8 PREDPOMNILNIKI

BRANKO ŠTER

OSNOVA PO KNJIGI - DUŠAN KODEK: ARHITEKTURA IN ORGANIZACIJA RAČUNALNIŠKIH SISTEMOV

Lokalnost pomnilniških dostopov

- Tipično je, da programi večkrat uporabijo iste ukaze in operande in da pogosteje uporabljajo ukaze in operande, ki so v pomnilniku blizu trenutno uporabljanim
  - tipičen program 90% časa uporablja samo 10% ukazov
- Lokalnost pomnilniških dostopov močno vpliva na arhitekturo današnjih računalnikov
  - omogoča, da GP zamenjamo s **pomnilniško hierarhijo**

- Štirinivojska pomnilniška hierarhija
  - M_1: predpomnilnik 1. nivoja (SRAM)
  - M_2: predpomnilnik 2. nivoja (SRAM)
  - M_3: GP (DRAM)
  - M_4: pomožni pomnilnik (magnetni disk)
- Pomnilniški prostor nivoja *i* je (v principu) podmnožica prostora na nivoju *i*+1
  - Če informacije ni v M_1, se naredi dostop do M_2; če je tudi v M_2 ni, se naredi dostop do M_3, ...
  - To se izvaja samodejno (ne da bi moral programer skrbeti za to)
- Zaporedje naslovov A(1), ..., A(*N*)
  - pri *N* dostopih do pomnilnika je število različnih naslovov mnogo manjše od *N*

- 2 vrsti lokalnosti:
  1. Prostorska
     - zaporedje ukazov je večinoma na zaporednih lokacijah
     - podatkovne strukture (npr. polja) se običajno obdeluje po zaporednih indeksih
  2. Časovna
     - zanke, začasne spremenljivke

Pomnilniška hierarhija

- Iz glavnega pomnilnika CPE jemlje ukaze in operande in vanj shranjuje rezultate
- Pomembni sta velikost in hitrost
  - velikost, da lahko rešujemo velike probleme
  - hitrost, da CPE ni treba čakati
- Oboje si nasprotuje
  - velik in hiter pomnilnik bi bil zelo drag
- GP: DRAM (dovolj poceni tehnologija za velik pomnilnik)
  - SRAM je predrag za GP

- Hitrost pomnilnikov DRAM se (tekom let) povečuje bistveno počasneje od hitrosti CPE
  - To vrzel je treba nekako premostiti, sicer CPE večino časa čaka na pomnilnik
- Zato se (poleg GP, ki je velik in relativno počasen) uporablja še majhen in hiter pomnilnik, ki mu rečemo **predpomnilnik** (cache)
  - le-ta je narejen v tehnologiji SRAM
- Če bi bili naslovi, ki jih generirajo CPE in V/I naprave, porazdeljeni naključno, ne bi pridobili ničesar
  - ker pa velja princip lokalnosti, se doseže bistveno povečanje hitrosti

Pomožni pomnilnik

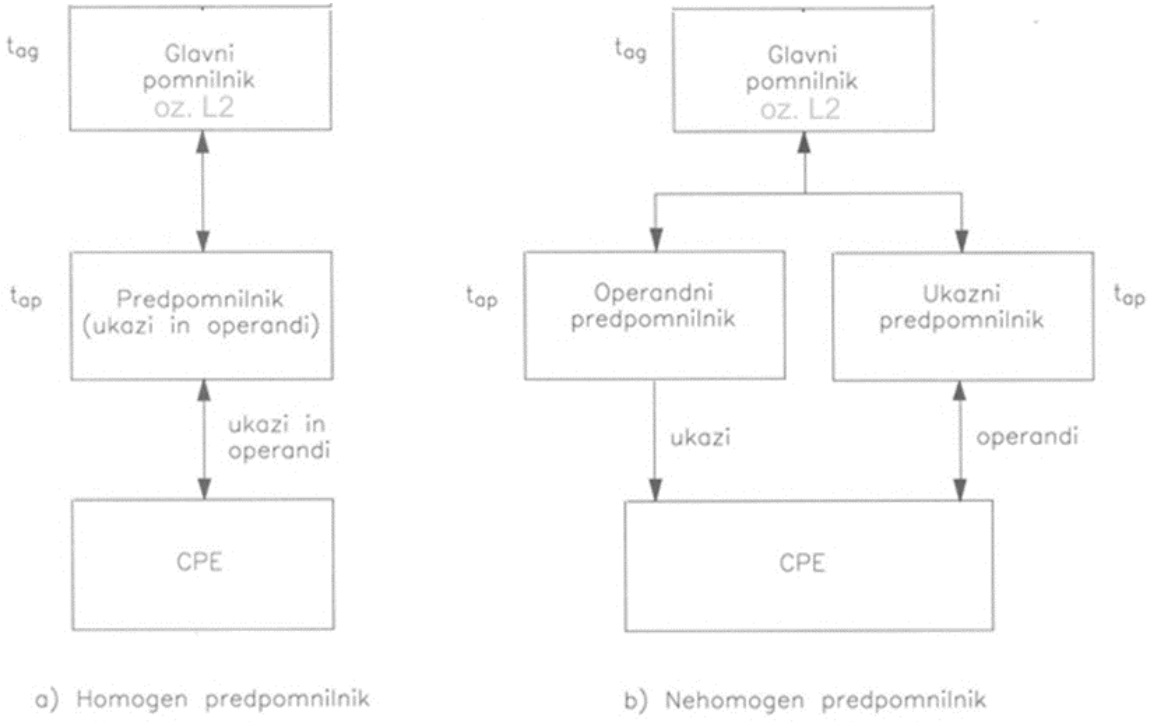
- Pomnilniška hierarhija vključuje tudi *pomožni pomnilnik* (oz. sekundarni, masovni). Kakšna je razlika?
  - Do GP ima CPE **neposreden dostop** (tako, da poda naslov pomnilniške besede)
  - pri pomožnih pomnilnikih je dostop **posreden** preko V/I ukazov, ki najprej prenesejo zahtevano besedo v GP, šele nato je možen neposreden dostop
- Zakaj je potreben pomožni pomnilnik?
  - cena enega bita na magnetnem disku je ~100x nižja kot v GP
  - vsebina je obstojna

- Bit je najmanjša enota informacije
  - shranjen je v eni pomnilniški celici, ki ima lahko 2 stanji (0, 1)
    - nekatere tehnologije sicer uporabljajo več stanj, vendar so manj zanesljive
- Danes so GP izključno elektronski, natančneje polprevodniški (iz integriranih vezij, tj. čipov)

Predpomnilnik

- PP hrani določene podatke, ki so tudi v glavnem pomnilniku
  - vsebina PP je podmnožica vsebine GP
- Pogosto imamo 2 ali 3 nivoje predpomnilnika:
  - L1 (level 1) je manjši in hitrejši in je kar na čipu CPE
  - L2 je malo večji in malo počasnejši (danes običajno tudi na CPE)
  - L3 je večji in počasnejši (običajno ni na CPE)
    - še vedno pa hitrejši od DRAMa

- Pri cevovodnih CPE (ki so danes običajne) je PP (zaradi potrebe po istočasnem dostopu do ukazov in operandov pri cevovodu) razdeljen v dva dela (nehomogeni PP):
  - ukazni in operandni (to velja za L1; L2 pa je običajno homogen)
  - podatkovna pot do PP je širša (128 ali 256 bitov)



