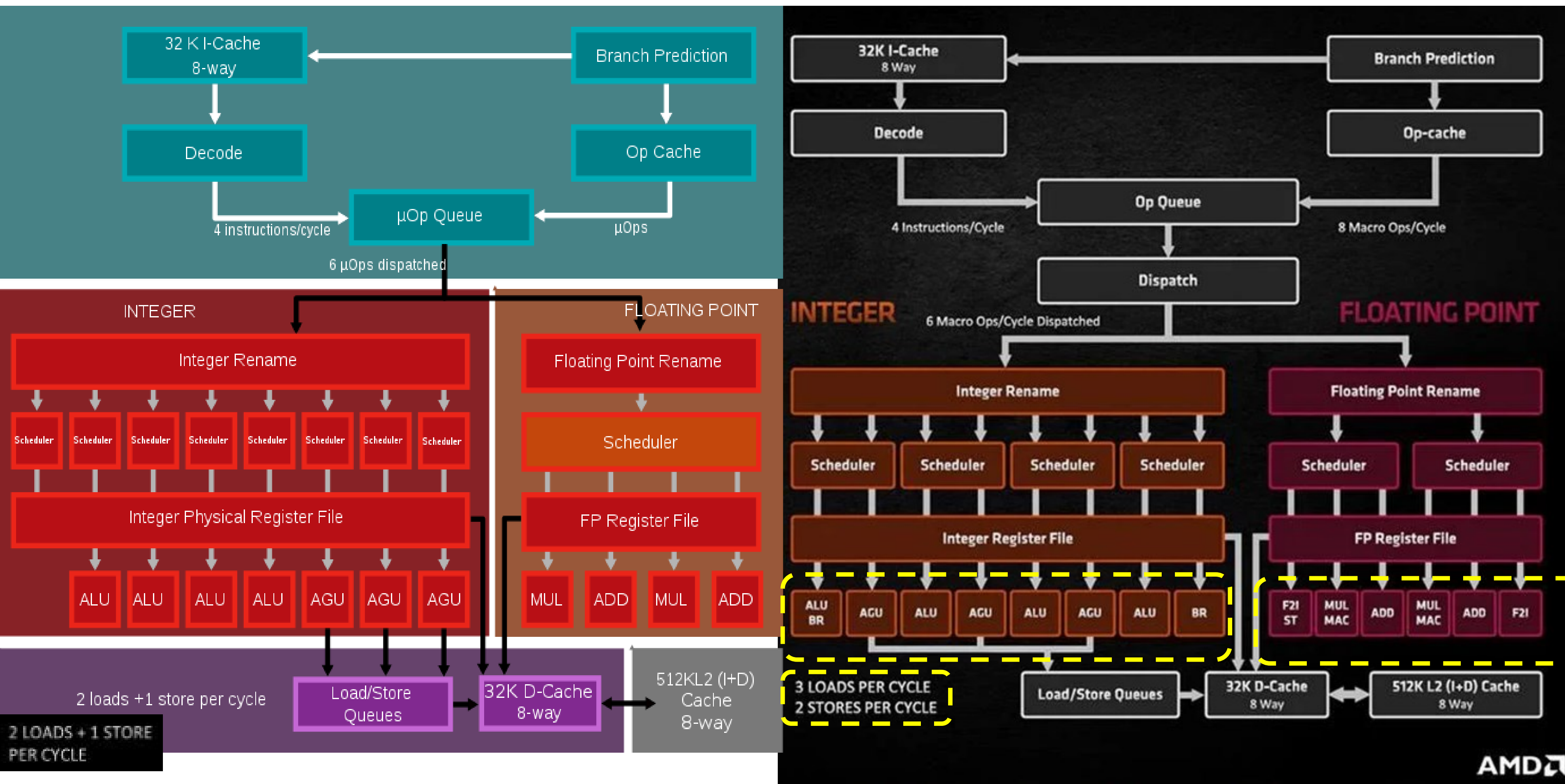


© 2024 Rozman

4.7.5 Zen 3 mikroarhitektura – poenostavljen model



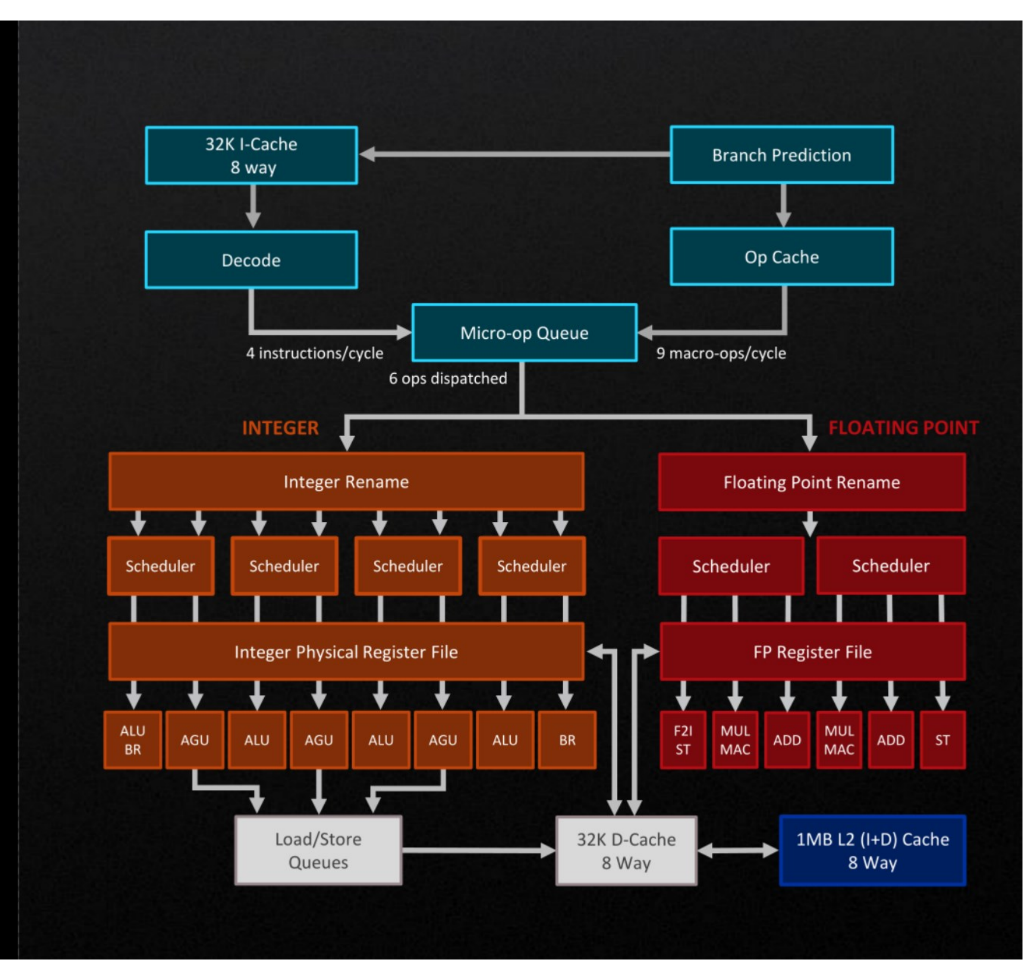
4.7.5 Zen 2 vs Zen 3 mikroarhitekturi – poenostavljena modela



