Predmet: Arhitektura računara
Profesor: redovni profesor dr Dušan Regodić, dipl. inž.

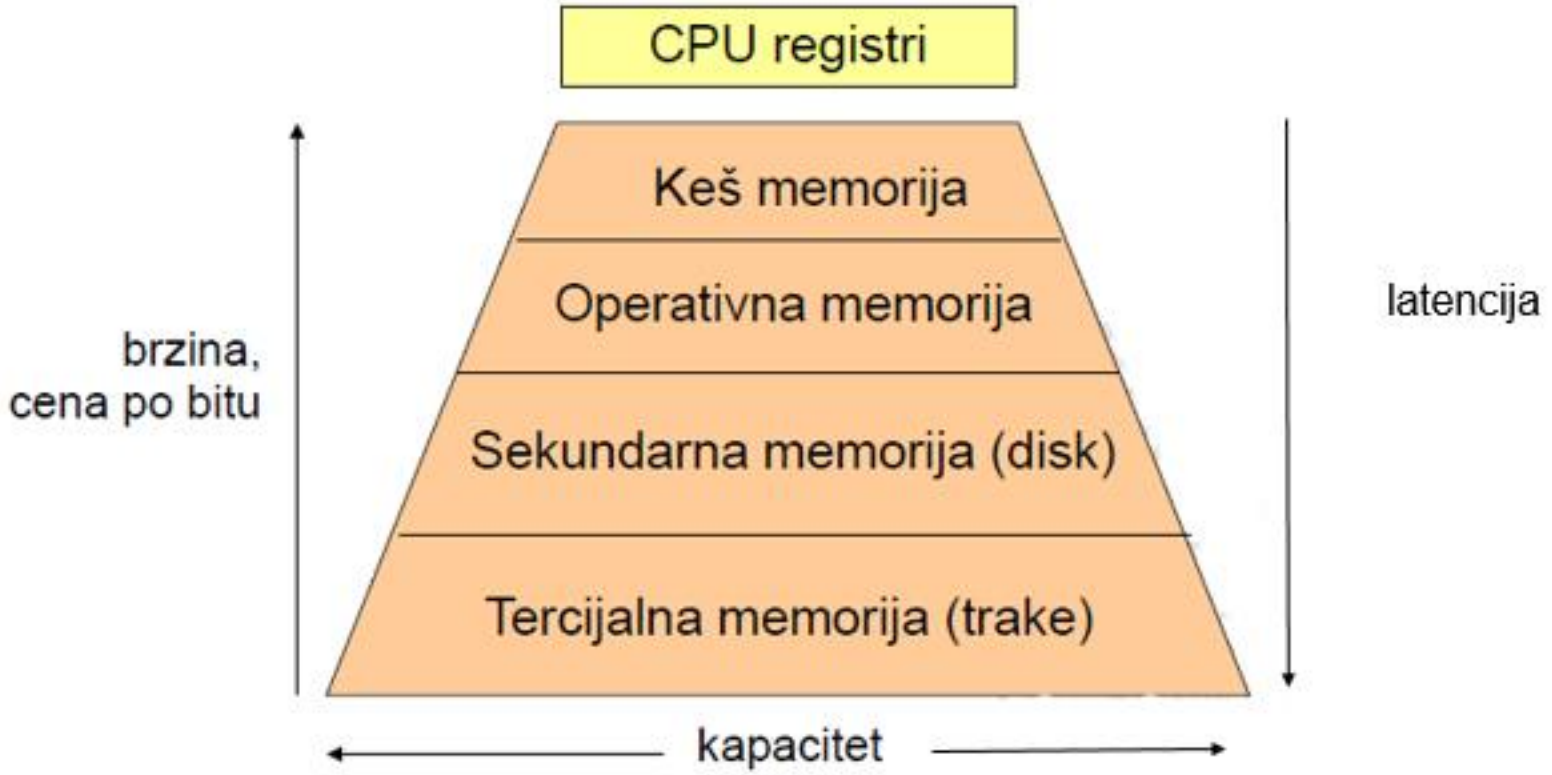
Memorijski sistem I

1. MEMORIJSKA HIJERARHIJA
2. KEŠ MEMORIJA

Memorijska hijerarhija (1)

- ideja o hijerarhijskoj organizaciji računarske memorije potiče još od Von Neumann-a (1946.g.)
- hijerarhija polazi od malih, skupih i relativno brzih memorijskih jedinica koje su praćene velikim, jeftinijim i relativno sporim jedinicama
- memorijska hijerarhija se karakteriše parametrima:
  - vreme pristupa (operacija upisa/čitanja)
  - kapacitet (meri se u bajtovima)
  - vreme ciklusa (vreme proteklo između dve uzastopne operacije upisa)
  - latencija (vreme proteklo od zahteva za informacijom do pristupa prvom bitu te informacije)
  - cena (valuta/megabajtu)

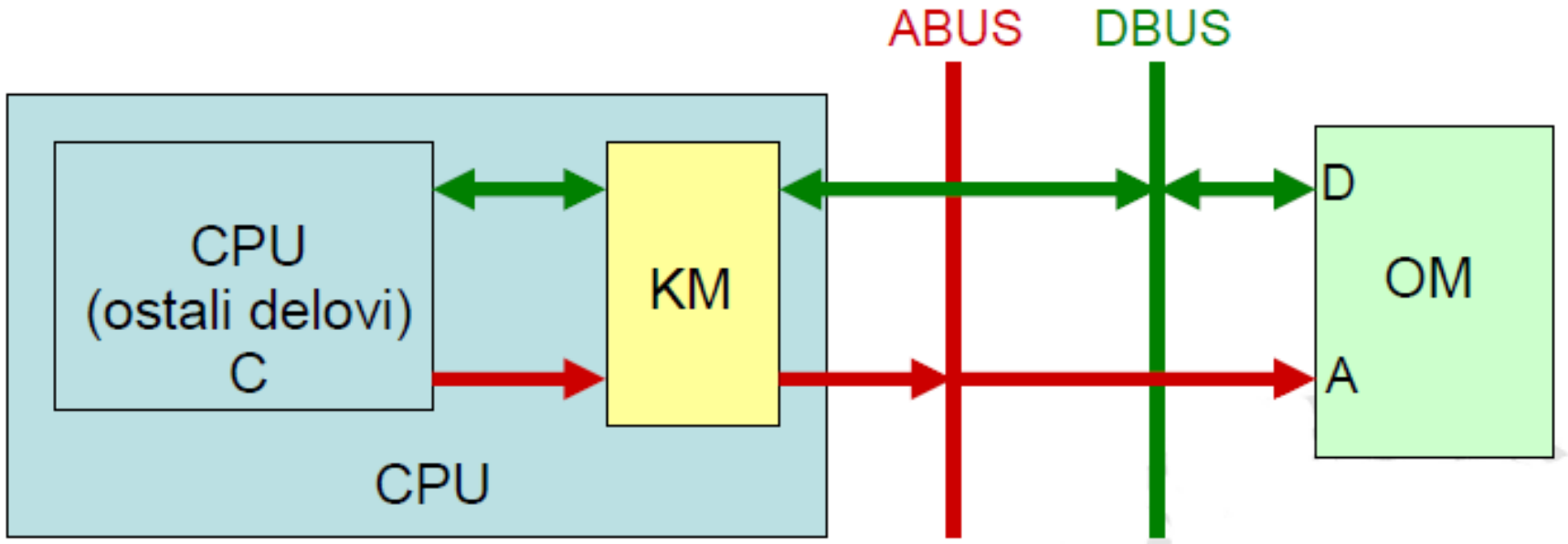
Memorijska hijerarhija (2)