- Zadetek: Kadar je naslov, do katerega se želi dostopiti, v PP
- Zgrešitev: sicer
  - v določenih primerih (npr. 2% dostopov) iskane besede ni v PP
  - v tem primeru je treba iz GP v PP prenesti nov blok besed (blok vsebuje iskano besedo), kar traja dolgo

- Vzemimo zaenkrat, da imamo samo L1
  - t_{ap} ... čas dostopa do PP
  - t_{ag} ... čas dostopa do GP
- Razmerje t_{ag}/t_{ap} je lahko tudi do nekaj sto
- Velikost PP je do 1% velikosti GP
  - Kako lahko sploh pričakujemo, da bo iskana informacija dovolj pogosto v PP?
  - Razlog je v lokalnosti

- Uspešnost delovanja PP merimo z **verjetnostjo zadetka** (hit ratio) H
  - Kadar je naslov, do katerega se želi dostopiti, v PP, imamo zadetek, sicer zgrešitev (verjetnost 1-H)
  - H izmerimo s štetjem pomnilniških dostopov, pri katerih pride do zadetka

$$H = N_p/N = N_p/(N_g+N_p)$$

N_p ... število zadetkov

N_g ... število zgrešitev ($=N-N_p$)

  - H je običajno celo večji od 0,95

- Čas dostopa

$$t_a = t_{ap} + (1-H)t_{ag}$$

- Treba pa je upoštevati, da se pri zgrešitvi ne prenese samo beseda, ampak celoten PP blok!
- Zato je bolje uporabiti enačbo

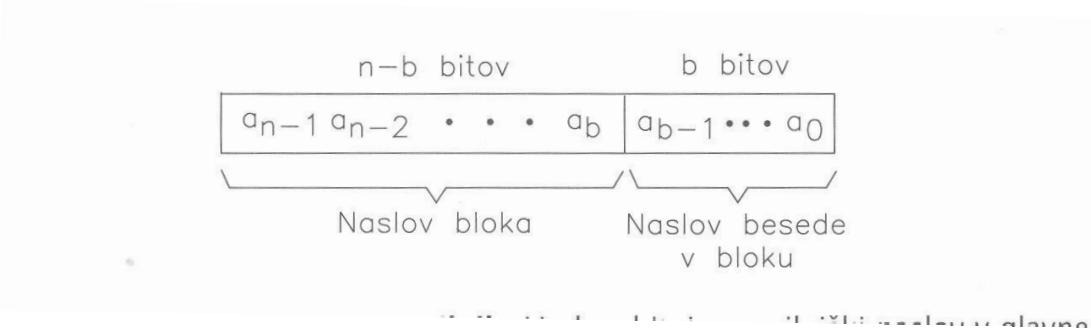
$$t_a = t_{ap} + (1-H)t_B$$

t_B ... čas za prenos bloka oz. **zgrešitvena kazen** (miss penalty) (10-100 urinih period)

Pozor: *miss penalty* ima lahko tudi druge pomene!

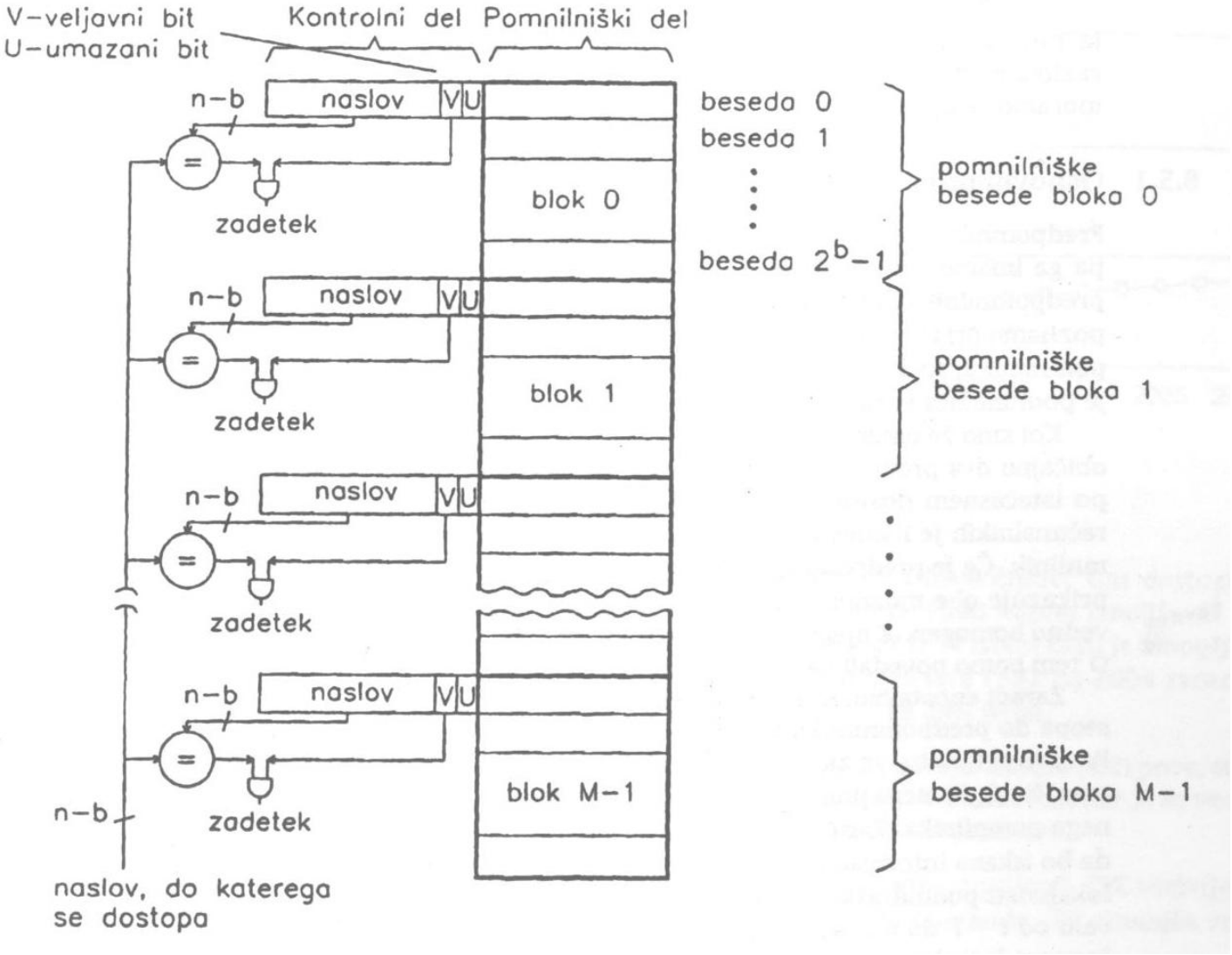
- Možno je definirati področja v GP, katerih besede se nikoli ne prenesejo v PP (*uncacheable* področja)
  - dostop do besede v takem področju je vedno zgrešitev
  - beseda se nikoli ne prenese v PP
  - npr. pri računalnikih, ki uporabljajo pomnilniško preslikan V/I, se določeni pomnilniški naslovi nanašajo na registre V/I naprav
    - pisanje v te registre povzroči odziv naprave