4.7.5 Zen 4 mikroarhitektura – poenostavljen model

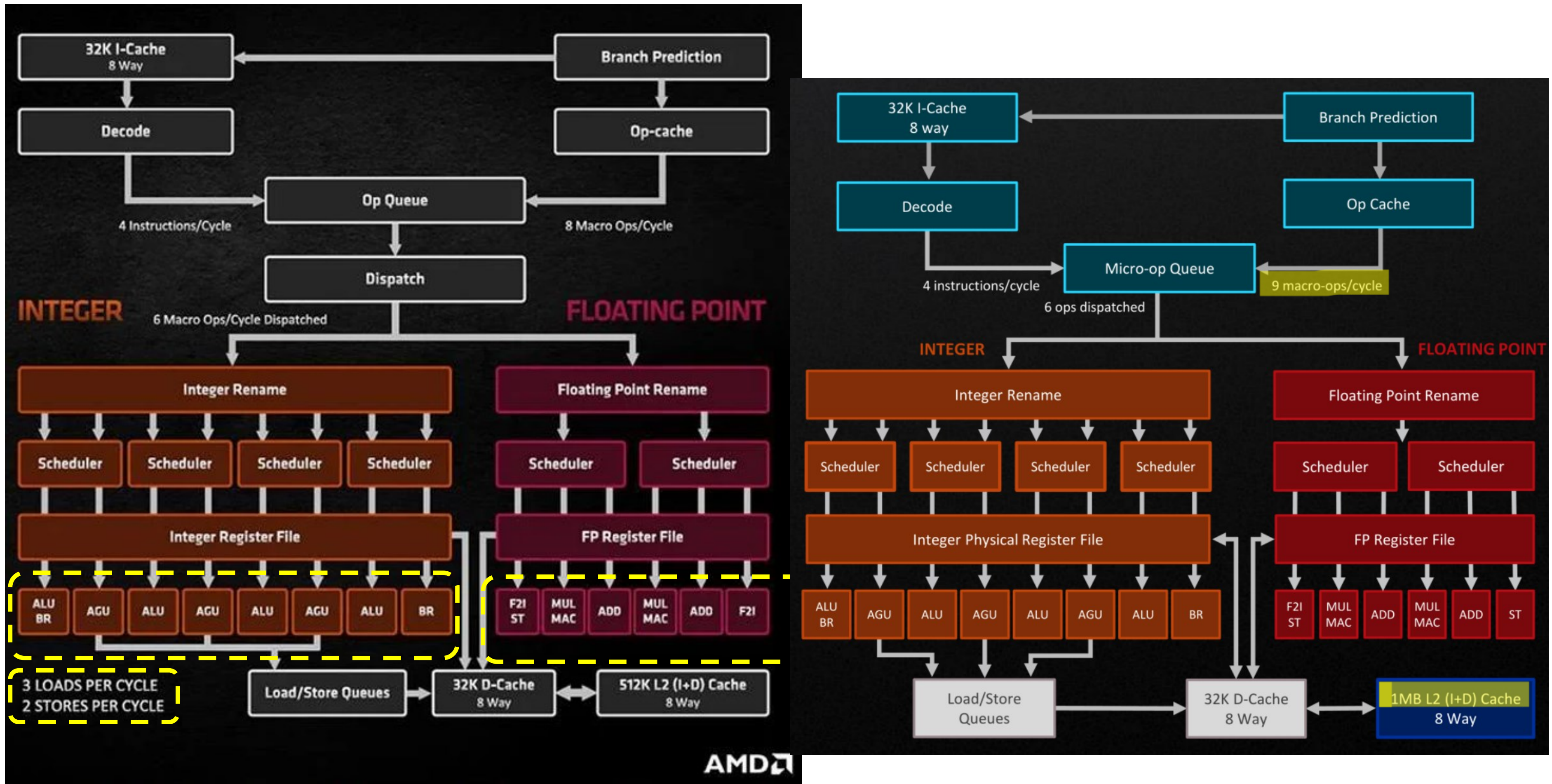
Microarchitecture Overview

- Branch Prediction Improvements
- Larger Op Cache
- Larger Instruction Retire Queue
- Larger Int/FP register file
- Deeper buffers throughout the core
- Power efficient AVX-512 support in the Floating-Point Unit
- Load/Store improvements
- L2 Cache 1M, 8-way



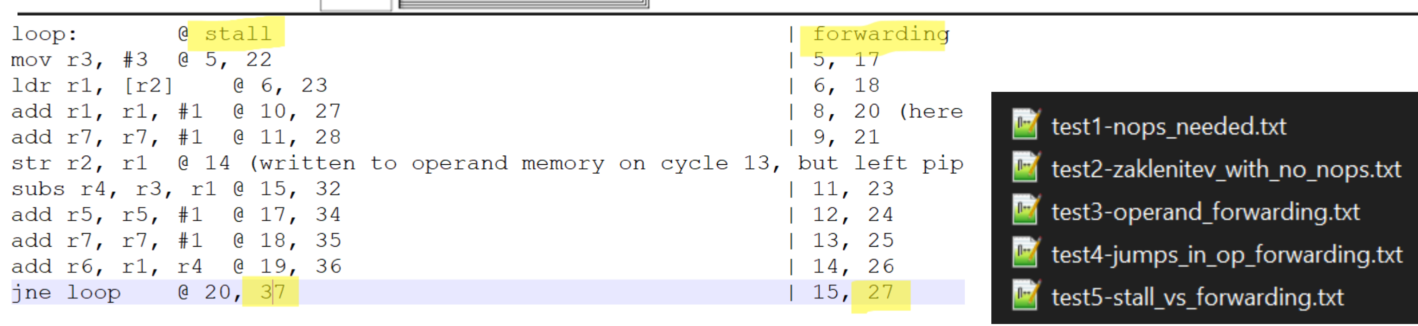
© 2024 Rozman

4.7.5 Zen 3 vs Zen 4 mikroarhitekturi – poenostavljena modela



4.7.6 MiMo v2 . Cevovodne različice

- mimo_32bit_v2.1 - Zaklenitev.circ
- mimo_32bit_v2.2 - Premoscanje.circ
- mimo_32bit_v2.3 - Predikcije.circ
- mimo_32bit_v2.circ



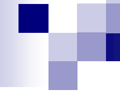
```
loop:         @ stall                                      | forwarding
mov r3, #3   @ 5, 22                                       | 5, 17
ldr r1, [r2]     @ 6, 23                                   | 6, 18
add r1, r1, #1   @ 10, 27                                  | 8, 20 (here