Keš memorija

- uveo je Wilkes 1965.g.
- rešava problem vremena pristupa, tj. omoćava brži pristup sadržaju memorijskih lokacija
- nalazi se u procesoru kao posebna komponenta
- realizuje se memorijskim jedinicama čije je vreme pristupa mnogo manje nego kod OM i vrlo blisko vremenu prenosa između registara
- cena po bitu je znatno veća nego kod OM, pa je kapacitet znatno manji od kapaciteta OM

Princip rada

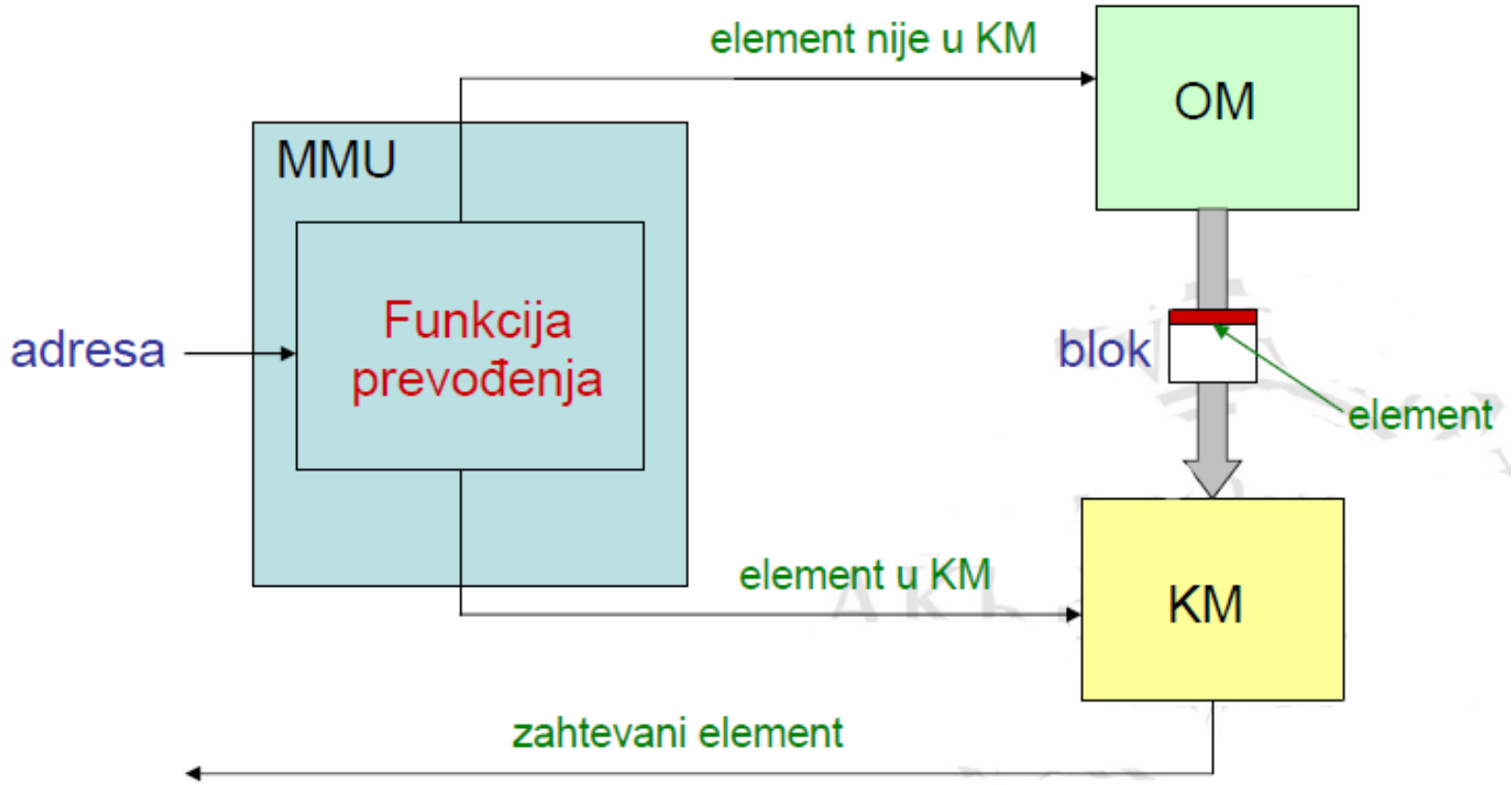


- C generiše adresu OM
- provera da li je sadržaj sa te adrese u KM
- u početku nije, pa se sadržaj na toj i susednim adresama prenosi iz OM u KM
- pristupa se sadržaju KM
- postupak se ponavlja za nekoliko sledećih adresa, tako da se KM puni
- u nekom trenutku, sadržaj će biti u KM, pa će pristup biti brži

Mapiranje adresa (1)

