VERZIJA:7.11.2005.

5. MEMORIJA

U okviru ove glave razmatraju se tehnike kojima se rešavaju dva problema pri radu sa memorijom i to vreme pristupa i veličina adresnog prostora. U okviru rešavanja problema vremena pristupa razmatraju se dve tehnike i to preklapanje pristupa memorijskim modulima i keš memorija. U okviru rešavanja problema veličine adresnog prostora korisnika razmatra se virtuelna memorija.

5.1. PREKLAPANJE PRISTUPA MEMORIJSKIM MODULIMA

U ovom poglavlju se prikazuje kako se korišćenjem tehnike preklapanja pristupa memorijskim modulima ubrzava pristup memoriji i kako je moguće rasporediti adrese po memorijskim modulima.

5.1.1. Ubrzavanje pristupa memoriji

Uobičajene konfiguracije računara sadrže procesor, više ulazno/izlaznih uređaja i jedan modul memorije (slika). Ukoliko ulazno/izlazni uređaji sadrže jedino kontrolere bez direktnog pristupa memoriji procesor je jedini gazda koji koristi magistralu i pristupa memoriji. Ukoliko ulazno/izlazni uređaji sadrže kontrolere sa direktnim pristupom memoriji korišćenje magistrale i pristup memoriji može da stvara probleme. Problemi nastaju zbog toga što sada pored procesora i ulazno/izlazni uređaji sa kontrolerima sa direktnim pristupom memoriji kao gazda koriste magistralu i pristupaju memoriji. Zbog toga mogu da nastaju situacije kada su istovremeno i procesoru i ulazno/izlaznim uređajima sa kontrolerima sa direktnim pristupom memoriji potrebni magistrala i memorija. U takvim situacijama arbitracijom se odlučuje po kom redosledu će procesor i ulazno/izlazni uređaji sa kontrolerima sa direktnim pristupom memoriji da pristupaju memoriji. Ovim su stvoreni neželjeni kontraefekti. Da bi ceo sistem imao veću propusnu moć ulazno/izlazni uređaji su realizovani sa kontrolerima sa direktnim pristupom memoriji. Procesor je sada rasterećen poslova vezanih za ulaz/izlaz. Dok kontroleri sa direktnim pristupom memoriji obavljaju prenose podataka između periferja i memorije procesor je slobodan da vrši neke druge obrade. Međutim, u sistemu sa slike to ne može da dođe do izražaja. Zbog toga što je memorija realizovana kao jedan memorijski modul kada god procesor pristupa memoriji ulazno/izlazni uređaji moraju da čekaju ili kad god neki ulazno/izlazni uređaj pristupa memoriji ostali ulazno/izlazni uređaji i procesor moraju da čekaju.

Mogući pristup za prevazilaženje ovog problema je sistem u kome se memorija sastoji ne iz jednog, već iz više memorijskih modula (slika). Memorija M sa slike određenog kapaciteta zamenjena je na slici memorijom istog kapaciteta koja se sastoji od 2^k modula M_k, pri čemu je kapacitet modula M_k manji 2^k puta od kapaciteta memorije M. Pri tome je vreme pristupa memorijskog modula M_k isto kao vreme pristupa memorije M. U ovakvom sistemu postoji mogućnost da procesor i uređaji U/I istovremeno pristupaju memoriji ukoliko su adrese iz različitih memorijskih modula. Međutim, ukoliko su ciklusi na magistrali tako realizovani da gazda drži magistralu sve vreme trajanja čitanja iz memorije ili upisa u memoriju pristupi memoriji su i dalje sekvencijalni. Zbog toga se može desiti da dok procesor pristupa jednom memorijskom modulu i drži magistralu, uređaji U/I, iako imaju potrebu da pristupe drugim memorijskim modulima, to ne mogu da učine. Da bi paralelan pristup procesora i uređaja U/I različitim memorijskim modulima bio moguć potrebno je da magistrala bude realizovana sa podeljenim ciklusima.

Kod magistrale sa podeljenim ciklusima procesor ili uređaj U/I drži magistralu samo onoliko vremena koliko je potrebno da se iz procesora ili uređaja U/I u memorijski modul prenesu informacije neophodne za realizaciju čitanja iz memorije ili upisa u memoriju. Svaki memorijski modul mora da ima adresni registar i registar podatka da bi čitanje iz memorije ili upis u memoriju mogli da se realizuju. Ciklusi na magistrali su sada slanje zahteva za čitanje i slanje zahteva za upis. Dok traje samo čitanje iz memorijskog modula ili upis u memorijski modul magistrala je slobodna pa procesor ili neki drugi uređaj U/I mogu na magistrali da realizuju neki novi ciklus slanje zahteva za čitanje ili slanje zahteva za upis sa nekim drugim memorijskim modulom. Sada se paralelno čita iz i/ili upisuje u dva memorijska modula. Za to vreme magistrala je slobodna da procesor ili neki treći uređaj inicira neki treći ciklus zahteva za čitanje ili zahteva za upis sa nekim trećim memorijskim modulom. Ovim je ostvaren paralelan rad procesora i U/I uređaja sa različitim memorijskim modulima.

Kod magistrale sa podeljenim ciklusima nema problema sa upisom u memoriju, dok se problem javlja prilikom čitanja iz memorije. U slučaju upisa, najpre se na magistrali realizuje ciklus slanje zahteva za upis tako sto se u adresni registar i registar podatka memorijskog modula izvrši upis adrese i podatka, zatim se magistrala oslobađa, a onda se unutar memorijskog modula realizuje sam upis. U slučaju čitanja iz memorije, najpre se na magistrali realizuje ciklus slanje zahteva za čitanje tako sto se u adresni registar memorijskog modula izvrši upis adrese, zatim se magistrala oslobađa, a onda se unutar memorijskog modula realizuje samo čitanje i upis očitanog podatka u registar podatka. Pošto je po iniciranju čitanja od strane procesora ili uređaja U/I veza sa memorijskim modulom raskinuta, postoji potreba za posebnim ciklusom na magistrali kojim se vraća podatak u procesor ili uređaj U/I koji je incirao čitanje. Međutim, čitanje iz datog memorijskog modula može da bude inicirano od strane procesora ili bilo kog uređaja U/I i memorijski modul ne zna kome treba da vrati očitani podatak. Zbog toga procesor i uređaji U/I koji kao gazde u memorijskim modulima mogu da iniciraju čitanje dobijaju identifikatore koje šalje u memorijski modul sa adresom prilikom realizacije ciklusa zahtev za čitanje na magistrali. Prilikom ciklusa vraćanje podatka uz očitani podatak memorijski modul šalje i idenifikator procesora ili uređaja U/I koji je inicirao čitanje u memorijskom modulu. Procesor ili uređaj U/I koji prikom ciklusa vraćanje podatka prepozna identifikator kao svoj, prihvata očitani podatak.

Iz izloženo se vidi da kod magistrale sa podeljenim ciklusima na magistrali postoje tri ciklusa i to:

slanje zahteva za upis,

slanje zahteva za čitanje i

vraćanje podatka.

Onaj ko inicira neki od ova tri ciklusa na magistrali naziva se gazda, a onaj koji na njih reaguje naziva se sluga. Procesor i uređaji U/I su gazde prilikom realizacije ciklusa zahtev za upis i zahtev za čitanje, a memorijski moduli su tom prilikom sluge. Memorijski moduli su gazde prilikom realizacije ciklusa vraćanje podatka, a procesor i uređaji U/I sluge. Treba imati na umu da procesor može da realizuje čitanje i upis ne samo sa memorijom, već i sa uređajima U/I. To se dešava kada procesor upisom u registre uređaja U/I vrši njihovu inicijalizaciju i startovanje i kada čitanjem registara uređaja U/I dobija iz njih statusne informacije. Procesor i je gazde prilikom realizacije ciklusa zahtev za upis i zahtev za čitanje, a uređaji U/I su tom prilikom sluge. Uređaji U/I su gazde prilikom realizacije ciklusa vraćanje podatka, a procesor sluga. I gazde i sluge za realizaciju ciklusa imaju adresni registar, registar podatka i registar identifikatora.

Kod realizaciju ciklusa slanje zahteva za čitanje gazda

5.1.2. Raspored adresa po memorijskim modulima

Zahtevi za pristup memoriji po načinu njihovog generisanja od strane uređaja računara mogu da se podele u dve grupe:

- pojedinačno generisani zahtevi i
- blokovski generisani zahtevi.

Kod pojedinačnog generisanja zahteva karakteristično je to da se zahtevi generišu za jednu memorijsku reč i da postoji određeni period vremena između generisanja dva zahteva. Ovakav način generisanja zahteva se javlja kod procesora prilikom izvršavanja instrukcija. Ovde je tipično da se procesor prvo obrati memoriji radi čitanja instrukcije, da zatim nešto uradi nezavisno od memorije, da se potom ponovo obrati memoriji radi čitanja jednog do dva operanda, da iza toga nešto uradi nezavisno od memorije, da se ponovo obrati memoriji radi upisa rezultata, itd…. Slična je situacija i kada DMA kontroler prenosi podatke između neke spore periferije i operativne memorije ili obratno. Ovde DMA kontroler pristupa memoriji radi čitanja ili upisa jedne memorijske reči, zatim čeka da periferija bude spremna da primi novi podatak ili da spremi novi podatak, pa se ponovo obraća memoriji, itd…. Ova obraćanja memoriji i čekanje na spremnost periferije se ponavljaju onoliko puta koliko reči treba preneti između periferije i operativne memorije ili obratno. Uređaji kao procesor, DMA kontroler, spore periferije, itd… koji na ovakav način generišu zahteve za pristup memoriji nazivaće se u daljem tekstu uređaji sa pojedinačnim pristupom memoriji.