add r7, r7, #1   @ 11, 28                                  | 9, 21
str r2, r1   @ 14 (written to operand memory on cycle 13, but left pip
subs r4, r3, r1 @ 15, 32                                   | 11, 23
add r5, r5, #1   @ 17, 34                                  | 12, 24
add r7, r7, #1   @ 18, 35                                  | 13, 25
add r6, r1, r4   @ 19, 36                                  | 14, 26
jne loop     @ 20, 37                                      | 15, 27
```

- test1-nops_needed.txt
- test2-zaklenitev_with_no_nops.txt
- test3-operand_forwarding.txt
- test4-jumps_in_op_forwarding.txt
- test5-stall_vs_forwarding.txt

- test1-1bit_vs_2bit.txt
- test2-1bit_vs_2bit_nested_loop.txt
- test3-correlating_lht.txt
- test4-tournament.txt
- test5-bubble_sort.txt

https://github.com/LAPSyLAB/MiMo_Student_Release/tree/main/MiMo_v2_Pipelined_versions

© 2024 Rozman



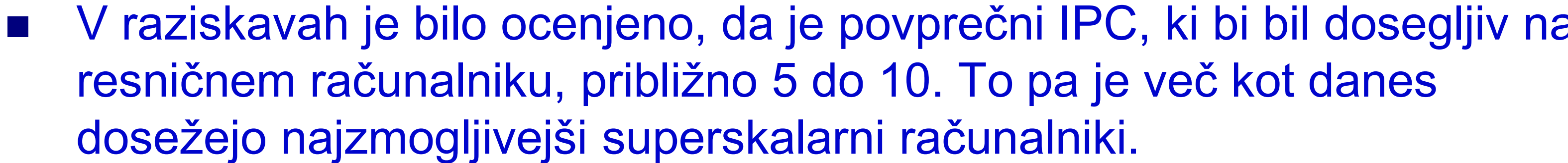
4.8 Omejitve paralelizma na nivoju ukazov

- 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}.