- procesor zahteva pristup nekom memorijskom elementu generisanjem njegove adrese
- adresa može da odgovara elementu koji trenutno postoji u KM
- ako ne odgovara, adresa može da odgovara elementu koji se nalazi u OM
- prema tome, potrebno je prevođenje adrese kako bi se došlo do željenog elementa
- mapiranje adresa je funkcija koju izvršava jedinica za upravljenje memorijom - MMU (*Memory Management Unit*)
- MMU se obično implementira kao deo CPU, ali može da bude i posebno integrisano kolo

Mapiranje adresa (2)



Problemi

- evidencija o tome koji se blokovi OM trenutno nalaze u keš memoriji i gde su smešteni – tehnike preslikavanja
- odlučivanje o tome koji blok treba izbaciti iz pune keš memorije kako bi se stvorio prostor za upis novog bloka – tehnike zamene
- ažuriranje sadržaja OM u slučaju kada je bilo upisa u lokacije keš memorije, jer tada sadržaji na istoj adresi u keš memoriji i OM nisu isti – tehnike ažuriranja

Tehnike preslikavanja

- direktno preslikavanje
- asocijativno preslikavanje
- set-asocijativno preslikavanje

Direktno preslikavanje (1)

- najjednostavnija tehnika preslikavanja
- smešta dolazeći blok OM na fiksnu lokaciju namenjenu bloku u keš memoriji

Smeštanje bloka OM u keš memoriju

broj dolazećeg bloka iz OM: *i*

broj bloka u KM: *j*

ukupan broj blokova u KM: *NCB*

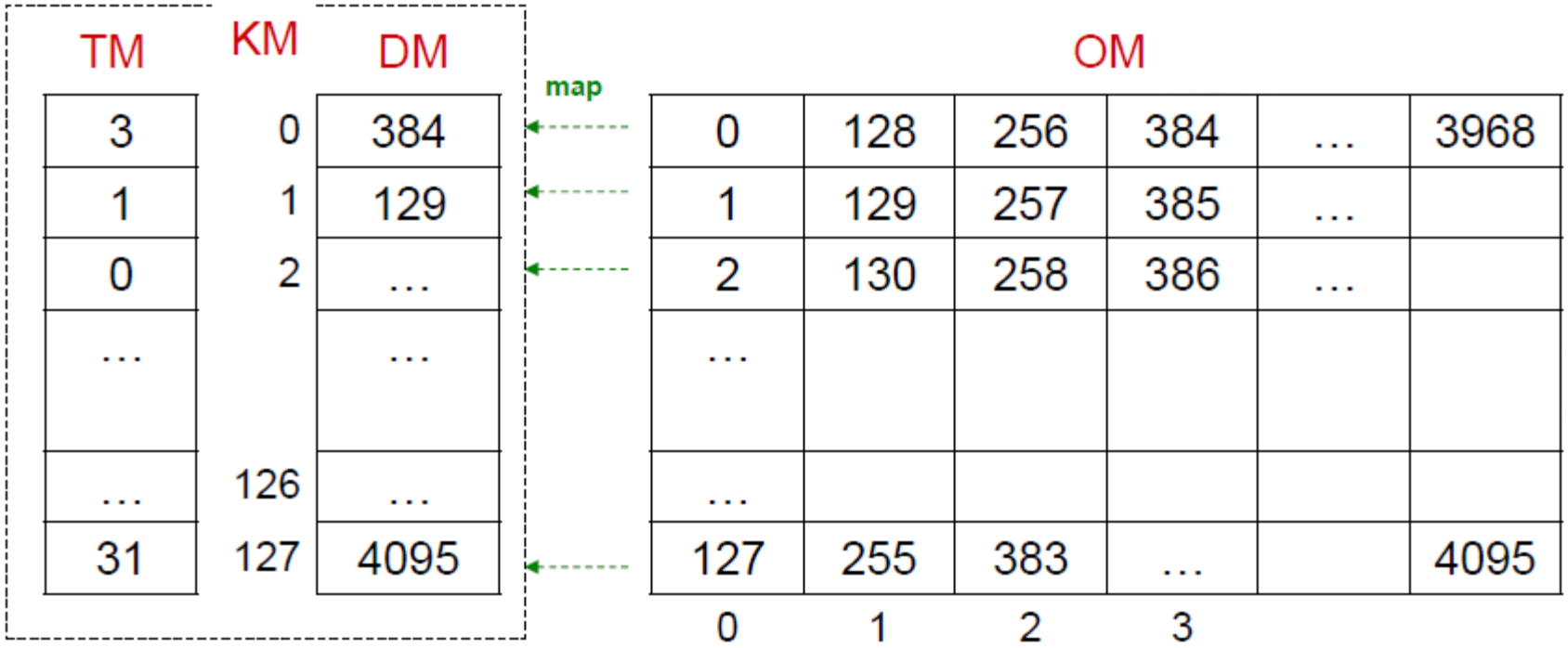
važi fiksna relacija: $j = i \bmod NCB$

- keš memorija se sastoji od TM (*Tag Memory*) i DM (*Data Memory*)

Direktno preslikavanje (2)

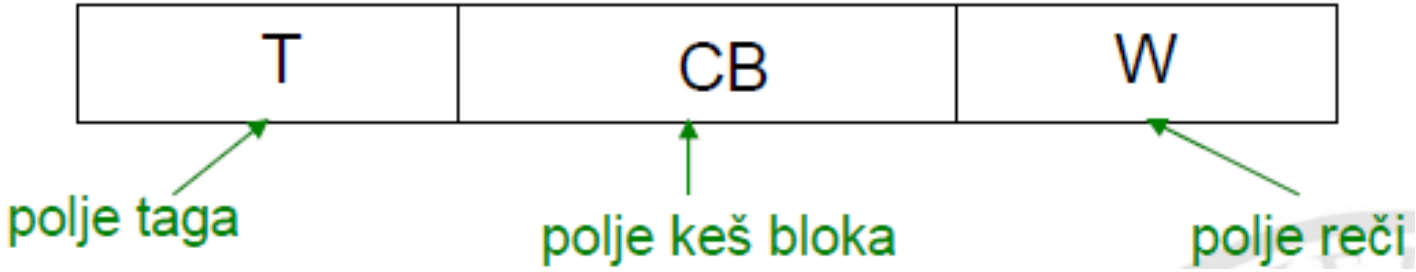
Primer 1:

- OM sadrži *NMB* = 4K blokova (4096)
- keš memorija sadrži *NCB* = 128 blokova
- veličina bloka je *B* = 16 memorijskih reči



Direktno preslikavanje (3)

- MMU interpretira adresu dobijenu od procesora tako što je podeli u tri polja sa *T*, *CB* i *W* bitova, respektivno



Određivanje broja bitova u poljima

$T = \log_2(NMB/NCB)$ $CB = \log_2 NCB$ $W = \log_2 B$

A - broj bitova u adresi ćelije u OM

$A = \log_2(B \cdot NMB)$

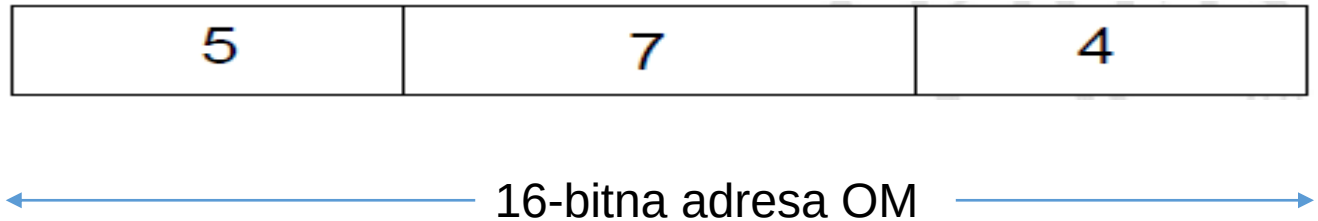
Direktno preslikavanje (4)

Primer 1:

NMB = 4096
NCB = 128
B = 16

$T = \log_2(NMB/NCB) = \log_2(4096/128) = \mathbf{5}$ bitova
$CB = \log_2 NCB = \log_2 128 = \mathbf{7}$ bitova
$W = \log_2 B = \log_2 16 = \mathbf{4}$ bita

$A = \log_2(16 \cdot 4096) = \mathbf{16}$ bitova



Direktno preslikavanje (5)

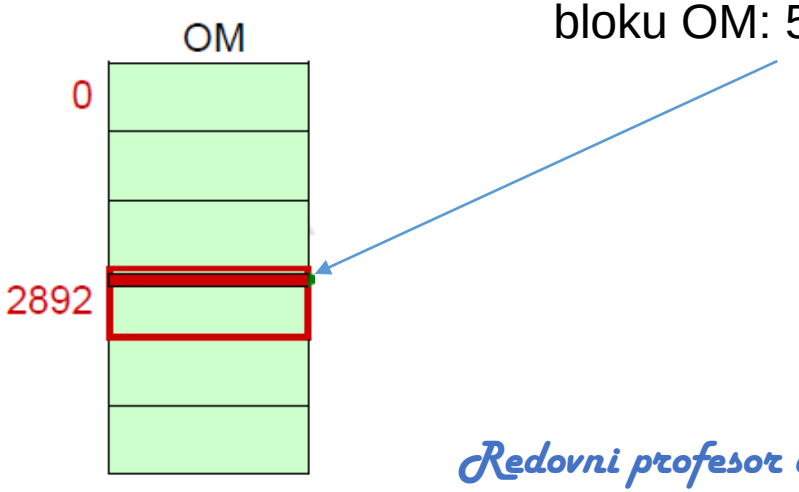
Primer 1: ilustracija da je adresa koju generiše procesor istovremeno i adresa podatka u OM i adresa tog podatka u KM

adresa koju generiše procesor (16-bitna): $1011010011000101_{(2)} = 46277_{(10)}$

46278 / 16 = 2892 (6)

blok OM u kome se nalazi podatak: 2892

pozicija podatka u bloku OM: 5



Napomena:
Adrese, blokovi u OM i KM, pozicije reči unutar bloka, kao i tagovi, broje se počevši od 0!

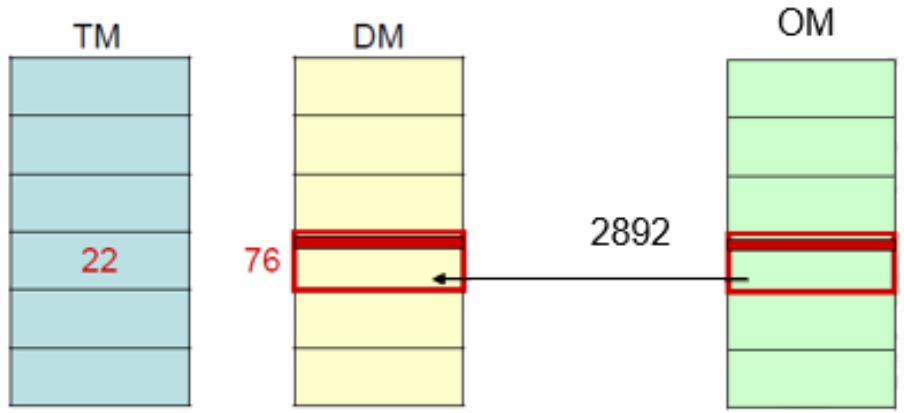
Direktno preslikavanje (6)

Primer 1:
adresa koju generiše procesor (16-bitna): 1011010011000101(2) = 46277(10)

i = 2892, *NCB* = 128 → *j* = *i* mod *NCB* = 2892 mod 128 = 76

računanje taga: 2892 / 128 = 22 (76)

T (5) polje:**10110(2)=22(10)** CB (7) polje: **1001100(2)=76(10)** W (4) polje: **0101(2)=5(10)**



Direktno preslikavanje (7)