Kod blokovskog generisanja zahteva karakteristično je to da se zahtevi generišu za niz memorijskih reči bez pravljenja pauze između dva zahteva. Ovakav način generisanja zahteva se javlja kod keš kontrolera prilikom prenosa bloka poodataka između keš memorije i operativne memorije ili obratno. Ovde je tipično da se odmah po prenosu prve reči bloka, prenosi sledeća reč i tako redom do prenosa svih reči bloka. Slična je situacija i kada DMA kontroler prenosi podatke između neke brze vremenski kritične periferije i operativne memorije ili obratno. Ovde DMA kontroler pristupa memoriji u *burst* režimu rada generišući zahteve za čitanje iz ili upis u memoriju jedan za drugim onom brzinom sa kojom je operativna memorija u stanju da ih prihvati. Uređaji kao keš kontroler, DMA kontroler, brze vremenski kritične periferije, itd…, koji na ovakav način generišu zahteve za pristup memoriji nazivaće se u daljem tekstu uređaji sa blokovskim pristupom memoriji.

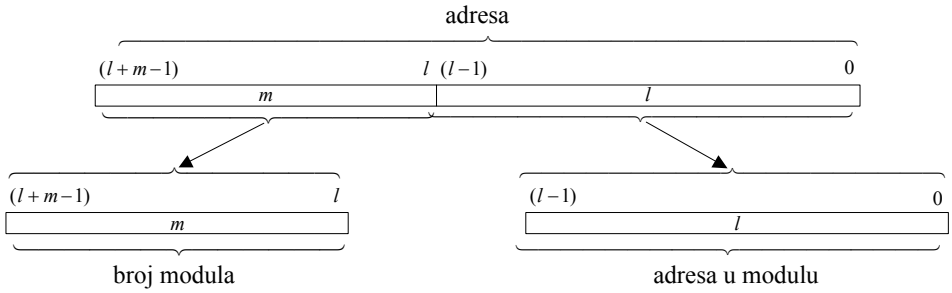
Uređaji sa pojedinačnim pristupom memoriji i uređaji sa blokovskim pristupom memoriji nazivaće se u daljem tekstu samo kao uređaji kada god se radi o nečemu što je zajedničko za obe vrste uređaja.

Za ostvarivanje efikasnog preklapanja pristupa memoriji sa više memorijskih modula posebna pažnja treba da se posveti rasporedu adresa po memorijskim modulima. On treba što je moguće više da bude prilagođen strukturi adresa generisanih pri većem broju istovremenih zahteva za obraćanje memoriji i to tako da one budu za različite memorijske module. Uočene su neke tipične situacije mogućih struktura generisanih adresa i na osnovu toga prihvaćena sledeća tri načina rasporeda adresa po memorijskim modulima:

- susedne u istom,
- susedne u susednim i
- mešovito.

Ova tri načina rasporeda adresa po memorijskim modulima biće razmotrena za slučaj da je totalni kapacitet memorije 2^h bajta i da je memorija realizovana od 2^m modula pri čemu je kapacitet jednog modula 2^l bajta. Adresa je, stoga, dužine h bita od kojih se m koristi za kodiranje broja modula, a l za kodiranje adrese bajta u bloku.

U slučaju rasporeda adresa susedne u istom struktura adrese i način formiranja broja modula i adrese u modulu su prikazani na slici 1. Sa m najstarijih bitova kodira se broj modula, a sa l najmlađih bitova adresa bajta u modulu. Raspored adresa po modulima je prikazan na slici 2. U nultom modulu nalaze se adrese od 0 do $2^l - 1$, u prvom modulu od 2^l do $2 \cdot 2^l - 1$ i tako redom do $(2^m - 1)$-og modula u kome se nalaze adrese iz opsega $(2^m - 1) \cdot 2^l$ do $(2^m - 1) \cdot 2^l + (2^l - 1)$. Ovim je totalni opseg adresa od 2^{l+m} bajta podeljen na 2^m kontinualnih celina, svaka kapaciteta 2^l bajta.

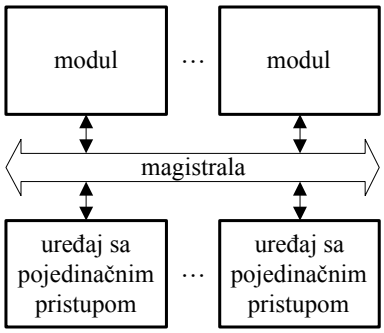


Slika 1 Broj modula i adresa u modulu za susedne u istom

Ovakav način rasporeda adresa po modulima pogodan je za konfiguraciju računara kao na slici 3. Uređaj sa pojedinačnim pristupom memoriji obično u nekom vremenskom intervalu generiše adrese iz određenog opsega adresa. Pošto tih uređaja ovde ima više, moguće je podesiti da se ti opsezi adresa za uređaje sa pojedinačnim pristupom memoriji nalaze u različitim memorijskim modulima. Kako uređaji sa pojedinačnim pristupom memoriji rade nezavisno, to oni mogu da generišu istovremeno zahteve za pristup memoriji. Ti zahtevi će u ovom slučaju biti za različite memorijske module i moći će da se realizuju paralelno.

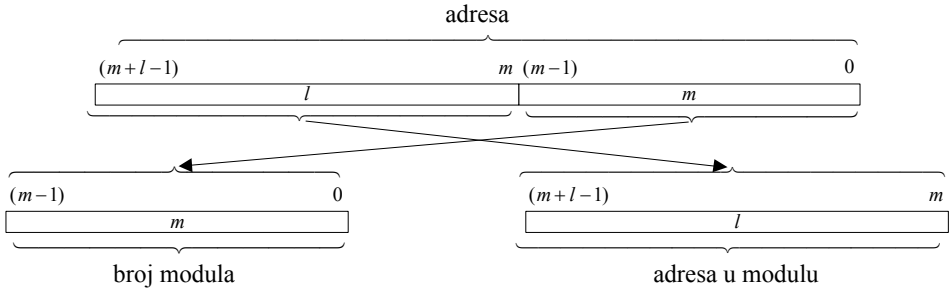
$0 \cdot 2^l + 0$
$0 \cdot 2^l + 1$
$0 \cdot 2^l + (2^l - 1)$
$1 \cdot 2^l + 0$
$1 \cdot 2^l + 1$
$1 \cdot 2^l + (2^l - 1)$
⋮
$(2^m - 1) \cdot 2^l + 0$
$(2^m - 1) \cdot 2^l + 1$
$(2^m - 1) \cdot 2^l + (2^l - 1)$

Slika 2 Raspored adresa po modulima za susedne u istom

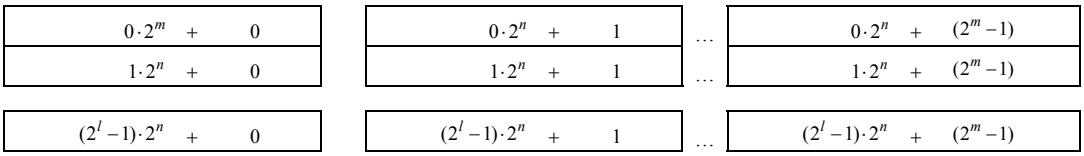


Slika 3 Konfiguracija računara za susedne u istom

U slučaju rasporeda adresa susedne u susednim struktura adrese i način formiranja broja modula i adrese u modulu su prikazani na slici 4. Sa m najmlađih bitova kodira se broj modula, a sa l najstarijih bitova adresa bajta u modulu. Raspored adresa po modulima je prikazan na slici 5. U nultom modulu nalaze se adrese $0\cdot 2^m+0$, $1\cdot 2^m+0$, $2\cdot 2^m+0$ i tako redom do $(2^l-1)\cdot 2^m+0$. U prvom modulu nalaze se adrese $0\cdot 2^m+1$, $1\cdot 2^m+1$, $2\cdot 2^m+1$ i tako redom do $(2^l-1)\cdot 2^m+1$. U drugom modulu nalaze se adrese $0\cdot 2^m+2$, $1\cdot 2^m+2$, $2\cdot 2^m+2$ i tako redom do $(2^l-1)\cdot 2^m+2$. Po analogiji sa ovim moguće je utvrditi i raspored adresa i po preostalim modulima. U poslednjem (2^l-1)-om modulu nalaze se adrese $0\cdot 2^m+(2^m-1)$, $1\cdot 2^m+(2^m-1)$, $2\cdot 2^m+(2^m-1)$ i tako redom do $(2^l-1)\cdot 2^m+(2^m-1)$. Ovim je totalni opseg adresa od 2^{l+m} bajta podeljen na 2^m modula kapaciteta od po 2^l bajta, ali tako da se opsezi susednih adresa od 0 do (2^m-1) nalaze u 2^m susednih modula. Takvih opsega adresa ima 2^l.