- Ker je vsebina PP podmnožica vsebine GP, mora predpomnilnik (poleg vsebine) vsebovati tudi naslove besed
- Zato je sestavljen iz dveh delov:
  - **kontrolni** in
  - **pomnilniški** del
- Pomnilniški del je razdeljen na **bloke** (po $B=2^b$ pomnilniških besed, b=3-8)
  - bloku se reče tudi **predpomnilniška vrstica** (cache line)
- Pomnilniški naslov:
  - Če je n-biten, rabimo v kontrolnem delu zgornjih n - b bitov naslova
    - spodnjih b bitov določa besedo v bloku, zgornjih n - b bitov pa naslov bloka



- Kontrolni del vsebuje informacijo, ki enolično opiše vsak blok:
  - **naslov bloka** v glavnem pomnilniku (ang. **TAG**)
    - pove, kateri del GP je trenutno v bloku (rečemo, da je *preslikan* v PP)
  - običajno pa še **veljavni** in **umazani bit**
    - veljavni bit pove, ali je vsebina PP veljavna
      - V=1: je
      - V=0: ni $\rightarrow$ zgrešitev
    - umazani bit U se ob prenosu bloka v PP postavi na 0. Če pride do pisanja v blok, se postavi na 1.

Splošna zgradba predpomnilnika



- Naslov je *n*-biten
- Velikost bloka je $B = 2^b$ besed
  - prva beseda v bloku (beseda 0) ima vedno naslov, ki je mnogokratnik dolžine bloka
- Število blokov je M_b

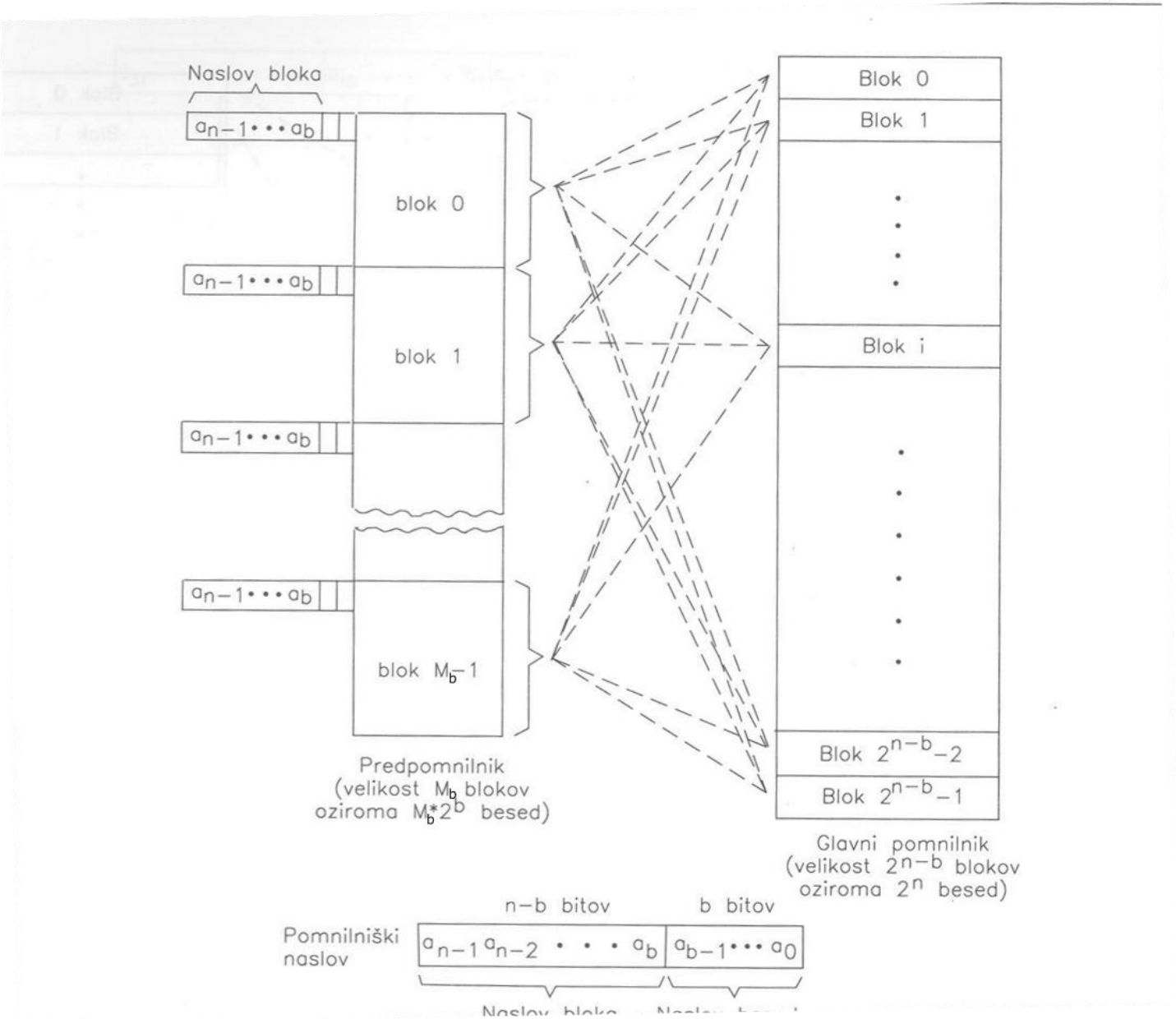
- Zgornjih *n* – *b* bitov naslova se v PP primerja z naslovi v kontrolni informaciji vseh blokov
  - če obstaja pri nekem bloku enakost, je zahtevana beseda v PP (zadetek); poleg tega mora biti še V=1
  - sicer imamo zgrešitev; potreben je dostop do GP; blok iz GP se prenese v PP
    - Če so vsi bloki zasedeni in veljavni, bo novi blok zamenjal enega od obstoječih (**zamenjava bloka**)
    - Ob zamenjavi se mora vsebina bloka, če se je spremenila, najprej prenesti v GP
    - Bit V se uporablja zato, ker včasih vsebina PP ni veljavna

- Primerjava zgornjih $n - b$ bitov naslova z vsebinami kontrolnih delov vseh blokov mora biti zelo hitra
  - zato se uporabljajo omejitve pri preslikavi iz GP v PP: neka beseda iz GP se lahko shrani v vnaprej določeno (majhno) število blokov
  - glede na strogost te omejitve ločimo 3 vrste PP:
    - asociativni
    - set asociativni
    - direktni

Predpomnilniki glede na omejitve pri preslikavi

- Primerjavo zgornjih *n* - *b* bitov lahko izvedemo z **asociativnim pomnilnikom**
  - pri le-tem poteka dostop preko vsebine
  - če damo na vhod kombinacijo bitov, se ta primerja z delom vsebine vsake besede
  - v primeru enakosti vrne celotno besedo
  - takemu PP rečemo asociativni PP (APP)
- Vsebina AP so naslovi v kontrolnem delu
  - kontrolni del je v bistvu kar AP
- Pri zadetku se nato naredi dostop do besede v bloku (določena z *b* biti)
- To je **čisti APP** - ni omejitev:
  - vsak blok PP lahko sprejme katerokoli besedo iz GP
  - čisti APP ima največji H
- Problem pa je v tem, da so veliki AP izjemno dragi
  - prav velikih pravzaprav sploh ni