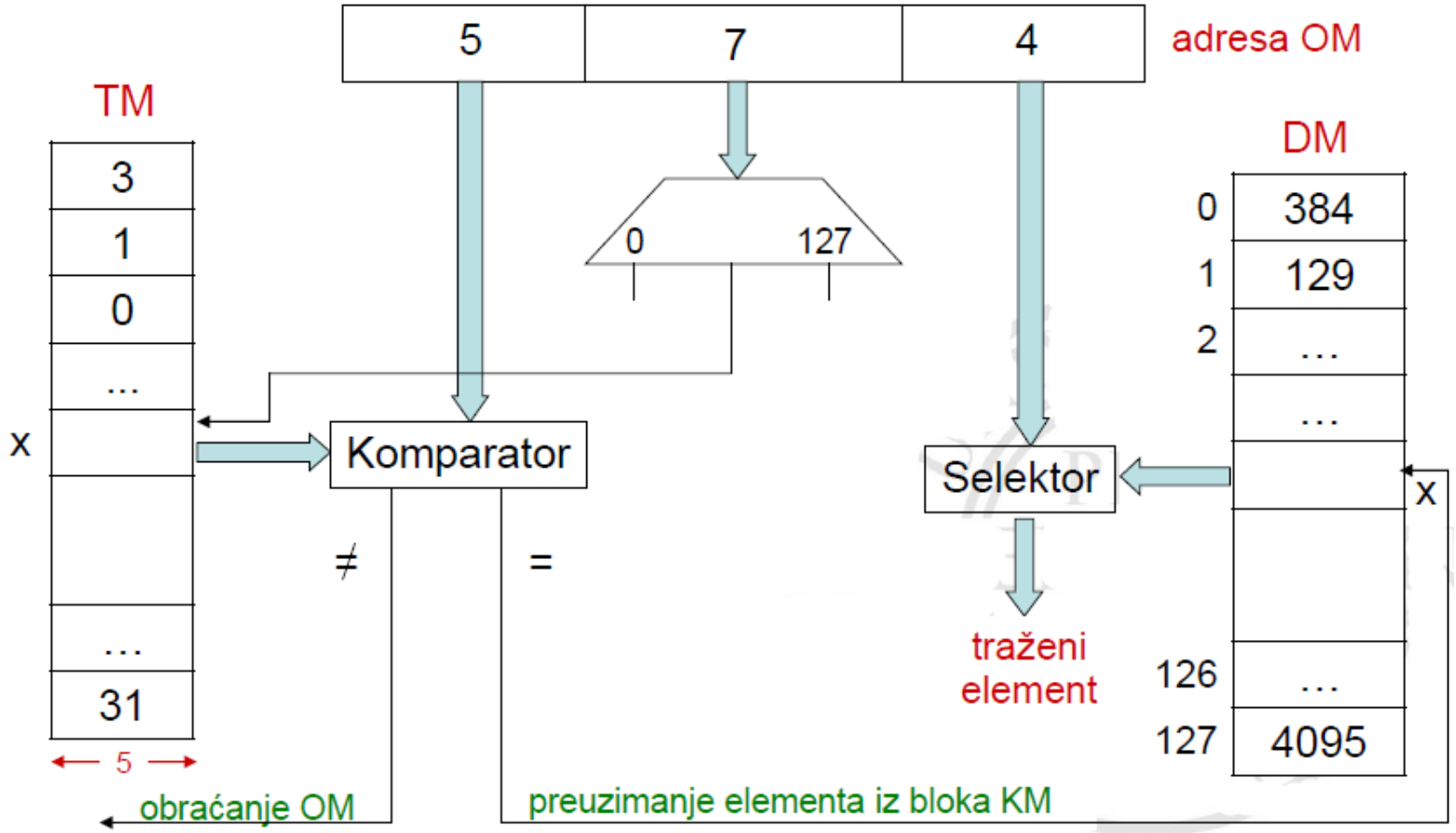
T	CB	W

Da bi se pristupilo elementu čiju je adresu generisao procesor, MMU primenjuje sledeći protokol:

1.na osnovu polja *CB* određuje se keš blok u kome bi trebalo da se nalazi zahtevani element
2.poređenjem vrednosti polja *T* sa TM zaključuje se da li je element u KM ili nije (iste vrednosti → element je u KM)
3.ako je element u KM, treba ga naći unutar bloka, što se postiže na osnovu vrednosti u polju *W*
4.ako element nije u KM, blok iz OM treba preneti u KM i element proslediti procesoru; na kraju, treba ažurirati TM i blok u DM

Direktno preslikavanje (8)

Primer 1:



Direktno preslikavanje (9)

- direktno preslikavanje je više-na-jedan tehnika mapiranja

Prednost:

•jednostavno, direktno određivanje mesta u KM gde će biti smešten blok OM

Nedostatak:

•neefikasno korišćenje KM (više memorijskih blokova konkurišu za isto mestu u KM, a možda postoje drugi, prazni keš blokovi)

Acosijativno preslikavanje (1)

- fleksibilnije od direktnog preslikavanja zato što se blok koji dolazi iz OM može smestiti u bilo koji raspoloživ blok KM

- adresa OM koju generiše procesor ima smo dva polja:

  - *T* – jednoznačno identifikuje blok OM koji se nalazi u KM
  - *W* – identifikuje element unutar bloka

$T = \log_2 NMB \quad W = \log_2 B \quad A = \log_2(B \cdot NMB) = T + W$

Acosijativno preslikavanje (2)

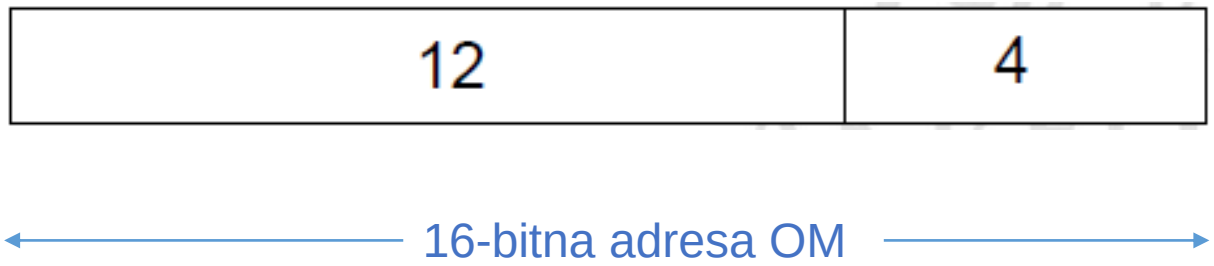
Primer 2:

NMB = 4096 $T = \log_2 NMB = \log_2 4096 = 12$ bitova

NCB = 128 $W = \log_2 B = \log_2 16 = 4$ bita

$B = 16$

$A = \log_2(16 \cdot 4096) = 16$ bitova



Acosijativno preslikavanje (3)

Da bi se pristupilo elementu čiju je adresu generisao procesor, MMU primenjuje sledeći protokol:

1.ispitati da li se vrednost polja *T* nalazi među postojećim tagovima u TM

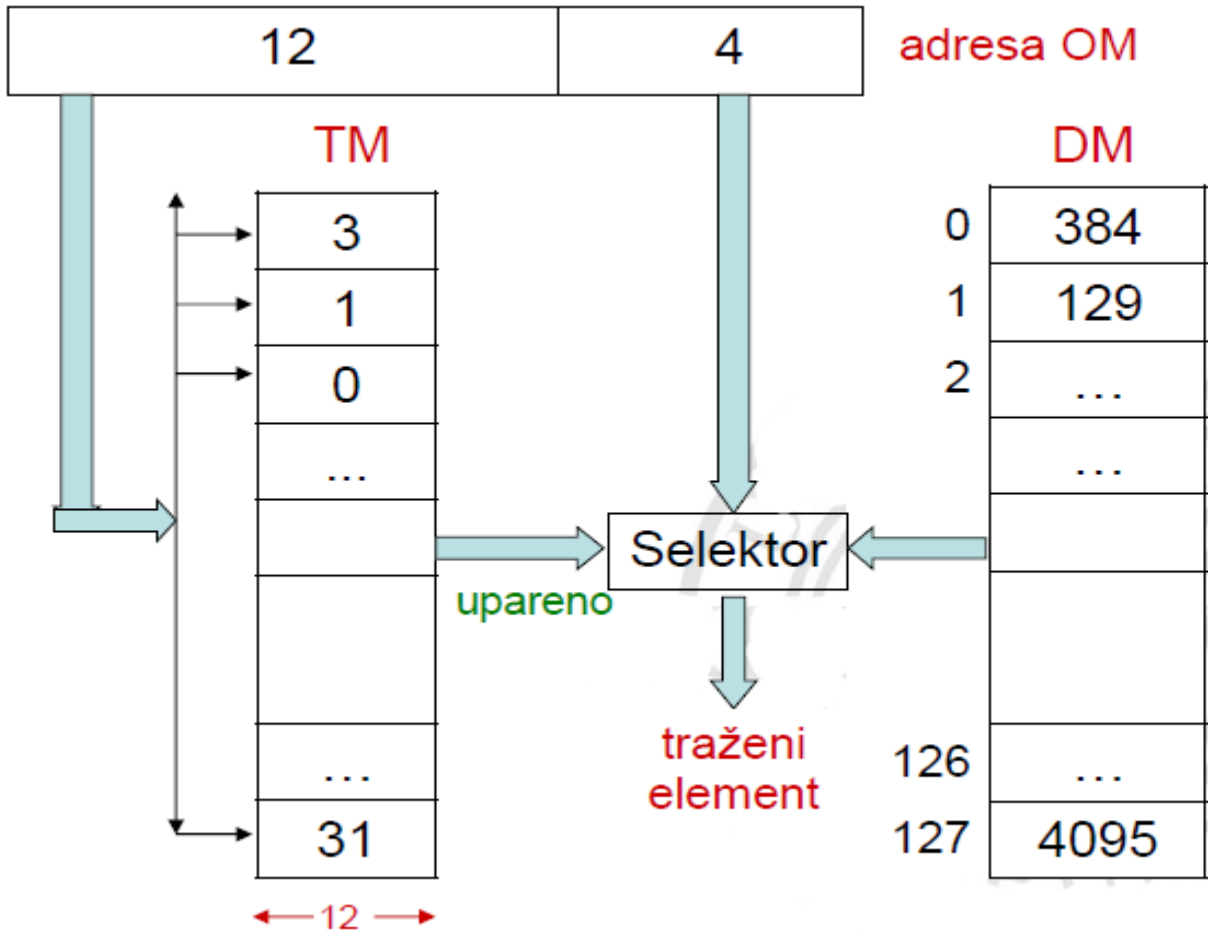
2.ako se *T* nalazi, traženi element je u odgovarajućem bloku KM

3.element se unutar bloka pronalazi pomoću vrednosti u polju *W*

4.ako se *T* ne nalazi, blok iz OM treba preneti u KM i element proslediti procesoru; na kraju, treba ažurirati TM i blok u DM

Acosijativno preslikavanje (4)

Primer 2:



Acosijativno preslikavanje (5)

- pretraživanje memorije tagova bi trajalo dugo, ako bi pretraživanje bilo sekvencijalno
- zato se tagovi čuvaju u asocijativnoj memoriji (paralelno pretraživanje)

Prednost:

•nema restrikcija po pitanju mesta u KM gde će se smestiti blok koji dolazi iz OM

Nedostatak:

•potreban hardver za asocijativno pretraživanje memorije tagova

Set-asocijativno preslikavanje (1)

- keš memorija je podeljenja u skupove (setove - *sets*), pri čemu svaki set sadrži određen broj blokova

- blok koji dolazi iz OM mapira jedan set u KM prema formuli

$$s = i \bmod NS$$

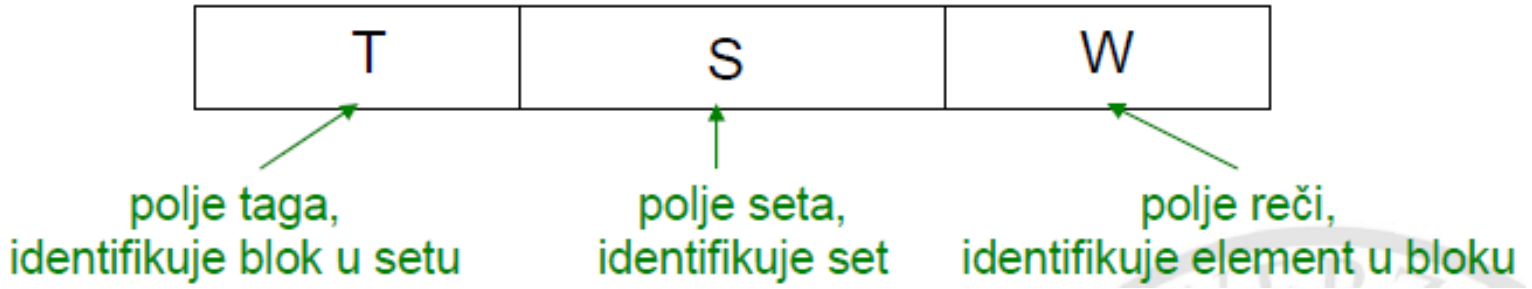
gde su

i – broj bloka OM
NS – broj setova u KM
s – broj seta u KM u koji se mapira

- blok *i* iz OM mapira bilo koji blok u setu *s* u KM

Set-asocijativno preslikavanje (2)