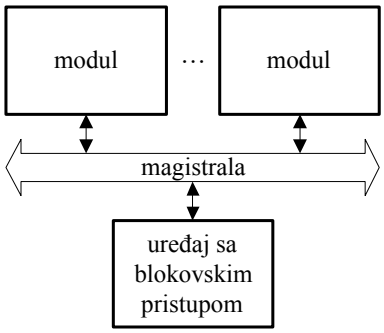


Slika 4 Broj modula i adresa u modulu za susedne u susednim



Slika 5 Raspored adresa po modulima za susedne u susednim

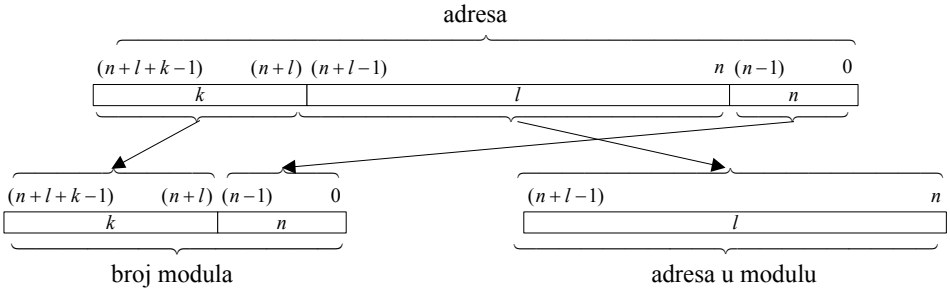
Ovakav način rasporeda adresa po modulima pogodan je za konfiguraciju računara kao na slici 6. Uređaj sa blokovskim pristupom memoriji obično u vremenskom intervalu koji odgovara vremenu pristupa memorije generiše adrese za pristup određenom broju susednih memorijskih lokacija. Ti zahtevi će u ovom slučaju da budu za različite memorijske module i moći će da se realizuju paralelno.



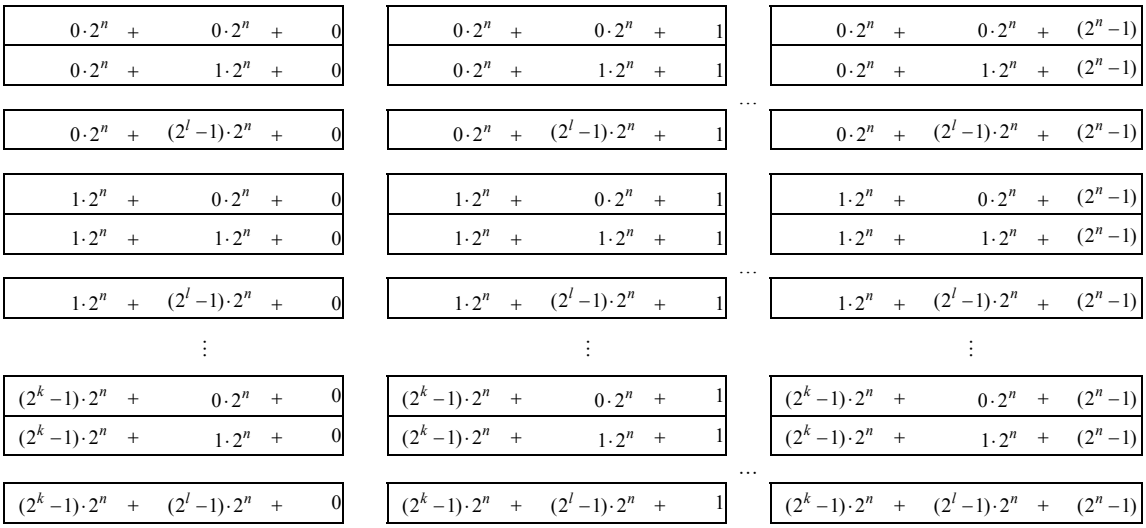
Slika 6 Konfiguracija računara za susedne u susednim

U slučaju rasporeda adresa mešovito struktura adrese i način formiranja broja modula i adrese u modulu su prikazani na slici 7. Sa *k* najstarijih i *n* najmlađih bitova adrese, pri čemu je $k+n=m$, kodira se broj modula, a sa *l* srednjih bitova adresa bajta u modulu. Raspored adresa po modulima je prikazan na slici 8. U nultom modulu nalaze se adrese $0\cdot 2^n+0\cdot 2^n+0$, $0\cdot 2^n+1\cdot 2^n+0$, $0\cdot 2^n+2\cdot 2^n+0$ i tako redom do $0\cdot 2^n+(2^l-1)\cdot 2^n+0$. U prvom modulu nalaze se adrese $0\cdot 2^n+0\cdot 2^n+1$, $0\cdot 2^n+1\cdot 2^n+1$, $0\cdot 2^n+2\cdot 2^n+1$ i tako redom do $0\cdot 2^n+(2^l-1)\cdot 2^n+1$. U drugom modulu nalaze se adrese $0\cdot 2^n+0\cdot 2^n+2$, $0\cdot 2^n+1\cdot 2^n+2$, $0\cdot 2^n+2\cdot 2^n+2$ i tako redom do $0\cdot 2^n+(2^l-1)\cdot 2^n+2$. Po analogiji sa ovim moguće je utvrditi i raspored adresa i po preostalim modulima do modula 2^n-1. U modulu 2^n-1 nalaze se adrese $0\cdot 2^n+0\cdot 2^n+(2^n-1)$, $0\cdot 2^n+1\cdot 2^n+(2^n-1)$, $0\cdot 2^n+2\cdot 2^n+(2^n-1)$ i tako redom do $0\cdot 2^n+(2^l-1)+(2^n-1)$. Ovim je pokriven opseg adresa iz nulte grupe od 2^n modula. Takvih grupa od 2^n modula sa istim načinom raspoređivanja adresa po modulima ima 2^k. U nultom modulu prve grupe nalaze se adrese $1\cdot 2^n+0\cdot 2^n+0$, $1\cdot 2^n+1\cdot 2^n+0$, $1\cdot 2^n+2\cdot 2^n+0$ i tako redom do $1\cdot 2^n+(2^l-1)\cdot 2^n+0$. U prvom modulu prve grupe nalaze se adrese $1\cdot 2^n+0\cdot 2^n+1$, $1\cdot 2^n+1\cdot 2^n+1$, $1\cdot 2^n+2\cdot 2^n+1$ i tako redom do $1\cdot 2^n+(2^l-1)\cdot 2^n+1$. U drugom modulu prve grupe nalaze se adrese $1\cdot 2^n+0\cdot 2^n+2$, $1\cdot 2^n+1\cdot 2^n+2$, $1\cdot 2^n+2\cdot 2^n+2$ i tako redom do $1\cdot 2^n+(2^l-1)\cdot 2^n+2$. Po analogiji sa ovim moguće je utvrditi i raspored adresa i po preostalim modulima do modula 2^n-1 grupe 1. U modulu 2^n-1 grupe 1 nalaze se adrese $1\cdot 2^n+0\cdot 2^n+(2^n-1)$, $1\cdot 2^n+1\cdot 2^n+(2^n-1)$, $1\cdot 2^n+2\cdot 2^n+(2^n-1)$ i tako redom do $1\cdot 2^n+(2^l-1)\cdot 2^n+(2^n-1)$. Ovim je pokriven opseg adresa iz prve grupe od 2^n modula. Na sličan način se raspoređuju adrese i po modulima preostalih grupa. Poslednja grupa od 2^n modula je (2^k-1)-va grupa. U nultom modulu (2^k-1)-ve grupe nalaze se adrese $(2^k-1)\cdot 2^n+0\cdot 2^n+0$, $(2^k-1)\cdot 2^n+1\cdot 2^n+0$, $(2^k-1)\cdot 2^n+2\cdot 2^n+0$ i tako redom do $(2^k-1)\cdot 2^n+(2^l-1)\cdot 2^n+0$. U prvom modulu (2^k-1)-ve grupe nalaze se adrese $(2^k-1)\cdot 2^n+0\cdot 2^n+1$, $(2^k-1)\cdot 2^n+1\cdot 2^n+1$, $(2^k-1)\cdot 2^n+2\cdot 2^n+1$ i tako redom do $(2^k-1)\cdot 2^n+(2^l-1)\cdot 2^n+1$. U drugom modulu (2^k-1)-ve grupe nalaze se adrese $(2^k-1)\cdot 2^n+0\cdot 2^n+2$, $(2^k-1)\cdot 2^n+1\cdot 2^n+2$, $(2^k-1)\cdot 2^n+2\cdot 2^n+2$ i tako redom do $(2^k-1)\cdot 2^n+(2^l-1)\cdot 2^n+2$. Po analogiji sa ovim moguće je utvrditi i raspored adresa i po preostalim modulima do modula (2^n-1) (2^k-1)-ve grupe. U modulu (2^n-1) (2^k-1)-ve grupe nalaze se adrese $(2^k-1)\cdot 2^n+0\cdot 2^n+(2^n-1)$,

$(2^k-1)\cdot 2^n+1\cdot 2^n+(2^n-1)$, $(2^k-1)\cdot 2^n+2\cdot 2^n+(2^n-1)$ i tako redom do $(2^k-1)\cdot 2^n+(2^l-1)\cdot 2^n+(2^n-1)$.