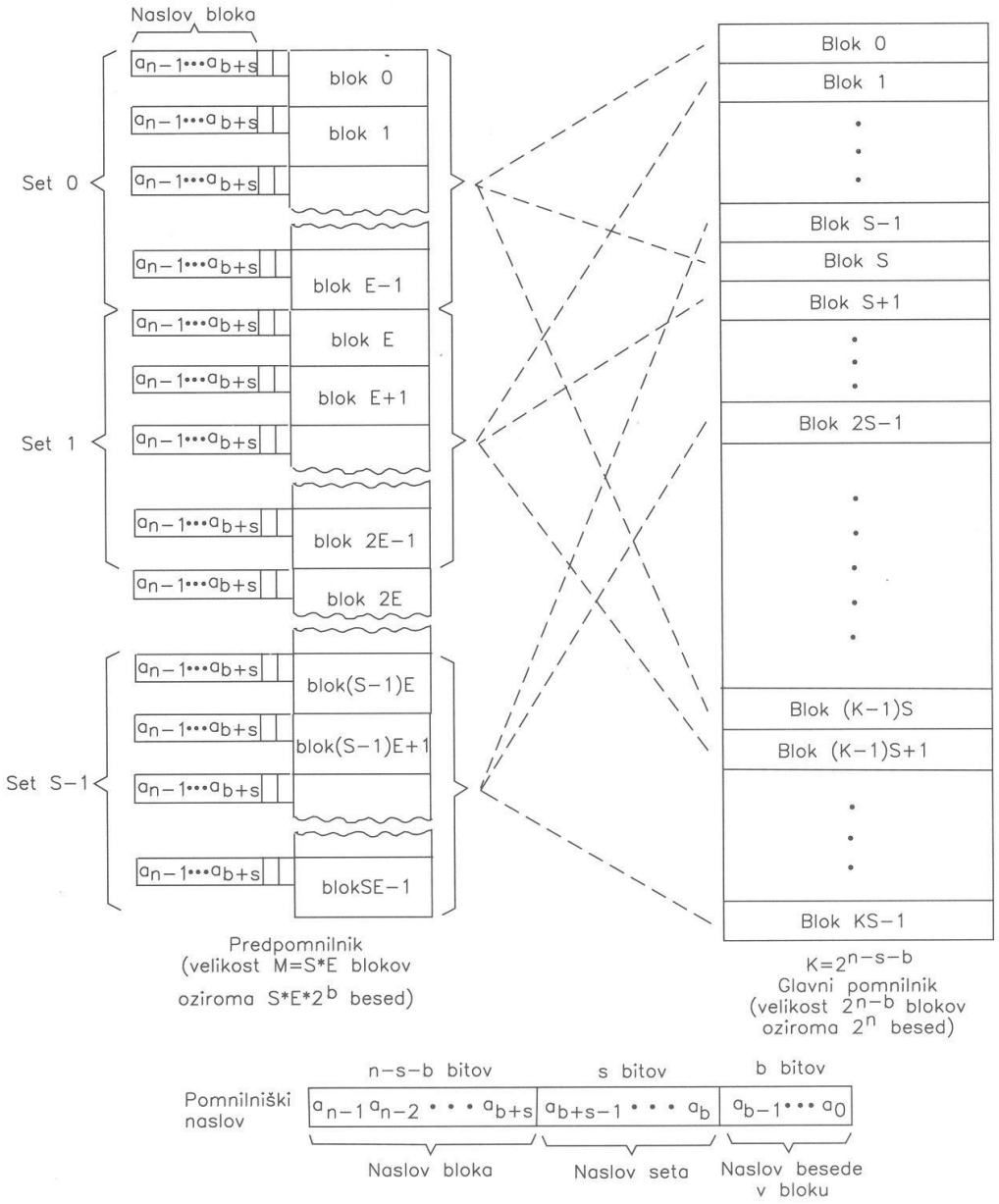
Čisti APP


Naslov bloka
$a_{n-1} \cdots a_b$
blok 0
blok 1
blok $M_b - 1$
Predpomnilnik
(velikost M_b blokov
oziroma $M_b * 2^b$ besed)
Blok 0
Blok 1
Blok i
Blok $2^{n-b}-2$
Blok $2^{n-b}-1$
Glavni pomnilnik
(velikost 2^{n-b} blokov
oziroma 2^n besed)
n−b bitov
b bitov
Pomnilniški naslov
$a_{n-1}\, a_{n-2} \cdots a_b$
$a_{b-1} \cdots a_0$

- potrebno je veliko število primerjalnikov
  - npr. za PP s 100000 bloki bi potrebovali 100000 (*n*-*b*)-bitnih primerjalnikov (ogromno logike)

- Velik PP lahko naredimo le, če v preslikovanje naslovov vpeljemo omejitve
- Namesto enega velikega AP uporabimo več majhnih
- Tako dobimo **set-asociativni predpomnilnik** (SAPP)

SAPP


Naslov bloka
Set 0
Set 1
Set S−1
blok 0
blok 1
blok E−1
blok E
blok E+1
blok 2E−1
blok 2E
blok(S−1)E
blok(S−1)E+1
blokSE−1
Predpomnilnik
(velikost M=S*E blokov
oziroma S*E*2^b besed)
Blok 0
Blok 1
Blok S−1
Blok S
Blok S+1
Blok 2S−1
Blok (K−1)S
Blok (K−1)S+1
Blok KS−1
K=2^(n−s−b)
Glavni pomnilnik
(velikost 2^(n−b) blokov
oziroma 2^n besed)
n−s−b bitov
s bitov
b bitov
Pomnilniški naslov
Naslov bloka
Naslov seta
Naslov besede v bloku

- SAPP je razdeljen na $S = 2^s$ setov, vsak set pa je majhen AP
- Število blokov v setu $E = 2^e$ je **stopnja asociativnosti** (običajno do 16)
  - to je velikost AP v setu (= št. primerjalnikov)
- Velikost PP je

  $M_b = S*E = 2^{s+e}$ blokov oz.