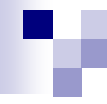
© 2024 Rozman

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,…)

© 2024 Rozman

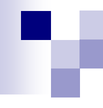


Prelomnica:

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.**

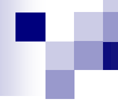
© 2024 Rozman



Prelomnica l.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 (l.2000) ?:
    - 5% več površine prinese 25% povečanje zmogljivosti
    - l. 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

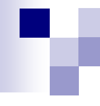
- Večnitnost ni transparentna !

  poskrbi programer (razen program = nit)

- Večnitnost na strojnem nivoju (Hardware Multithreading) omogoča, da si več niti deli funkcijske enote enega procesorja.

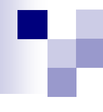
- Vsaka nit v procesorju mora imeti svoje stanje, ki je neodvisno od stanj drugih niti.

© 2024 Rozman



- 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.

© 2024 Rozman



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

© 2024 Rozman

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);
}
```

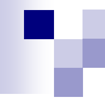
```
'3' - Got '0'
'2' - Got '0'
'1' - Got '0'
'3' - Got '1'
'1' - Got '1'
'2' - Got '1'
'1' - Got '2'
'3' - Got '2'
'2' - Got '2'
'1' - Got '3'
'2' - Got '3'
'3' - Got '3'
'1' - Got '4'
'3' - Got '4'
'2' - Got '4'
'1' - Got '5'
'3' - Got '5'
'1' - Got '6'
'3' - Got '6'
'2' - Got '5'
'1' - Got '7'
```

```
/* 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

 © 2024 Rozman



4.9.1 Drobnozrnata večnitnost

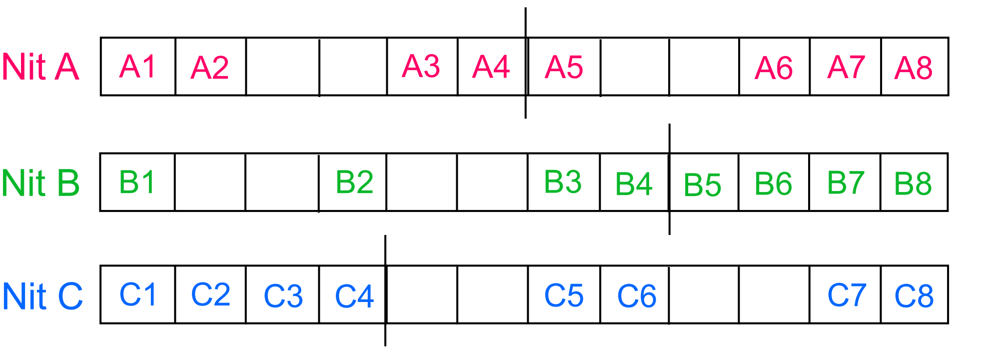
(„fine-grained multithreading“).

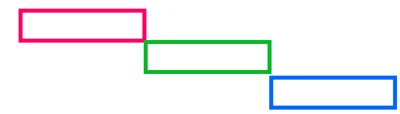
Procesor preklaplja med nitmi vsako urino periodo.

- Niti se preklapljajo 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.

 © 2024 Rozman

Izvajanje treh niti posamezno pri drobnozrnati večnitnosti :