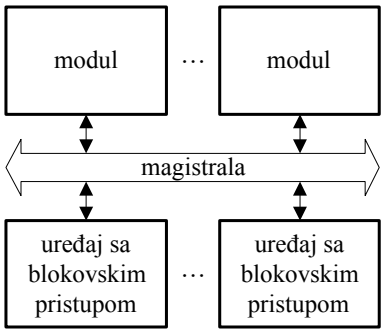
Slika 7 Broj modula i adresa u modulu za mešovito



Slika 8 Raspored adresa po modulima za mešovito

Ovim je totalni opseg adresa od 2^{l+k+n} bajta podeljen na 2^{k+n} modula kapaciteta od po 2^l bajta pri čemu je $k+n=m$. Međutim, moduli su sada grupisani u 2^k grupa sa po 2^n modula po grupi. Time je totalni opseg adresa podeljen na 2^k kontinualnih celina svaka kapaciteta 2^{l+n} bajta. Svaki takav opseg adresa od po 2^{l+n} bajta podeljen je dalje na 2^n modula kapaciteta od po 2^l bajta, ali tako da se opsezi susednih adresa od 0 do 2^n-1 nalaze u 2^n susednih modula. Takvih opsega adresa ima 2^l.

Ovakav način rasporeda adresa po modulima pogodan je za konfiguracije reačunara kao na slici 9. Uređaj sa blokovskim pristupom memoriji obično u nekom vremenskom intervalu generiše zahteve za pristup određenom broju susednih memorijskih lokacija iz određenog opsega adresa. Pošto tih uređaja ovde ima više moguće je podesiti da se opsezi adresa za uređaje sa blokovskim pristupom memoriji nalaze u različitim grupama memorijskih modula. Kako uređaji sa pojedinačnim pristupom rade nezavisno, to oni mogu da generišu zahteve za pristup memoriji istovremeno. Ti zahtevi će u ovom slučaju da budu za različite grupe memorijskih modula i moći će da se realizuju paralelno. S druge strane, uređaj sa blokovskim pristupom memoriji obično u vremenskom intervalu koji odgovara vremenu pristupa memorije generiše zahteve za pristup određenom broju susednih memorijskih lokacija. Ti zahtevi će u ovom slučaju da budu za različite memorijske module unutar date grupe modula i moći će da se realizuju paralelno.



Slika 9 Konfiguracija računara za mešovito

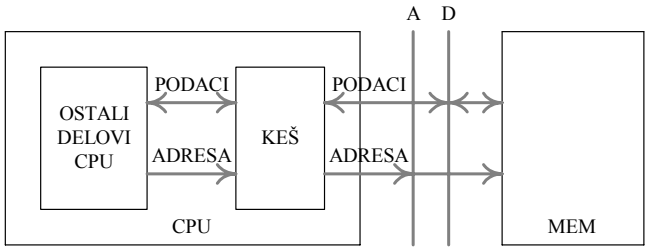
Raspored adresa po modulima susedne u istom i susedne u susednim su specijalni slučajevi za mešovito. Ako je $k = m$ i $n = 0$, tada mešovito postaje susedne u istom. Ako je, pak, $k = 0$ i $n = m$, tada mešovito postaje susedne u susednim.

5.2. KEŠ MEMORIJA

U ovom poglavlju se najpre daju osnovni pojmovi vezani za ubrzavanje pristupa memorijskim lokacijama korišćenjem keš memorije. Zatim se razmatraju osnovna pitanja vezana za realizaciju keš memorije, i to tehnika preslikavanja, zamena blokova keš memorije i ažuriranje sadržaja operativne memorije. Na kraju se daju neka specifična rešenja čijim se korišćenjem poboljšava rad keš memorija.

5.2.1. Osnovni pojmovi

Mehanizam keš memorije podrazumeva da u procesoru postoji posebna memorija, koja se naziva keš memorija (slika 1). Keš memorija se realizuje sa memorijskim komponentama čije je vreme pristupa daleko manje od vremena pristupa memorijskih komponenata operativne memorije i veoma blisko vremenu prenosa podataka između registara. Zbog toga je cena po bitu keš memorije daleko veća od cene po bitu operativne memorije. To ima za posledicu da je kapacitet keš memorije daleko manji od kapaciteta operativne memorije.



Slika 1 Asocijativno preslikavanje

Algoritam izvršavanja instrukcija u računarima sa keš memorijom je takav da se, pri svakom generisanju adrese operativne memorije od strane nekog dela procesora radi čitanja ili upisa, najpre, vrši provera da li se sadržaj sa generisane adrese nalazi u keš memoriji. Na početku dati sadržaj nije u keš memoriji. Zbog toga se sadržaj sa date i nekoliko susednih adresa, koje čine blok, najpre, prebacuje iz operativne u keš memoriju, pa se potom ili čita sadržaj iz keš memorije i prosleđuje onom delu procesora koji je adresu generisao ili upisuje u keš memoriju sadržaj iz onog dela procesora koje je adresu generisao.

Za nekoliko sledećih generisanih adresa operativne memorije verovatno će se i dalje utvrđivati da se sadržaji takođe ne nalaze u keš memoriji, pa će se odgovarajući blokovi dovlačiti iz operativne memorije u keš memoriju i sa keš memorijom realizovati čitanje ili upis. Time će se broj blokova operativne memorije dovučenih u keš memoriju povećavati. Zbog toga će se od određenog trenutka za neke od generisanih adresa operativne memorije utvrđivati da se sadržaji nalaze u keš memoriji. U svakoj takvoj situaciji sadržaj će se ili čitati iz keš memorije umesto iz operativne memorije ili upisivati u keš memoriju umesto u operativnu memoriju. S obzirom na to da je keš memorija daleko brža od operativne memorije, u svako takvoj situaciji čitanje ili upis sadržaja će se realizovati sa vremenom pristupa keš memorije umesto sa vremenom pristupa operativne memorije.

Efikasnost mehanizma keš memorije direktno zavisi od toga koliko često će se utvrđivati da se sadržaj sa generisane adrese nalazi u keš memoriji. Analiza tragova generisanih adresa prilikom izvršavanja tipičnih programa ukazuju na dva efekta koji direktno utiču na efikasnost mehanizma keš memorije. Prvi je da se jedanput generisana adresa obično posle toga još nekoliko puta generiše. Ovaj efekat se naziva vremenski lokalitet programa i javlja se kod

generisanja adresa instrukcija i skalarnih veličina u petlji. Drugi je da se posle neke generisane adrese veoma često generišu adrese koje slede sekvencijalno. Ovaj efekat se naziva prostorni lokalitet programa i javlja se kod sekvencijalnog izvršavanja instrukcija i sekvencijalnog pristupa podacima koji predstavljaju elemente vektora. Da bi se iskoristio potencijal prvof efekta potrebno je sadržaje memorijskih lokacija kojima se pristupa čuvati u keš memoriji sa sledeće pristupe, a da bi se iskoristio potencijal drugog efekta potrebno je dovlačiti ne samo sadržaj memorijske lokacije kojoj se pristupa, već blok od nekoliko susednih memorijskih lokacija.

Naziv za keš memoriju odražava činjenicu da je mehanizam keš memorije "skriven" od programera i da programer ne može programskim putem da utiče na ono što se dešava u keš memoriji. Provera da li se sadržaj sa generisane adrese nalazi u keš memoriji, eventualno dovlačenje bloka podataka iz operativne memorije u keš memoriju, čitanje podatka iz keš memorije i upis podatka u keš memoriju realizuju se kompletno hardverski. Programer je "svestan" postojanja mehanizma keš memorije indirektno na osnovu toga što će se program izvršavati brže u procesoru sa keš memorijom nego u procesoru bez keš memorije. Zbog toga mehanizam keš memorije ne pripada arhitekturi već organizaciji procesora.