- adresa koju generiše procesor ima tri polja sa *T*, *S* i *W* bitova, respektivno



Određivanje broja bitova u poljima

$T = \log_2(NMB \cdot BS / NCB)$ $S = \log_2 NS$ $W = \log_2 B$

BS - broj blokova u setu
A - broj bitova u adresi ćelije OM

$A = \log_2(B \cdot NMB)$

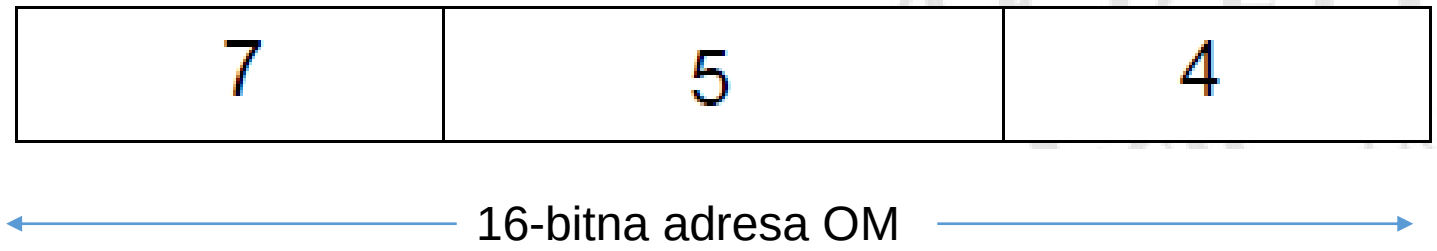
Set-asocijativno preslikavanje (3)

Primer 3:

NMB = 4096
NCB = 128
B = 16
BS = 4
NS = 128/4 = 32

W = log2*B* = log216 = **4** bita
S = log2*NS* = log232 = **5** bitova
T = log2(*NMB*·*BS*/*NCB*) = log2(4096·4/128) = **7** bitova

A = log2(16·4096) = **16** bitova

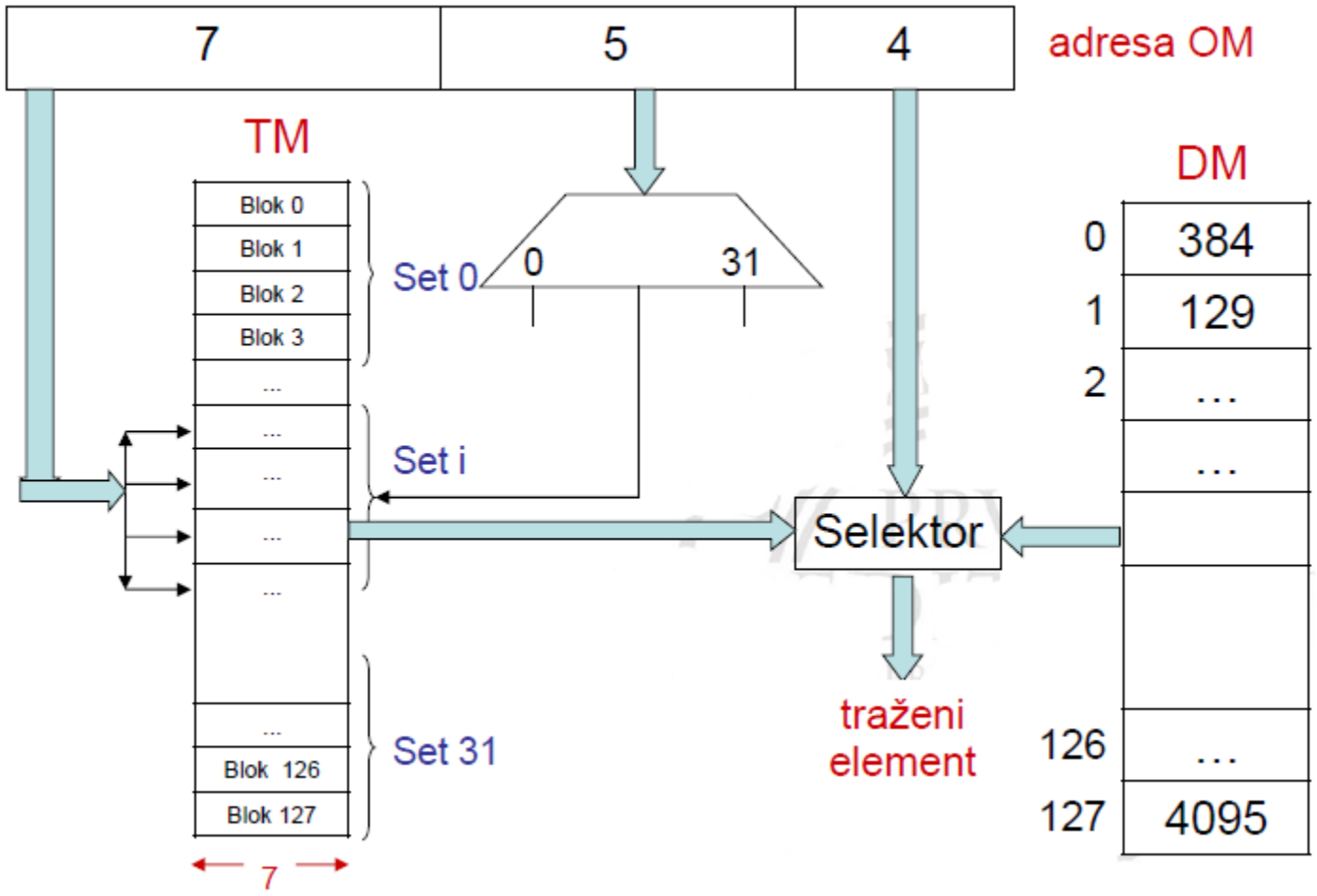


Set-asocijativno preslikavanje (4)

Da bi se pristupilo elementu čiju je adresu generisao procesor, MMU primenjuje sledeći protokol:

1.na osnovu polja *S* direktno određuje set u koji se mapira
2.ispituje da li se vrednost polja *T* nalazi među postojećim tagovima u TM za dati set; ukoliko se nalazi, to znači da je taj blok već u KM
3.element unutar bloka se pronalazi pomoću vrednosti u polju *W*
4.ako se *T* ne nalazi, blok iz OM treba preneti u odgovarajući set KM i element proslediti procesoru; na kraju, treba ažurirati TM i blok u DM

Set-asocijativno preslikavanje (5)



Set-asocijativno preslikavanje (6)

- tagovi se čuvaju u asocijativnoj memoriji da bi pretraživanje bilo brže (paralelno)
- po osobinama, ova tehnika je kombinacija direktnog i asocijativnog preslikavanja
- nije toliko efikasna kao asocijatvno preslikavanje, ali je preuzela jednostavnost direktnog preslikavanja

Poređenje tehnika preslikavanja

Tehnika preslikavanja	Jednostavnost	Asocijativno pretraživanje tagova	Očekivano iskorišćenje keša	Tehnika zamen
Direktno	Da	Ne	Nisko	Nepotrebna
Asocijativno	Ne	Da	Visoko	Potrebna
Set-asocijativno	Srednja	Umereno	Srednje	Potrebna

Tehnike zamene

- rešavaju problem izbora bloka u punoj KM koji će biti zamenjen blokom koji dolazi iz OM

Izbor bloka može biti:

- po slučajnom izboru (*Random Selection*)
- po vremenu provedenom u KM (*FIFO: First-In-First-Out*)
- po trenutku poslednjeg korišćenja bloka (*LRU: Least Recently Used*)

Slučajan izbor

- u računaru postoji generator slučajnih brojeva GSB (*random generator*) koji po uključenju računara počinje da generiše brojeve od 0 do N-1
- tehnika se zasniva na tome da se broj bloka u KM koga treba zameniti blokom iz OM određuje izlazom GSB u trenutku zamene
- tehnika je vrlo jednostavna
- nedostatak: ne uzima se u obzir lokalnost referenci
- prvo korišćenje tehnike je bilo u Intel-ovoj iAPX seriji mikroprocesora

FIFO izbor

- mera za zamenu je vreme koje je blok proveo u KM
- tehnika se zasniva na tome da se izbacuje blok koji je najviše vremena proveo u KM, bez obzira na njegovo korišćenje u tom periodu
- tehnika zahteva vođenje evidencije o trenucima unosa blokova u KM, pa je složenija od slučajnog izbora
- tehnika je pogodna za pravolinijske programe gde lokalnost nije bitna

LRU izbor (1)

- tehnika se zasniva na tome da se iz KM izbacuje blok koji je najranije poslednji put korišćen
- najefikasnija tehnika, zato što prati istoriju upotrebe blokova
- tehnika zahteva korišćenje keš kontrolera koji čuva istoriju referenci na sve blokove u KM za sve vreme dok su tu

LRU izbor (2)

Jedna implementacija keš kontrolera:

- svakom bloku u KM je pridružen brojač; vrednost brojača je veća ako je blok ranije korišćen; svi brojači imaju različite vrednosti koje pokazuju redosled korišćenja blokova

- ako je podatak u bloku koji već postoji u KM
  - brojač odgovarajućeg bloka, koji trenutno ima vrednost *b*, postavlja se na 0
  - svi brojači sa vrednošću manjom od *b* se inkrementiraju za 1
  - svi brojači sa vrednošću većom od *b* se ne menjaju

- ako je podatak u bloku koji ne postoji u KM
  - zamenjuje se blok sa najvećom vrednošću brojača
  - brojač zamenjenog bloka se postavlja na 0
  - vrednosti svih ostalih brojača se inkrementiraju za 1

LRU izbor (3)

Primer 4: iz OM dolaze blokovi 6, 11, 1 i 20

promene vrednosti brojača

KM	brojači	Blok 6	Blok 11	Blok 1	Blok 20
B10	3	4	5	5	6
B3	4	5	6	6	7
B15	1	2	3	3	4
B1	0	1	2	0	1
B6	5	0	1	2	3
B9 →B20	10	10	11	11	0
B12	6	6	7	7	8
B5	2	3	4	4	5
B32	7	7	8	8	9
B4→B11	12	12	0	1	2

Tehnike ažuriranja

- bave se problemom koherencije keša
- koherencija podrazumeva usaglašenost između reči u KM i njihovih kopija u OM

Vrste tehnika

- tehnika upisa ako blok postoji u KM
- tehnika upisa ako blok ne postoji u KM
- tehnika čitanja ako blok postoji u KM (samo se pročita vrednost)
- tehnika čitanja ako blok ne postoji u KM

Tehnika upisa ako blok postoji u KM

(*Cache Write Policy Upon a Cache Hit*)

Postoje dve mogućnosti upisa:

- **trenutni upis (*writte-through*)**

→ svakom operacijom upisa se istovremeno vrši upis i u KM i u OM

- **odloženi upis (*writte-back*)**

→ upis se radi samo u KM, dok je upis u OM odložen do trenutka kada se javi potreba za zamenom bloka
→ svakom bloku u KM pridružen je bit (*dirty bit*) čija vrednost pokazuje da li je postojao bar jedan upis u taj blok od kada je on došao u KM
→ u trenutku zamene, ispituje se bit, pa ako je 1, blok KM se upisuje u OM, a ako je 0, samo se dolazeći blok prepiše preko bloka KM

Tehnika upisa ako ne postoji blok u KM

(*Cache Write Policy Upon a Cache Hit*)

Postoje dve mogućnosti upisa:

- **dovuci blok (*writte-allocate*)**

→ blok se dovlači iz OM u KM i onda se radi kao u prethodnoj tehnici

- **ne dovlači blok (*writte-no-allocate*)**

→ blok se ne dovlači iz OM, već se upis vrši samo u OM

Tehnika upisa ako blok ne postoji u KM

(*Cache Write Policy Upon a Cache Hit*)

Postoje dve mogućnosti čitanja:

- **direktno prosleđivanje**

→ blok se dovlači iz OM u KM, a traženi podatak se odmah po pristizanju u KM prosleđuje u CPU

- **prosleđivanje sa zadrškom**

→ blok se dovlači iz OM u KM, pa kad se ceo smesti, onda se traženi podatak prosleđuje u CPU

HVALA VAM NA PAŽNJI

Redovni profesor dr Dušan Regodić, dipl.inž.