Niti posamezno = 36 t_{CPE}

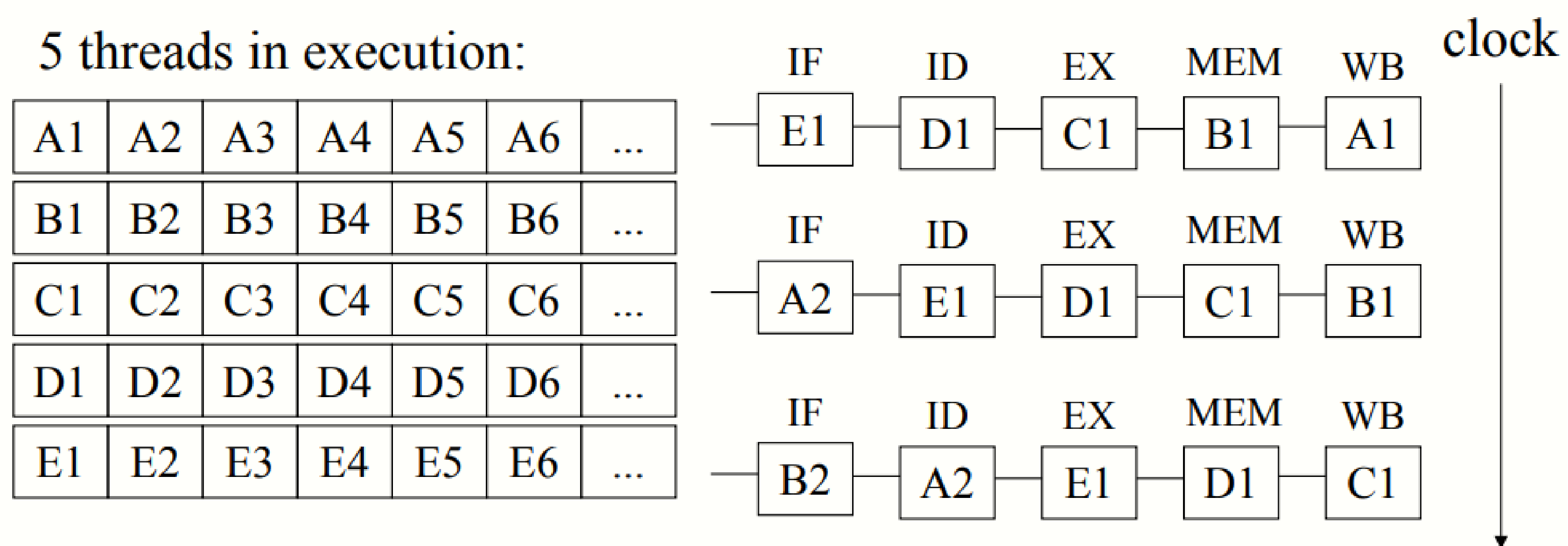
Posamezne niti: **18 t_{CPE}** = 6 (do A4) + 8 (do B4) + 4 (do C4)

Drobnozrnata večnitnost (primer do A4,B4,C4)

Urine periode →

Večnitnost: **12 t_{CPE}**

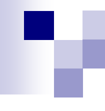
Idealna situacija – 5 stopenjski cevovod in izvajanje petih niti pri drobnozrnati večnitnosti :



Nevarnosti ?

Pogosta situacija ?

© 2024 Rozman



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.

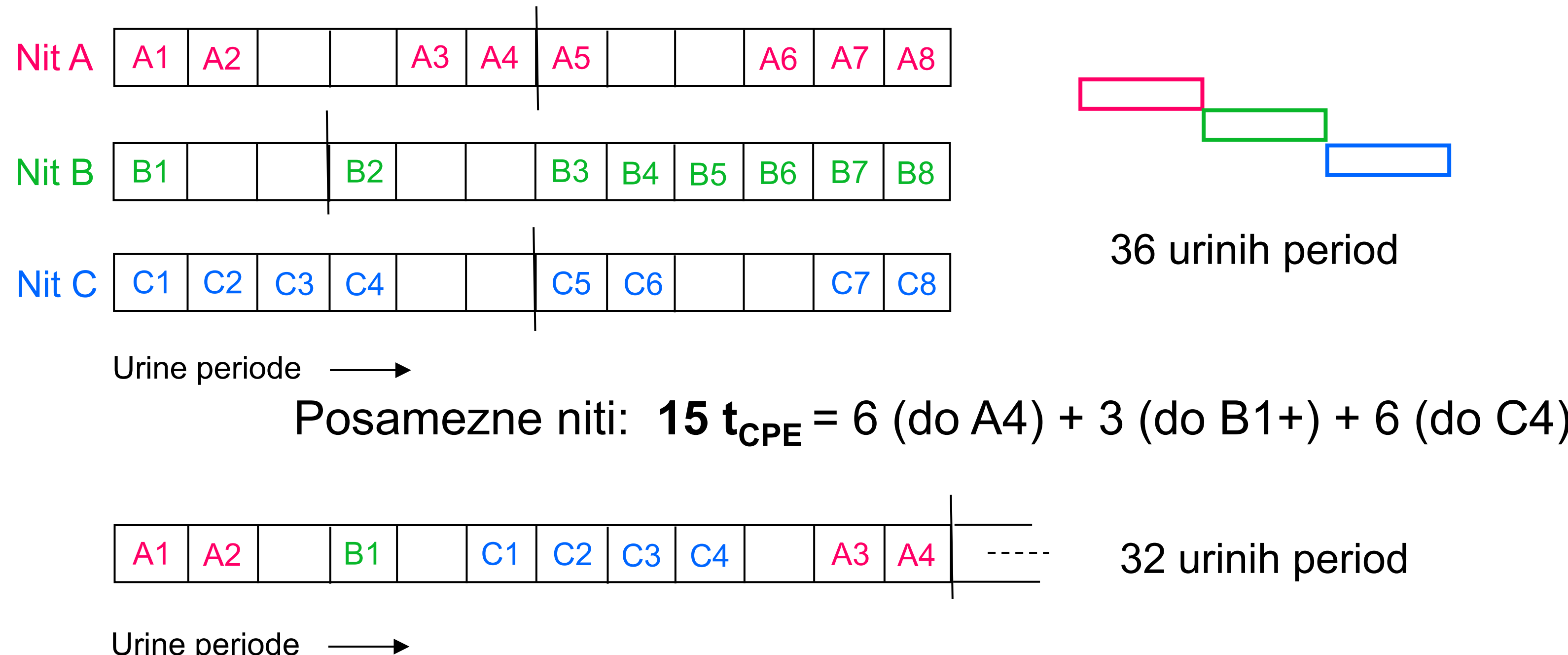
- Procesor je enostavnejši, ker ob preklopu ni treba za vsako nit shraniti celotnega cevovoda, ker lahko počakamo, da se izprazni.
- Preklopov je manj in so zato lahko počasnejši.
- Izvrševanje posamezne niti se ne upočasni, če nima daljših čakanj.