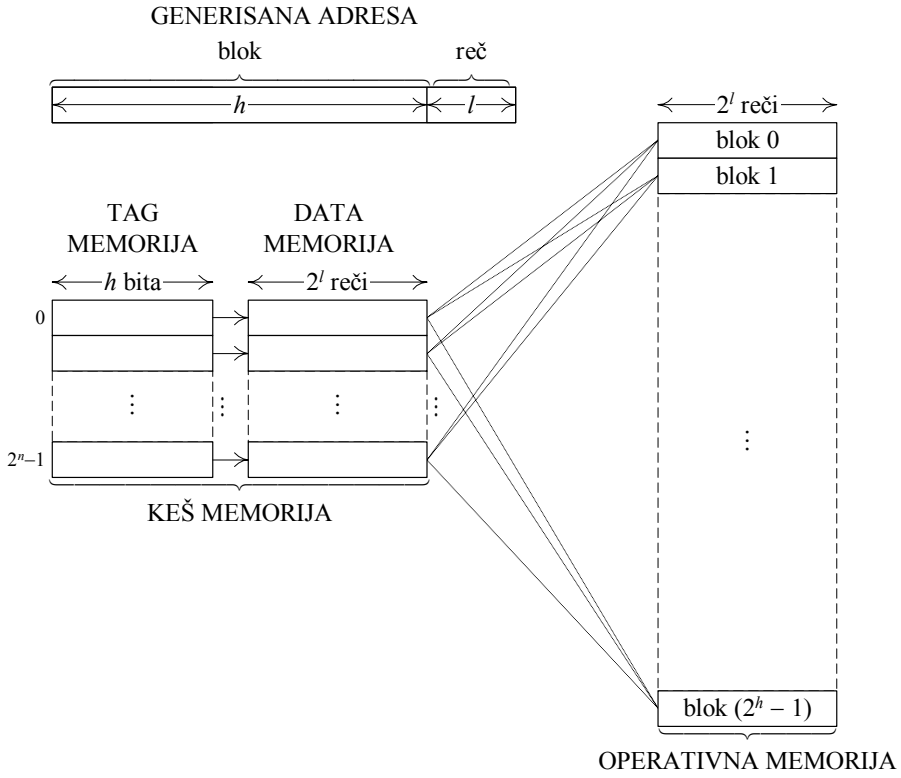
Prilikom realizacije keš memorije postoje tri osnovna pitanja koja treba rešiti. Kapacitet keš memorije je manji od kapaciteta operativne memorije i postoji potreba da se u keš memoriji vodi evedincija o tome koji se blokovi operativne memorije nalaze u keš memoriji i gde se nalaze u keš memoriji. Ovo vođenje evidencije se naziva tehnika preslikavanja. Kapacitet keš memorije je manji od kapaciteta operativne memorije, pa će se posle određenog vremena dešavati da se generišu adrese sa kojih se sadržaji ne nalaze u keš memoriji, a keš memorija je popunjena. Tada postoji potreba da se odluči koji će se blok izbaciti iz keš memorije da bi se u njoj stvorio prostor za dovlačenje bloka iz operativne memorije kome pripada generisana adresa. Ovo odlučivanje se realizuje prema nekom od algoritama zamene. Prilikom operacija upisa i utvrđivanja da u keš memoriji postoji blok kome pripada generisana adresa, upis će se realizovati u keš memoriju. Time se javlja razlika u vrednosti kopije sadržaja sa određene adrese u keš memoriji i u operativnoj memoriji. Sadržaji određenih lokacija operativne memorije se nalaze u keš memoriji samo privremeno radi ubrzanja pristupa. Zbog tog postoji potreba da se na neki način za sve generisane adrese za koje je upis izvršen u kopiju sadržaja u keš memoriji obezbedi ažuriranje i kopije sadržaja u operativnoj memoriji. Načini realizacije zavise od usvojene tehnike ažuriranja sadržaja operativne memorije. Ova tri pitanja su predmet razmatranja u sledeća tri poglavlja.

5.2.2. Tehnike preslikavanja

Tehnika preslikavanja određuje način vođenja evidencije o tome koji se blokovi operativne memorije nalaze u pojedinim blokovima keš memorije. Koriste se tri tehnike preslikavanja i to: asocijativno, direktno i set–asocijativno.

5.2.2.1. Asocijativno preslikavanje

Kod tehnike asocijativnog preslikavanja, keš memorija se sastoji iz DATA MEMORIJE i TAG MEMORIJE (slika 2). U DATA MEMORIJU se smeštaju blokovi preneti iz operativne u keš memoriju. U TAG MEMORIJU se, za blokove prenete iz operativne u keš memoriju, smeštaju brojevi blokova operativne memorije, koji se nazivaju *tag*–ovi. Za realizaciju DATA MEMORIJE i TAG MEMORIJE koriste se RAM memorija i asocijativna memorija, respektivno. Ukoliko u DATA MEMORIJU može da se smesti 2^n blokova, kaže se da keš memorija ima 2^n ulaza. Za svaki ulaz DATA MEMORIJE postoji odgovarajući ulaz TAG MEMORIJE.



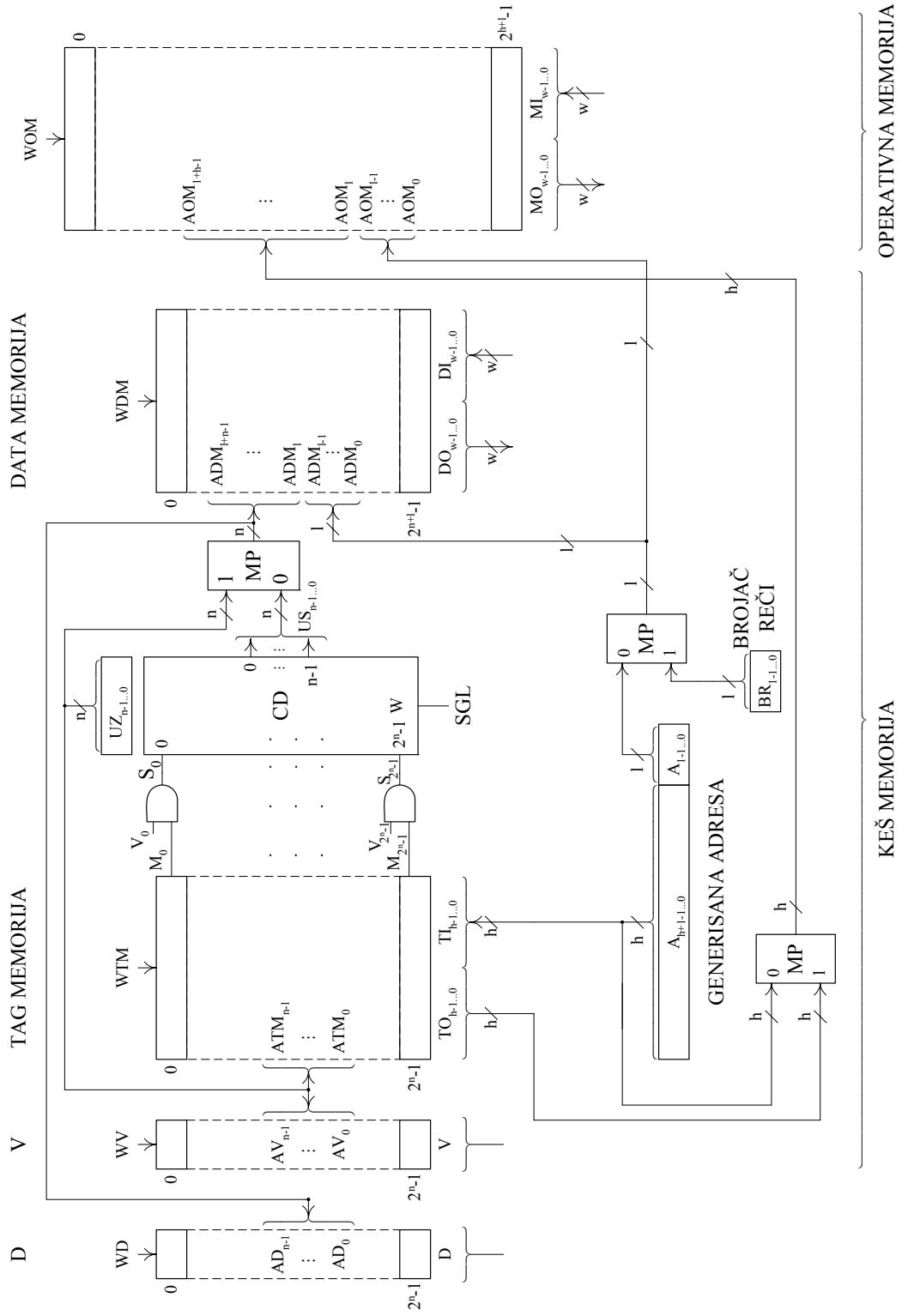
Slika 2 Asocijativno preslikavanje

U slučaju keš memorije realizovane u tehnici asocijativnog preslikavanja, zamišlja se kao da je operativna memorija podeljena na 2^h blokova. Ukoliko se uzme da je dimenzija bloka 2^l reči, tada generisana adresa ima sledeću strukturu: nižih l bitova određuju adresu reči unutar bloka i viših h bitova određuju broj bloka u operativnoj memoriji.

U tehnici asocijativnog preslikavanja bilo koji blok operativne memorije može da se smesti u bilo koji ulaz keš memorije, prilikom dovlačenja iz operativne u keš memoriju. Pošto u i–tom ulazu DATA MEMORIJE može da se nađe bilo koji od 2^h blokova operativne memorije, u i–tom ulazu TAG MEMORIJE se čuva broj bloka operativne memorije kome

pripada dati blok. S obzirom da u operativnoj memoriji ima 2^h blokova, širina memorijske reči TAG MEMORIJE je *h* bita.

Realizacija keš memorije sa asocijativnim preslikavanjem je data na slici 3.