  $M = M_b*B = S*E*B = 2^{s+e+b}$ pomnilniških besed

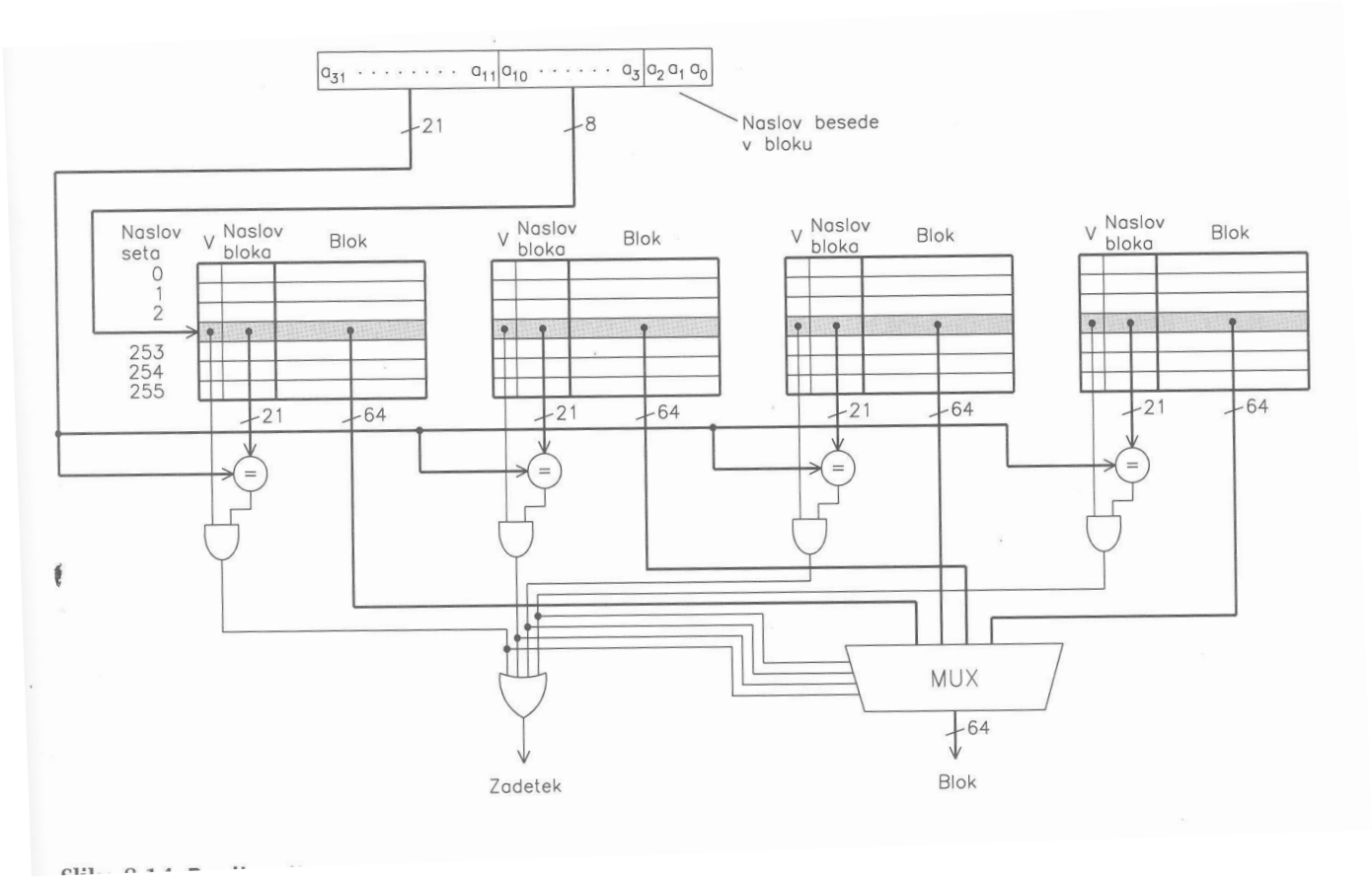
- Pri SAPP se pojavi omejitev pri preslikovanju naslovov:
  - za vsako besedo GP je vnaprej določeno, v katerega od setov se lahko preslika
    - to določajo naslovni biti $a_{b+s-1}, a_{b+s-2}, \ldots, a_b$
    - ta naslov seta (SET) se imenuje tudi **predpomnilniški indeks** (cache index, CI)
      - če so ti biti 0,0,...,0, se lahko preslika v set 0
      - če so ti biti 0,0,...,1, se lahko preslika v set 1
      - itd.
    - naslov A_i se torej lahko preslika le v enega od blokov seta S_i

$$S_i = A_i(n-1 : b) \bmod 2^s$$

- Pri SAPP lahko s spreminjanjem *E* vplivamo na njegove lastnosti
  - pri *S* = 1 (*s* = 0) je *E* enako *M* (čisti APP)
  - pri *E* = 1 (*e* = 0) je v vsakem setu le en blok (**Direktni PP**)
    - pri tem je torej za vsako besedo vnaprej določeno, v kateri blok se preslika
      - blok enak setu
    - potreben samo 1 primerjalnik

Realizacija SAPP

- n=32, b=3, e=2, s=8
- Vsak izmed 256 setov ima 4 bloke, vsak blok 8 bajtov (=8x8=64 bitov)
- $M = 2^{13} = 8$ KB

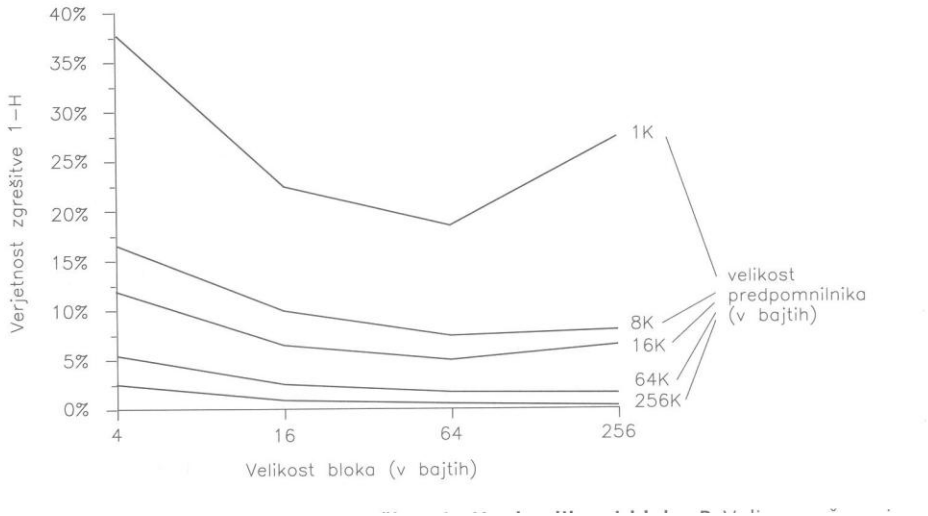


- Pri podani velikosti PP (M) se z E spreminja tudi H
  - manjši E $\rightarrow$ manjša cena in tudi manjši H

- *Predpomnilniško pravilo 2:1*
  - (1-H) direktnega PP velikosti M $\approx$ (1-H) SAPP z E=2 velikosti M/2
  - izkustveno

Kako velik naj bo blok?

- PP lahko povečujemo s povečevanjem E, S ali B
  - najlažje B
  - vsak blok ima eno kontrolno informacijo, kontrolni del pa je najbolj zapleten
- Pri dani velikosti PP:
  - če povečamo bloke, je boljša prostorska lokalnost, toda slabša časovna lokalnost, ker je blokov manj
  - 1-H se najprej zmanjšuje, nato pa začne naraščati



- toda 1-H ni edini parameter, ki vpliva na delovanje PP
  - pomembna je tudi zgrešitvena kazen t_B (čas prenosa bloka v PP)
  - sestavljena je iz latence in časa dejanskega prenašanja
  - pri večjem bloku je t_B večja
    - od nekod naprej lahko prevlada nad zmanjšanjem 1-H in poslabša delovanje PP
- t_B se lahko zmanjša tako, da se najprej prenese zahtevana beseda (CPE lahko takoj nadaljuje z delom), nato ostale (*requested word first*)

Kateri blok naj se zamenja ob zgrešitvi?