© 2024 Rozman

- 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.

 © 2024 Rozman

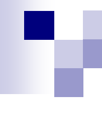
Izvajanje treh niti posamezno in pri grobozrnati večnitnosti



Primer do A4,B1,C4:

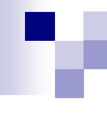
Večnitnost: **12 t_{CPE}**

© 2024 Rozman



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.



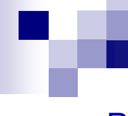
4.9.3 Istočasna večnitnost

(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.

© 2024 Rozman



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

	1											12
Nit A	A1	A2			A3	A4	A5			A6	A7	A8
Nit B	B1			B2			B3	B4	B5	B6	B7	B8
Nit C	C1	C2	C3	C4			C5	C6			C7	C8

Urine periode →

3*12=36 urinih period

Nit A	A1			A3	A5			A6	A8			
	A2			A4				A7				

9 urinih period

Nit B	B1			B2			B3	B5	B7			
							B4	B6	B8			

9 urinih period

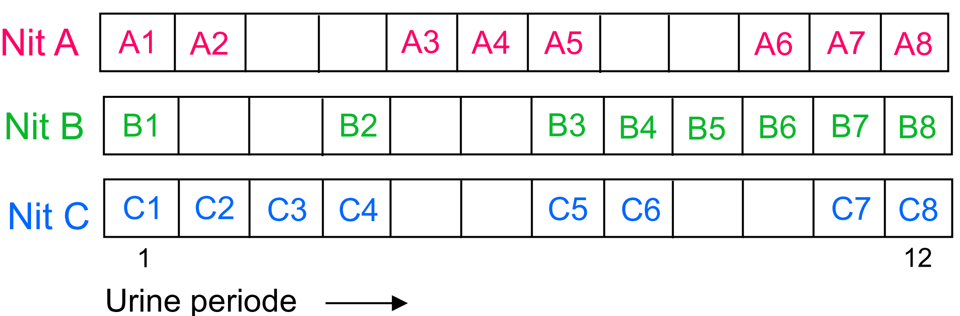
Nit C	C1	C3			C5			C7				
	C2	C4			C6			C8				