Slika 3 Keš memorija sa asocijativnim preslikavanjem

S obzirom da je operativna memorija kapaciteta 2^{h+l} reči i da je veličina bloka 2^l reči, uzima se kao da je operativna memorija organizovana u 2^h blokova veličine 2^l reči. Stoga se adresa operativne memorije dužine h+l bita može podeliti na sledeći način: viših h bitova označavaju broj bloka i nižih l bitova označavaju adresu reči u bloku.

Pri generisanju zahteva za čitanje od strane procesora, viših h bitova generisane adrese vodi se na ulazne linije podataka $TI_{h-1...0}$ TAG MEMORIJE da bi se, njihovim istovremenim upoređivanjem sa sadržajima svih 2^n ulaza TAG MEMORIJE, utvrdilo da li postoji saglasnost sa sadržajem nekog ulaza. Za svaki od 2^n ulaza TAG MEMORIJE postoji poseban signal saglasnosti M_0 do $M_2{}^n{}_{-1}$, koji svojom aktivnom vrednošću određuje da je na datom ulazu otkrivena saglasnost. Otkrivena saglasnost je važeća, i jedan od signala važećih saglasnosti S_0 do $S_2{}^n{}_{-1}$ je aktivan, ukoliko je odgovarajući indikator važećih ulaza V aktivan. Signal saglasnosti SGL, koji se dobija sa izlaza W kodera CD, ima aktivnu vrednost, ukoliko jedan od signala S_0 do $S_2{}^n{}_{-1}$ ima aktivnu vrednost. Binarna vrednost broja ulaza u kome je otkrivena saglasnost je određena sa n bita sa izlaza kodera CD na osnovu signala S_0 do $S_2{}^n{}_{-1}$. Ukoliko postoji saglasnost, sa n bitova sa izlaza kodera CD i l nižih bitova generisane adrese, adresira se reč DATA MEMORIJE i obavlja čitanje.

Ukoliko saglasnost ne postoji u jedan od ulaza keš memorije treba iz operativne memorije da se dovuče blok u kome se nalazi željeni sadržaj. Ulaz u koji se dovlači blok naziva se ulaz za zamenu. Broj ulaza je određen vrednošću $UZ_{n-1...0}$ do koje se dolazi na osnovu nekog od algoritama zamene. Pre dovlačenja željenog bloka proverava se da li se u ulazu koji je odabran za zamenu nalazi blok koji je nekom od operacija upisa modifikovan. Ovo se utvrđuje na osnovu sadržaja indikatora D adresiranog vrednošću $UZ_{n-1...0}$. Ukoliko je indikator D ulaza odabranog za zamenu 1, blok je modifikovan, pa prvo treba dati blok vratiti u operativnu memoriju pa tek onda dovući željeni blok. Ukoliko je indikator D ulaza odabranog za zamenu 0, blok nije modifikovan, pa se željeni blok odmah dovlači.

Prilikom vraćanja bloka odabranog za zamenu 2^l reči datog bloka se čita iz DATA MEMORIJE sa adresa formiranih od vrednosti $UZ_{n-1...0}$ koja daje n starijih bitova adrese i vrednosti $BR_{l-1...0}$ koja je daje l mlađih bitova adrese. Date reči se upisuju u OPERATIVNU MEMORIJU na adresama formiranih od vrednosti na linijama $TO_{h-1...0}$ koja daje n starijih bitova adrese i vrednosti $BR_{l-1...0}$ koja je daje l mlađih bitova adrese. Vrednost na linijama $TO_{h-1...0}$ pročitana je iz TAG MEMORIJE sa adrese određene vrdnošću $UZ_{n-1...0}$.

Prilikom dovlačenja željenog bloka 2^l reči datog bloka se upisuje u DATA MEMORIJU na adresama formiranih kao i u slučaju vraćanja bloka od vrednosti $UZ_{n-1...0}$ koja daje n starijih bitova adrese i vrednosti $BR_{l-1...0}$ koja je daje l mlađih bitova adrese. Date reči se čitaju iz OPERATIVNE MEMORIJE sa adresa formiranih od vrednosti bitova $A_{h-1...0}$ generisane adrese koja daje n starijih bitova adrese i vrednosti $BR_{l-1...0}$ koja je daje l mlađih bitova adrese. Pored toga bitovi $A_{h-1...0}$ generisane adrese se upisuju u ulaz TAG MEMORIJE čija je adresa određena vrednošću $UZ_{n-1...0}$. Indikatori V i D ulaza adresiranih vrednošću $UZ_{n-1...0}$ postavljaju se na 1 i 0, respektivno.

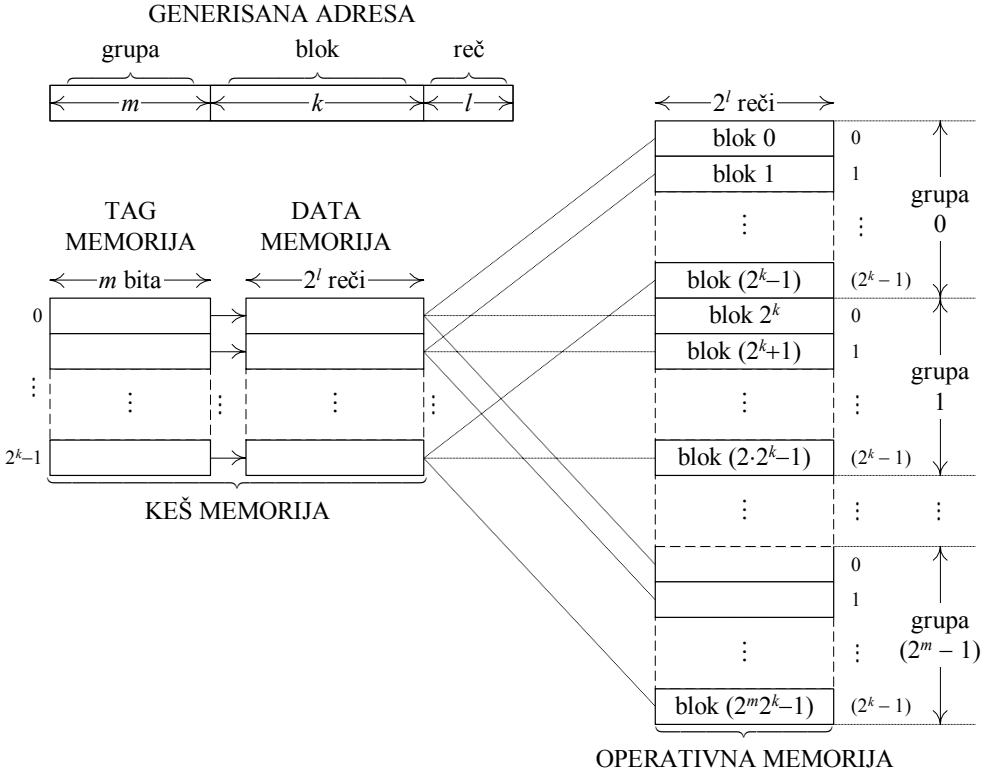
Po dovlačenju bloka podataka u keš memoriju, utvrđuje se da sada postoji saglasnost, pa se na već opisani način adresira reč DATA MEMORIJE i obavlja čitanje.

Pri generisanju zahteva za upis od strane procesora, na isti način se ispituje saglasnost sa sadržajem keš memorije kao u slučaju operacije čitanja. Ukoliko postoji saglasnost, na isti način se adresira reč DATA MEMORIJE i vrši upis, pri čemu se sada indikator modifikovanog ulaza D, adresiran vrednošću $US_{n-1...0}$ sa izlaza kodera CD i koja predstavlja broj ulaza u kome je otkrivena saglasnost, postavlja na 1.

Ako saglasnost ne postoji, na identičan način kao i za operaciju čitanja, se, najpre, blok iz ulaza keš memorije, određenog vrednošću $UZ_{n-1...0}$, vraća u operativnu memoriju, ukoliko je modifikovan, a zatim, u isti ulaz keš memorije, dovlači novi blok iz operativne memorije. Potom se ponovo, na već opisani način, vrši provera da li postoji saglasnost, utvrđuje da postoji saglasnost i realizuje upis.

5.2.2.2. Direktno preslikavanje

Kod tehnike direktnog preslikavanja, keš memorija se sastoji iz DATA MEMORIJE i TAG MEMORIJE (slika 4). U DATA MEMORIJU se smeštaju blokovi preneti iz operativne u keš memoriju. U TAG MEMORIJU se, za blokove prenete iz operativne u keš memoriju, smeštaju brojevi grupa operativne memorije, koji se nazivaju *tag*–ovi. Za realizaciju DATA MEMORIJE i TAG MEMORIJE, koristi se RAM memorija. Ukoliko u DATA MEMORIJU može da se smesti 2^k blokova operativne memorije, kaže se da keš memorija ima 2^k ulaza. Za svaki ulaz DATA MEMORIJE postoji odgovarajući ulaz TAG MEMORIJE.