- tudi to vpliva na 1-H
- 2 strategiji:
  1. Naključna.
     - enostavna za realizacijo
  2. LRU (Least Recently Used)
     - zamenja se blok, do katerega najdalj ni bil narejen dostop
     - izkoriščanje časovne lokalnosti
     - pri E > 4 zapletena realizacija

- pri večjih E naključna strategija
- pri E = 2 je 1-H pri naključni strategiji ~1,1x večja kot pri LRU

Vpliv E in zamenjevalne strategije na 1-H (pri večjem PP sta oba vpliva manjša):

M	1-H					
	E = 2		*E* = 4		*E* = 8	
	LRU	Naključno	LRU	Naključno	LRU	Naključno
16K	5,2%	5,7%	4,7%	5,3%	4,4%	5,0%
64K	1,9%	2,0%	1,5%	1,7%	1,4%	1,4%
256K	1,2%	1,2%	1,1%	1,1%	1,1%	1,1%

Pisanje

- branje je enostavnejše (in tudi bolj pogosto)
- pisanje v PP se lahko začne le, če je bil ugotovljen zadetek
- Kako se sprememba v PP odraža v GP:
  1. **Pisanje skozi** (*write through*)
     - vedno se piše v oba
  2. **Pisanje nazaj** (*write back*)
     - piše se samo v PP
     - pri zamenjavi je spremenjeni blok treba prenesti v GP
     - **umazani bit** (dirty bit) je 0 ob prenosu bloka v PP. Po pisanju v blok se postavi na 1.
     - pri zamenjavi se zapišejo v GP samo bloki z 1

- Pisanje nazaj:
  - hitrost
  - manj prometa z GP
  - ob zamenjavi bloka najhitrejši način pisanja v DRAM
- Pisanje skozi:
  - enostavno za realizacijo
  - vsebini PP in GP sta **skladni** (**koherentni**)
    - dobro za druge naprave

- **Pisalni izravnalnik** (write buffer)
  - vanj CPE shrani podatek, ki se bo (s pomočjo dodatne logike) vpisal v GP
  - s tem se odpravi čakanje zaradi hitrejšega pisanja v PP kot v GP
  - pri *pisanju skozi* je nujno potreben
- Danes se uporablja pretežno pisanje nazaj
  - uporablja se tudi pisalni izravnalnik
  - podoben je čistemu APP
  - pri pisanju CPE vedno preveri, če je beseda v enem od blokov v izravnalniku
  - umazani blok se piše v pisalni izravnalnik namesto direktno v GP
- Pri pisanju tudi pri zadetkih rabimo 2 periodi
  - najprej je potrebno branje
  - v bistvu je možna tudi le 1 perioda
    - na osnovi neke vrste cevovoda (več v knjigi)

- Zgrešitve
  - pri bralnih zgrešitvah se blok vedno prenese v PP (zamenja enega od obstoječih)
  - pri pisalnih zgrešitvah 2 možnosti:
    1. **Pisalna zamenjava** (write allocate)
       - prenos novega bloka v PP (podobno kot bri branju)
       - bolj običajno pri pisanju nazaj
       - bolj razširjena
    2. **Pisanje naokrog** (write around, no write allocate)
       - zamenjava bloka samo v GP (ne v PP)
       - bolj običajno pri pisanju skozi

Vrste zgrešitev

- **Vrste zgrešitev**
  1. **Obvezne zgrešitve** (compulsory misses), OZ
     - reče se tudi zgrešitve prvega dostopa
     - *kadar pride blok prvič v PP, je vedno obvezna zgrešitev*
  2. **Velikostne zgrešitve** (capacity misses), VZ
     - zaradi končne velikosti PP običajno ne more vsebovati vseh blokov, ki jih program potrebuje, zato prihaja do zamenjav blokov, ki so kmalu spet potrebni
  3. **Konfliktne zgrešitve** (conflict misses), KZ
     - zamenjava blokov, ki se preslikajo v isti set
     - pri čistem APP jih ni
     - *Konfliktna zgrešitev je tista, do katere ne bi prišlo, če bi bil PP čisti APP z zamenjevalno strategijo LRU*
     - *Npr., direktni PP s 4 seti dostopa do blokov:*
       - *0(OZ), 1(OZ), 2(OZ), 3(OZ), 4(OZ), 1, 2, 3, 0(VZ), 4(VZ), 0(KZ)*

Vrste zgrešitev glede na M in E (Alpha, B=64, LRU, SPEC2000):

Velikost predpomnilnika v bajtih	Stopnja asociativnosti *E*	Skupna verjetnost zgrešitve $1-H$	Deleži posameznih vrst (vsota = skupna verjetnost zgrešitve)					
			Obvezna zgrešitev		Velikostna zgrešitev		Konfliktna zgrešitev	
1K	1	0,191	0,009	5%	0,141	73%	0,042	22%
1K	2	0,161	0,009	6%	0,141	87%	0,012	7%
1K	4	0,152	0,009	6%	0,141	92%	0,003	2%
1K	8	0,149	0,009	6%	0,141	94%	0,000	0%
2K	1	0,148	0,009	6%	0,103	70%	0,036	24%
2K	2	0,122	0.009	7%	0,103	84%	0,010	8%
2K	4	0,115	0,009	8%	0,103	90%	0,003	2%
2K	8	0,113	0,009	8%	0,103	91%	0,001	1%
4K	1	0,109	0,009	8%	0,073	67%	0,027	25%
4K	2	0,095	0,009	9%	0,073	77%	0,013	14%
4K	4	0,087	0,009	10%	0,073	84%	0,005	6%
4K	8	0,084	0,009	11%	0,073	87%	0,002	3%
8K	1	0,087	0,009	10%	0,052	60%	0,026	30%
8K	2	0,069	0,009	13%	0,052	75%	0,008	12%
8K	4	0,065	0,009	14%	0,052	80%	0,004	6%
8K	8	0,063	0,009	14%	0,052	83%	0,002	3%
16K	1	0,066	0,009	14%	0,038	57%	0.019	29%
16K	2	0,054	0,009	17%	0,038	70%	0,007	13%
16K	4	0,049	0,009	18%	0,038	76%	0,003	6%
16K	8	0,048	0,009	19%	0,038	78%	0,001	3%
32K	1	0,050	0,009	18%	0,028	55%	0,013	27%
32K	2	0,041	0,009	22%	0,028	68%	0,004	11%
32K	4	0,038	0,009	23%	0,028	73%	0,001	4%
32K	8	0,038	0,009	24%	0,028	74%	0,001	2%
64K	1	0.039	0,009	23%	0,019	50%	0,011	27%
64K	2	0,030	0,009	30%	0,019	65%	0,002	5%
64K	4	0,028	0,009	32%	0,019	68%	0,000	0%
64K	8	0,028	0,009	32%	0,019	68%	0,000	0%
128K	1	0,026	0,009	34%	0,004	16%	0,013	50%
128K	2	0.020	0,009	46%	0,004	21%	0,006	33%
128K	4	0,016	0,009	55%	0,004	25%	0,003	20%
128K	8	0,015	0,009	59%	0,004	27%	0,002	14%