8 urinih period

2*9+8=26 urinih period

© 2024 Rozman

Primerjava: večnitnost pri 2-izstavitvenem superskalarnem procesorju

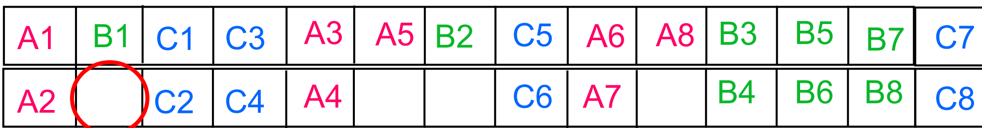


3*12=36 urinih period

Drobnozrnata večnitnost — 14 urinih period

A1	B1	C1	A3	B2	C3	A5	B3	C5	A6	B5	C7	A8	B7
A2		C2	A4		C4		B4	C6	A7	B6	C8		B8

Grobozrnata Večnitnost I *(„preventiva“)* — 14 urinih period



Grobozrnata Večnitnost II *(„daljši zastoj“)* — 22 urinih period

A1		B1		C1	C3		A3	A5		B2		C5		A6	A8		B3	B5	B7		C7
A2				C2	C4		A4					C6		A7			B4	B6	B8		C8

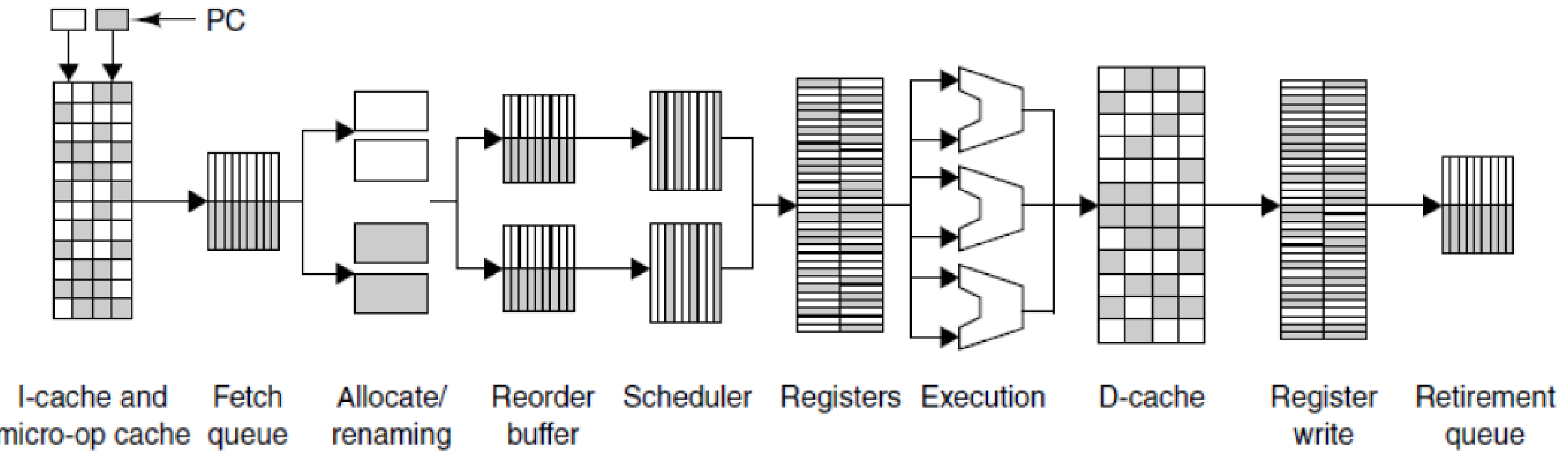
Istočasna večnitnost — 12 urinih period

A1	B1	C2	C4	A4	B2	C6	A7	B3	B5	B7	C7
A2	C1	C3	A3	A5	C5	A6	A8	B4	B6	B8	C8

© 2024 Rozman

- 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

© 2024 Rozman

Istočasna večnitnost ali „Hyperthreading“ na Core i7

Primerjava „SMT“ vs „non-SMT“

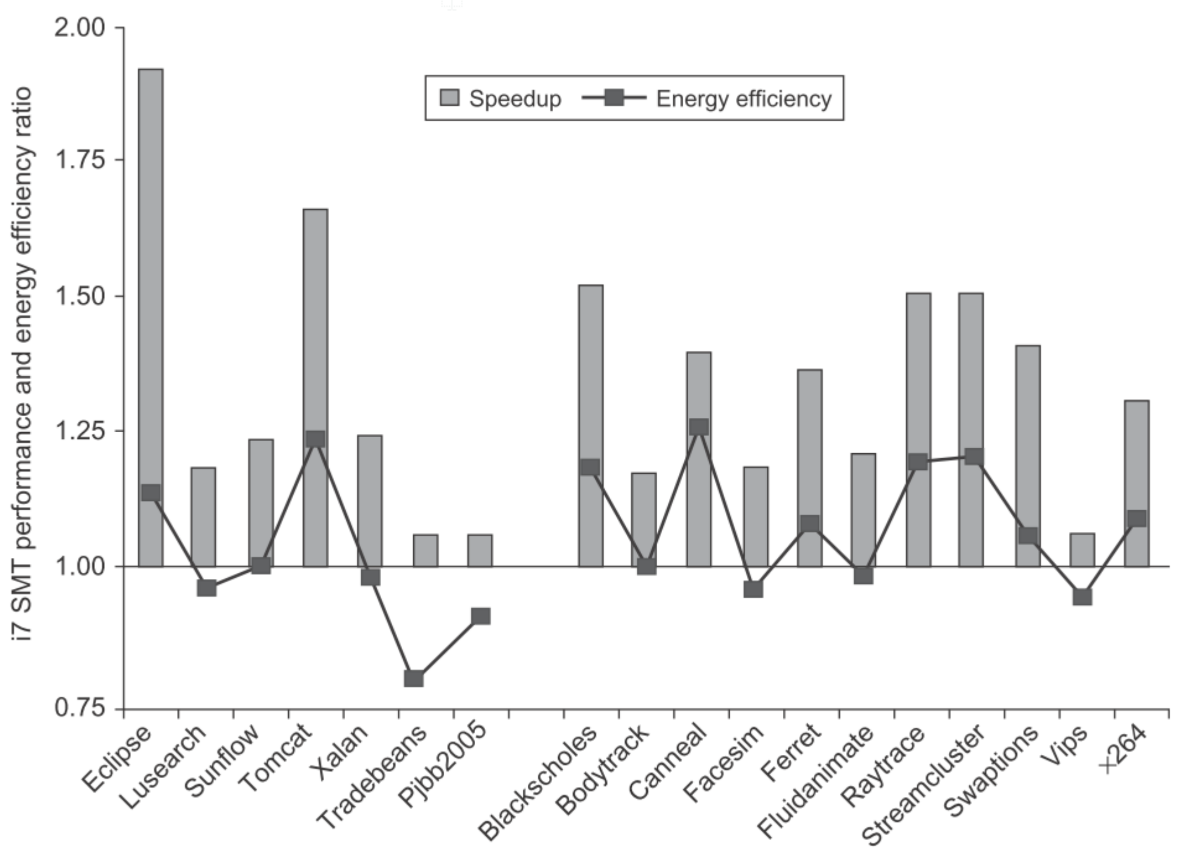
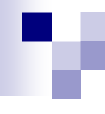