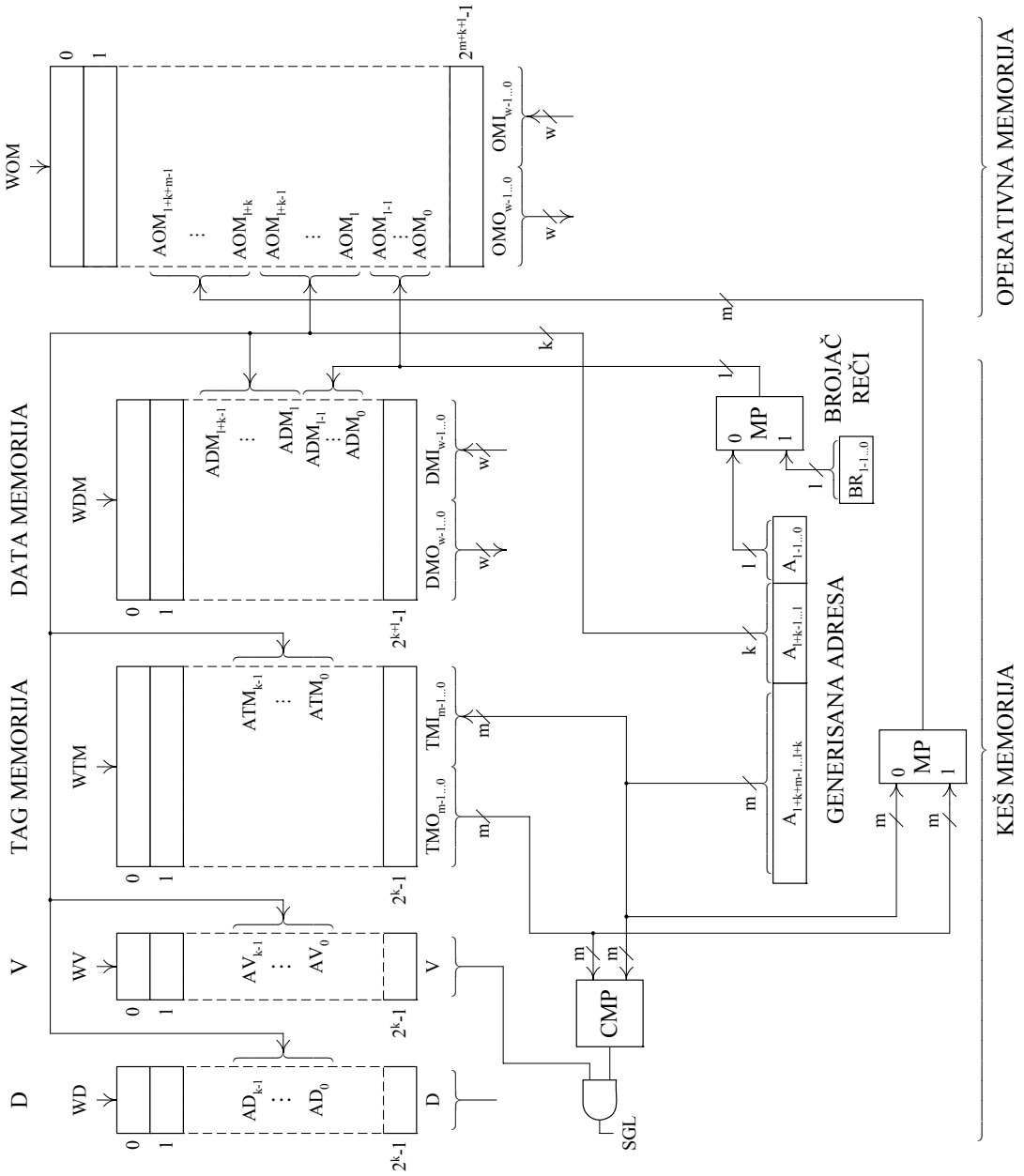
Slika 4 Direktno preslikavanje

U slučaju keš memorije sa 2^k ulaza realizovane u tehnici direktnog preslikavanja, zamišlja se kao da je operativna memorija kapaciteta 2^{m+k} blokova podeljena na 2^m grupa, pri čemu veličina grupe odgovara veličini DATA MEMORIJE i iznosi 2^k blokova. Ukoliko se uzme da je dimenzija bloka 2^l reči, tada generisana adresa ima sledeću strukturu: najnižih l bitova određuju adresu reči unutar bloka, srednjih k bitova određuju broj bloka unutar grupe i najviših m bitova određuju broj grupe u operativnoj memoriji kojoj pripada blok.

U tehnici direktnog preslikavanja svaki blok operativne memorije ima fiksno definisan ulaz u keš memoriji u koji se smešta prilikom dovlačenja iz operativne u keš memoriju. Tako i–ti blok iz bilo koje od 2^m grupa operativne memorije prilikom dovlačenja može da se smesti jedino u i–ti ulaz DATA MEMORIJE. Pošto u i–tom ulazu DATA MEMORIJE može da se

nađe *i*–ti blok iz bilo koje od 2^m grupa operativne memorije, u *i*–tom ulazu TAG MEMORIJE se čuva broj grupe kojoj pripada dati blok. S obzirom da u operativnoj memoriji ima 2^m grupa, širina memorijske reči TAG MEMORIJE je *m* bitova.

Realizacija keš memorije sa direktnim preslikavanjem je data na slici 5.



Slika 5 Keš memorija sa direktnim preslikavanjem

S obzirom da je operativna memorija kapaciteta 2^{m+k+l} reči, da je veličina bloka 2^l reči i da keš memorija ima 2^k ulaza, uzima se kao da je operativna memorija organizovana u 2^m grupa, od kojih svaka sadrži 2^k blokova veličine 2^l reči. Stoga se adresa operativne memorije dužine m+k+l bita može podeliti na sledeći način: najviših m bitova označavaju broj grupe, srednjih k bitova označavaju broj bloka i najnižih l bitova označavaju adresu reči u bloku.

Pri generisanju zahteva za čitanje od strane procesora istovremeno se pristupa i TAG MEMORIJI i DATA MEMORIJI. Srednjih k bitova generisane adrese određuju ulaz u TAG MEMORIJU i selektuju jedan od indikatora važećih ulaza V. Sadržaj $TMO_{m-1...0}$ adresirane lokacije TAG MEMORIJE poredi se sa najviših m bitova generisane adrese $A_{l+k+m-1...l+k}$. Ukoliko su sadržaji identični i selektovani indikator V je aktivan, u keš memoriji je otkrivena saglasnost i signal SGL je aktivan. Istovremeno se srednjih k i najnižih l bitova generisane adrese koriste za adresiranje i čitanje reči DATA MEMORIJE. Ukoliko postoji saglasnost, željeni blok je u keš memoriji i očitani podatak je dobar.

Ukoliko saglasnost ne postoji očitani podatak nije dobar i u taj isti ulaz keš memorije za koji je utvrđeno da ne postoji saglasnost treba iz operativne memorije da se dovuče blok u kome se nalazi željeni sadržaj. Pre dovlačenja željenog bloka proverava se da li se u datom ulazu nalazi blok koji je nekom od operacija upisa modifikovan. Ovo se utvrđuje na osnovu sadržaja indikatora D adresiranog sa srednjih k bitova generisane adrese $A_{l+k-1...l}$. Ukoliko je indikator D datog ulaza 1, blok je modifikovan, pa prvo treba dati blok vratiti u operativnu memoriju pa tek onda dovući željeni blok. Ukoliko je indikator D datog ulaza 0, blok nije modifikovan, pa se željeni blok odmah dovlači.

Prilikom vraćanja bloka odabranog za zamenu 2^l reči datog bloka se čita iz DATA MEMORIJE sa adresa formiranih od srednjih k bitova generisane adrese $A_{l+k-1...l}$ koji daju k starijih bitova adrese i vrednosti $BR_{l-1...0}$ koja je daje l mlađih bitova adrese. Date reči se upisuju u OPERATIVNU MEMORIJU na adresama formiranih od vrednosti na linijama $TO_{m-1...0}$ koja daje m starijih bitova adrese, srednjih k bitova generisane adrese $A_{l+k-1...l}$ koji daju k srednjih bitova adrese i vrednosti $BR_{l-1...0}$ koja je daje l mlađih bitova adrese. Vrednost na linijama $TO_{m-1...0}$ pročitana je iz TAG MEMORIJE sa adrese određene sa srednjih k bitova generisane adrese $A_{l+k-1...l}$.

Prilikom dovlačenja željenog bloka 2^l reči datog bloka se upisuje u DATA MEMORIJU na adresama formiranih kao i u slučaju vraćanja bloka od srednjih k bitova generisane adrese $A_{l+k-1...l}$ koji daju k starijih bitova adrese i vrednosti $BR_{l-1...0}$ koja je daje l mlađih bitova adrese. Date reči se čitaju iz OPERATIVNE MEMORIJE sa adresa formiranih od bitova $A_{l+k+m-1...l+k}$ generisane adrese koji daju m starijih bitova adrese, srednjih k bitova generisane adrese $A_{l+k-1...l}$ koji daju k srednjih bitova adrese i bitova $BR_{l-1...0}$ koji daje l mlađih bitova adrese. Pored toga bitovi $A_{l+k+m-1...l+k}$ generisane adrese se upisuju u ulaz TAG MEMORIJE čija je adresa određena sa srednjih k bitova generisane adrese $A_{l+k-1...l}$. Indikatori V i D ulaza adresiranih sa $A_{l+k-1...l}$ postavljaju se na 1 i 0, respektivno.