Rezultati

- Rezultati:
  - kaj opazimo za vsako od vrst zgrešitev?
    - pogostost obveznih neodvisna od M
      - delež le-teh zelo majhen, če se je program dolg
    - pogostost velikostnih pada z M
    - pogostost konfliktnih pada z E
- Kako bi zmanjšali vsako od 3 vrst zgrešitev:
  - obvezne: večji blok
    - vendar se lahko poveča zgrešitvena kazen
  - velikostne: večji PP
  - konfliktne: večji E
    - vendar se lahko poveča čas dostopa

Skladnost

- Problem **skladnosti PP** (cache coherency): vsebina bloka v PP se lahko razlikuje od vsebine v GP ali v drugih PP
  - treba je zagotoviti, da zaradi neskladnosti ne pride do napak
- En vzrok za neskladnost so prenosi med V/I napravami in GP
- Neskladnost pa se pojavlja tudi na računalnikih, ki imajo več CPE (oz. CPE jeder)

Primer

- Imamo procesor z ločenima UPP in OPP. Pomnilniška beseda je velikosti 1B(bajt). Pri delovanju nekega programa je procesor zaporedoma dostopal do podanih naslovov:
  - **0x180, 0x1C8, 0x0B1, 0x188, 0x142, 0x084, 0x247, 0x1C4, 0x246, 0x140.**

- OPP je SAPP
  - M = 256 bajtov,
  - B = 16 bajtov,
  - stopnja asociativnosti E = 2
  - zamenjalna strategija LRU

- Za vsak naslov določimo:
  - naslov bloka v GP ((n-b)-bitni)
  - št. (naslov) seta, v katerega se blok preslika
  - ali pride do zadetka ali zgrešitve (in katerega tipa je)
  - kateri blok v setu se zapolni ali zamenja (če pride do tega)

- Rešitev:

  M = SEB,

  S = M/EB = 256 /(2*16) = $2^8/2^5 = 2^3$ = 8 setov

  E = 2 bloka / set

Naslov hex	Naslov bin	Št. bloka v GP	Tag (oznaka)	Naslov seta (CI - cache index)	Blok v setu	Stanje seta CI (oznake blokov)	Zgrešitev / zadetek
0x180	0001 1000 0000	24(=3*8+0)	3	0	b0	3,-	OZ
0x1C8	0001 1100 1000	28	3	4	b0	3,-	OZ
0x0B1	0000 1011 0001	11	1	3	b0	1,-	OZ
0x188	0001 1000 1000	24	3	0	b0	3,-	Zadetek
0x142	0001 0100 0010	20	2	4	b1	3,2	OZ
0x084	0000 1000 0100	8	1	0	b1	3,1	OZ
0x247	0010 0100 0111	36	4	4	b0	4,2	OZ
0x1C4	0001 1100 0100	28	3	4	b1	4,3	KZ
0x246	0010 0100 0110	36	4	4	b0	4,3	Zadetek
0x140	0001 0100 0000	20	2	4	b0	2,3	KZ

- 8 zgrešitev (6 obveznih, 2 konfliktni) in 2 zadetka
  - H = 2/10 = 0,2 oz. 20%
  - Verjetnost zadetka pri predpomnilnikih v praksi je seveda večja (naš primer je karikiran, zaradi ilustracije)

Vpliv PP na hitrost delovanja CPE

- Čas izvrševanja programa:

$$CPEčas = (CPEperiode_{izvrš.} + CPEperiode_{čak.})*t_{CPE}$$

$$CPEperiode_{izvrš} = I*CPI_{idealni}$$

$$CPEperiode_{čak} = N*(1-H)*K_Z$$

$$N = I*(1+M_I)$$

$$CPEčas = I*(CPI_{idealni} + M_I*(1-H)*K_Z)*t_{CPE}$$

N_R ... število bralnih dostopov

N_W ... število pisalnih dostopov

N ... število vseh pom. dostopov

I ... število ukazov

H ... povprečna verjetnost zadetka

K_Z … povprečna zgrešitvena kazen

M_I ... povprečno število operandnih pomnilniških dostopov na ukaz

$CPI_{idealni}$ predpostavi, da ni zgrešitev

- Če $H_R \neq H_W$:

  $CPEperiode_{čak} = N_R*(1-H_R)*K_{Z,R} + N_W*(1-H_W)*K_{Z,W}$

- Če $H_{UPP} \neq H_{OPP}$:

  Število zgr.: $N*(1-H) \rightarrow I*(1-H_{UPP}) + I*M_I*(1-H_{OPP})$

  Verj. zgr.: $1-H = 1-H_{UPP} + M_I*(1-H_{OPP})$

  $CPEperiode_{čak} = (I*(1-H_{UPP}) + I*M_I*(1-H_{OPP}))*K_z$

Primer 1

- f_{CPE} = 300 MHz
- ločen ukazni in operandni PP
- $CPI_{idealni}$ = 2 (izmerjen na nekem programu)
- 36% pomnilniških dostopov na ukaz (pri tem programu)
- verjetnost zgrešitve v ukaznem PP = 2%
- verjetnost zgrešitve v operandnem PP = 4%
- DRAM: prvi dostop 60ns, naslednji trije po 10ns
- PP: 256-bitni blok, 64-bitna podatkovna pot do DRAMa
- Za koliko zgrešitve upočasnijo delovanje računalnika?

Prenos bloka zahteva 4 64-bitne prenose, torej 90ns

- zgrešitvena kazen torej 27 period ure

$CPEperiode_{čak,UPP} = I*(1-H) * K_Z = I*0,02*27 = 0,54*I$

$CPEperiode_{čak,OPP} = I*M_I*(1-H) * K_Z = I*0,36*0,04*27 = 0,39*I$

Skupno je čakalnih period 0,93*I

$CPI = 2,93$

$Upočasnitev = CPI/CPI_{idealni} = 2,93/2 = 1,47$

Primer2

- f_{CPE} = 600MHz
  - hitrost obeh PP ustrezno večja
- ostalo enako

$CPEperiode_{čak,UPP} + CPEperiode_{čak,OPP}$

= I*0,02*54 + I*0,36*0,04*54 = 1,86*I

CPI = 3,86

Upočasnitev = $CPI/CPI_{idealni}$ = 3,86/2 = 1,93

$CPI_1 * t_{CPE1} / CPI_2 * t_{CPE2}$ = 2,93*2 / 3,86 = 1,52

Računalnik z 600MHz uro je (v našem primeru) le 1,52 krat hitrejši od tistega z 300MHz (zaradi PP)!