Figure 3.33: The speedup from using multithreading on one core on an i7 processor averages **1.28** for the Java benchmarks and **1.31** for the PARSEC benchmarks



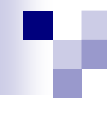
Primer (kaže tudi na slabosti večnitnosti):

- Dve niti, ki za dobro delovanje potrebujeta vsaka ¾ 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

© 2024 Rozman



Intelovi procesorji :

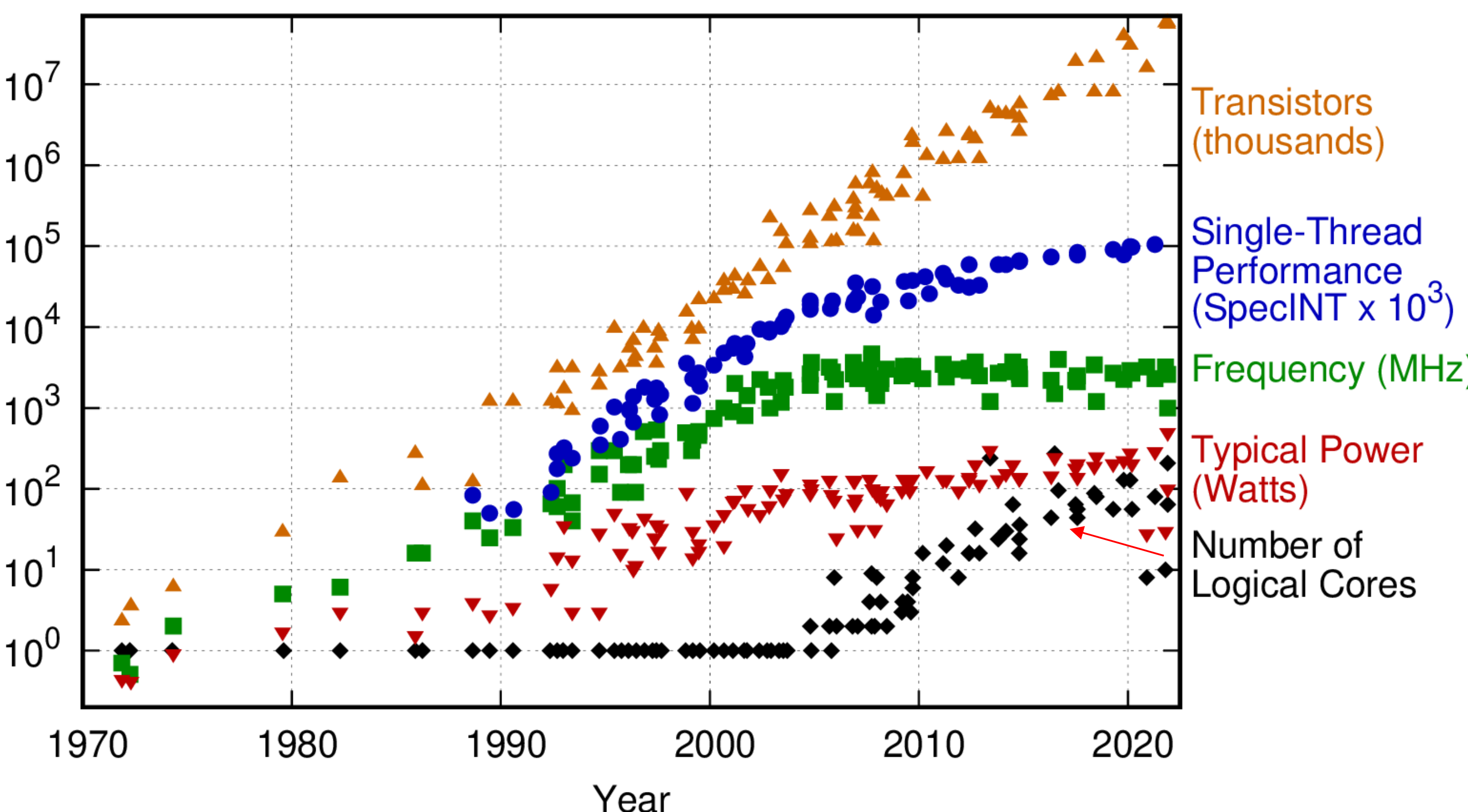
- Intel je prvič uporabil istočasno večnitnost („hyper-threading“) v 2-nitnem procesorju Pentium 4.

- 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).

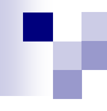
- Pri Core (Intel Core2) mikroarhitekturi je Intel večnitnost opustil, pri mikroarhitekturi Nehalem (Intel Core iX) pa je zopet uvedel istočasno večnitnost z dvema nitma.

© 2024 Rozman

4.9 Pregled dosedanjega razvoja



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



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«