- Škoda zaradi zgrešitev se povečuje
  - s f_{CPE}
    - zgrešitvena kazen se meri v številu period ure
  - pa tudi z zmanjšanjem CPI
    - npr. zaradi povečane paralelnosti
- Zgrešitveno kazen lahko zmanjšamo tudi z uvedbo L2

$$CPEperiode_{čak} = N*(1-H_{L1}) * K_{Z,L1}$$

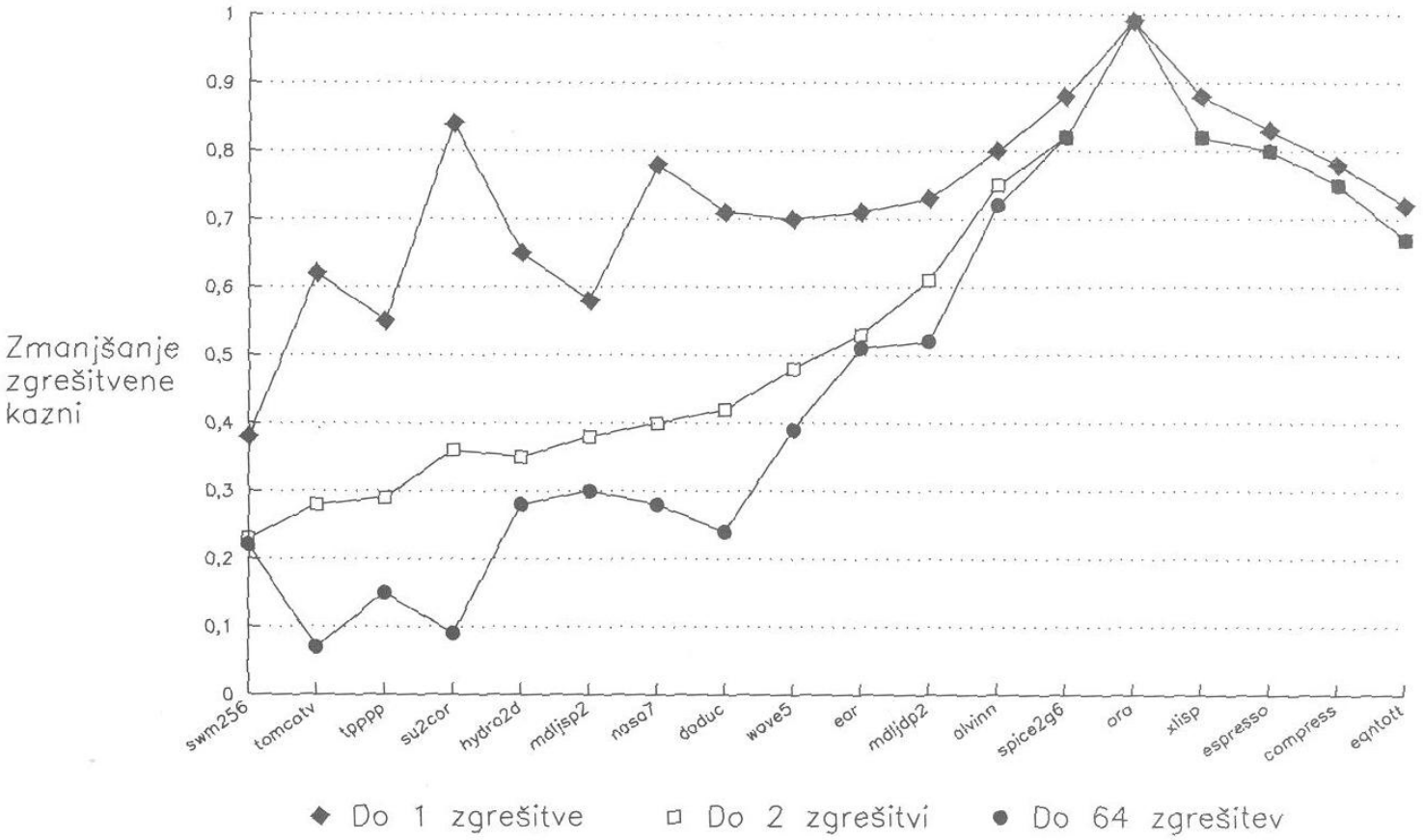
$$K_{Z,L1} = t_{B,L2} + (1-H_{L2}) * K_{Z,L2}$$

- $t_{B,L2}$ ... čas prenosa bloka iz L2 v L1 pri zadetku v L2
- H_{L2} je lokalna verjetnost zgrešitve (pogojna verjetnost, pogoj je zgrešitev v L1)
  - Do L2 se dostopa, kadar je v L1 zgrešitev
- Globalna verjetnost zgrešitve na nivoju L2 je $(1-H_{L1})*(1-H_{L2})$

Načini za zmanjševanje zgrešitvene kazni

- Na izgubo hitrosti vplivata 1-H in t_B
  - zmanjšanje 1-H: večji PP, večji E, ustrezen B, dobra zamenjevalna strategija
  - zmanjšanje t_B: vrstni red prenosa besed v bloku, L2
  - so pa danes tudi druge možnosti:
    1. **Vnaprejšnji prevzem bloka** (block prefetch)
       - pri prenosu bloka k v PP se prebereta še npr. bloka k+1 in k+2 in shranita v **bralni izravnalnik** (read buffer), do katerega se da hitro dostopiti
    1. **Neblokirajoči PP** (nonblocking cache)
       - med zamenjavo bloka PP deluje naprej in CPE lahko špekulativno izvršuje naslednje ukaze
       - načinu delovanja se reče *zadetek pod zgrešitvijo* (hit under miss)
       - možno je tudi *zadetek pod večkratno zgrešitvijo* (hit under multiple miss)

Razmerje zgrešitvene kazni neblokirajočega in blokirajočega PP:



Primer. Oglejmo si na praktičnem primeru, kako način dostopa do pomnilnika in predpomnilnikov lahko vpliva na CPEčas.
Množimo matriki **A** in **B**, obe velikosti 1000 x 1000 double (dvojna natančnost v plavajoči vejici), rezultat je **C**. Do elementov matrik dostopamo na dva različna načina.

```
#define N 1000

double **A = new double *[N];
double **B = new double *[N];
double **C = new double *[N];

for (i = 0; i < N; i++) {
   A[i] = new double[N];
   B[i] = new double[N];
   C[i] = new double[N];
}
```

a)
```
   for (i = 0; i < N; i++)
      for (j = 0; j < N; j++)
         for (k = 0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j];
```

V primeru a) do j-tega stolpca (vektorja) matrike **B** dostopamo s stalnim menjavanjem blokov PP, medtem ko je dostop do i-te vrstice matrike **A** zaporeden.

b)
```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        Bt[i][j] = B[j][i];

for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        for (k = 0; k < N; ++k)
            C[i][j] += A[i][k] * Bt[j][k];
```

V primeru b) najprej matriko **B** transponiramo, zato da pri dostopanju do stolpca ne skačemo med različnimi bloki PP.

Poskus smo izvedli na računalniku s procesorjem Intel Core i7-4790 @ 3.6 GHz, 8 GB RAM, in predpomilniki s podatki spodaj:

Cache:	L1 data	L1 instruction	L2	L3
Size:	4 x 32 KB	4 x 32 KB	4 x 256 KB	8 MB
Associativity:	8-way set associative	8-way set associative	8-way set associative	16-way set associative
Line size:	64 bytes	64 bytes	64 bytes	64 bytes
Comments:	Direct-mapped (lasten enemu jedru, ne direktni v smislu 1-way!)	Direct-mapped	Non-inclusive Direct-mapped	Inclusive Shared between all cores

V prvem primeru dobimo CPEčas okrog 5 s, v drugem okrog 2 s (pohitritev torej kar **2.5**).

Še enostavneje pa je, da dostopamo z zankami v spremenjenem vrstnem redu. Npr.:

```
for (i = 0; i < N; i++)
   for (k = 0; k < N; k++)
      for (j = 0; j < N; j++)
         C[i][j] += A[i][k] * B[k][j];
```

V tem primeru do matrik v notranji zanki (j) dostopamo po vrsti, ker se matrikama C in B spreminja drugi indeks (j), kar pomeni, da gre po sosednjih elementih -> prostorska lokalnost!