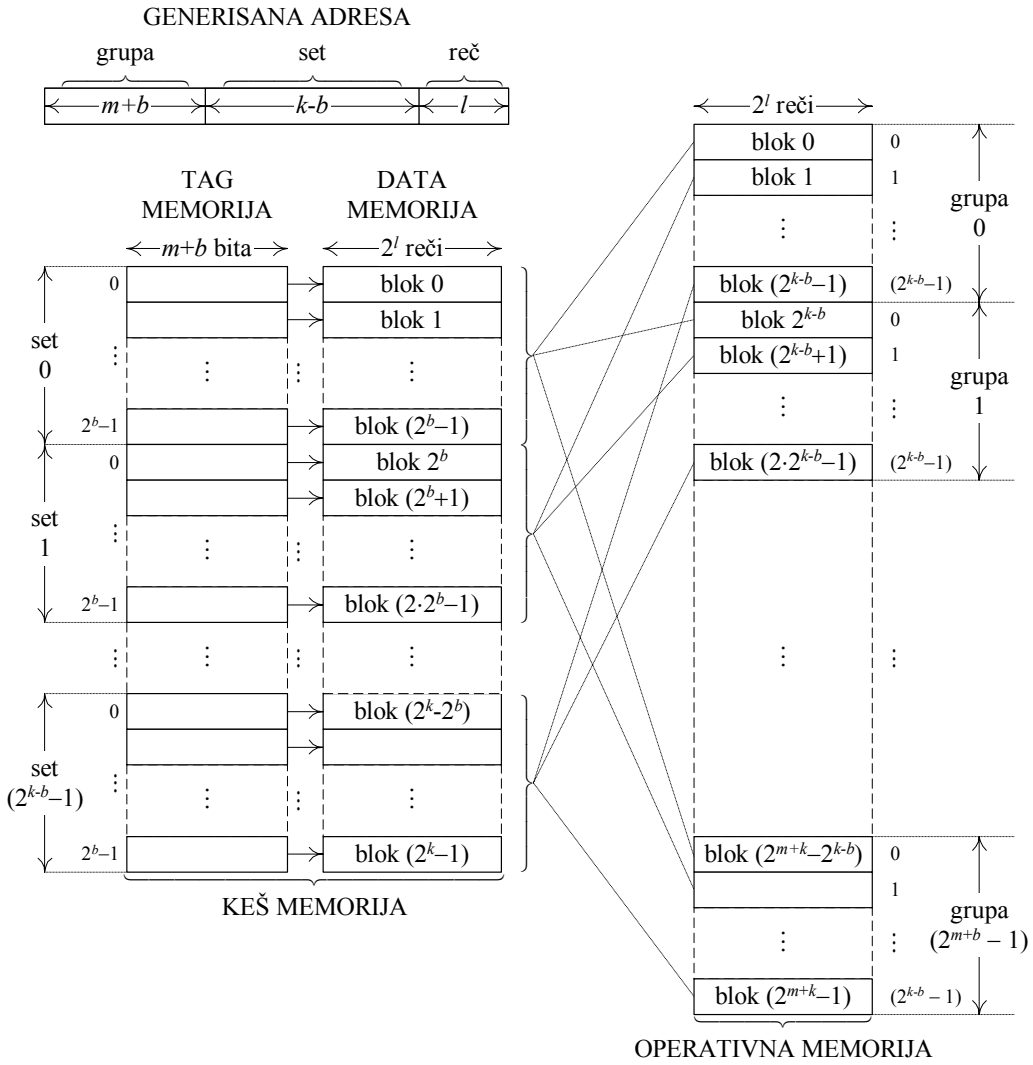
Po dovlačenju bloka podataka u keš memoriju, utvrđuje se da sada postoji saglasnost, pa se na već opisani način adresira reč DATA MEMORIJE i obavlja čitanje.

Pri generisanju zahteva za upis od strane procesora, na isti način se ispituje saglasnost sa sadržajem keš memorije kao u slučaju operacije čitanja. Ukoliko postoji saglasnost, na isti način se adresira reč DATA MEMORIJE i vrši upis, pri čemu se sada indikator modifikovanog ulaza D, adresiran sa srednjih k bitova generisane adrese $A_{l+k-1...l}$, postavlja na 1. Za razliku od zahteva za čitanje kada se istovremeno i utvrđivala saglasnost i čitao podatak, kod zahteva za upis se najpre utvrđuje saglasnot pa tek onda upisuje podatak.

Ako saglasnost ne postoji, na identičan način kao i za operaciju čitanja, se, najpre, blok iz ulaza keš memorije, određenog sa srednjih k bitova generisane adrese $A_{l+k-1...l}$, vraća u operativnu memoriju, ukoliko je modifikovan, a zatim, u isti ulaz keš memorije, dovlači novi blok iz operativne memorije. Potom se ponovo, na već opisani način, vrši provera da li postoji saglasnost, utvrđuje da postoji saglasnost i realizuje upis.

5.2.2.3. Set-asocijativno preslikavanje

Tehnika set–asocijativnog preslikavanja predstavlja kombinaciju prethodne dve tehnike preslikavanja. Kod set–asocijativnog preslikavanja se može uzeti da se keš memorija sastoji od 2^b identičnih manjih keš memorija realizovanih tehnikom direktnog preslikavanja i da je operativna memorija podeljena na više grupa. Pri tome, broj blokova u grupi operativne memorije odgovara broju blokova jedne manje keš memorije. Pošto je uzeto da se keš memorija sastoji od 2^b manjih keš memorija, *i*–ti blok bilo koje grupe operativne memorije može da se preslika u *i*–ti blok bilo koje od 2^b manjih keš memorija. Time je stvoreno onoliko setova keš memorije koliko ima blokova u grupi operativne memorije, sa onoliko blokova po setu keš memorije koliko ima manjih keš memorija. Na nivou seta preslikavanje je direktno, jer je brojem bloka u grupi operativne memorije jednoznačno određen set keš memorije u koji se dati blok preslikava. Unutar seta preslikavanje je asocijativno, jer se dati blok operativne memorije može smestiti u bilo koji od 2^b blokova seta keš memorije (slika 6).



Slika 6 Set–asocijativno preslikavanje

Kod tehnike set–asocijativnog preslikavanja, keš memorija se sastoji iz 2^b manjih keš memorija i to keš memorije blokova 0, u kojoj se za sve setove čuvaju samo blokovi 0, keš memorije blokova 1, u kojoj se za sve setove čuvaju samo blokovi 1, i tako redom do keš

memorije blokova (2^b–1), u kojoj se za sve setove čuvaju samo blokovi (2^b–1). Svaka od 2^b manjih keš memorija se sastoji od DATA MEMORIJE i TAG MEMORIJE. U DATA MEMORIJE se smeštaju blokovi preneti iz operativne u keš memoriju, a u TAG MEMORIJE se smeštaju brojevi grupa, koji se nazivaju *tag*–ovi, za blokove prenete iz operativne u keš memoriju. Za realizaciju DATA MEMORIJA i TAG MEMORIJA, koristi se RAM memorija. Ukoliko u DATA MEMORIJU može da se smesti 2^{k-b} blokova operativne memorije, kaže se da keš memorija ima 2^{k-b} setova. Za svaki set postoji poseban ulaz u svakoj od DATA MEMORIJA i TAG MEMORIJA. DATA MEMORIJE i TAG MEMORIJE 2^b manjih keš memorija obrazuju DATA MEMORIJU i TAG MEMORIJU keš memorije sa set–asocijativnim preslikavanjem.

U slučaju keš memorije sa 2^{k-b} setova realizovane u tehnici set–asocijativnog preslikavanja, zamišlja se kao da je operativna memorija kapaciteta 2^{m+k} blokova podeljena na 2^{m+b} grupa, pri čemu veličina grupe odgovara veličini DATA MEMORIJE manje keš memorije i iznosi 2^{k-b} blokova. Ukoliko se uzme da je dimenzija bloka 2^l reči, tada generisana adresa ima sledeću strukturu: najnižih *l* bitova određuju adresu reči unutar bloka, srednjih (*k*–*b*) bitova određuju broj bloka unutar grupe operativne memorije i broj seta u keš memoriji i najviših (*m*+*b*) bitova određuju broj grupe u operativnoj memoriji kojoj pripada blok.

U ovoj tehnici svaki blok operativne memorije ima fiksno definisan set u keš memoriji u koji se smešta prilikom dovlačenja iz operativne u keš memoriju. Tako *j*–ti blok iz bilo koje od 2^{m+b} grupa operativne memorije, prilikom dovlačenja, može da se smesti jedino u *j*–ti set keš memorije. Pošto u bilo kom od 2^b blokova *j*–tog seta može da se nađe *j*–ti blok iz bilo koje od 2^{m+b} grupa operativne memorije, u *j*–tom ulazu jedne od 2^b TAG MEMORIJA manjih keš memorija se čuva broj grupe kojoj pripada dati *j*–ti blok. S obzirom da u operativnoj memoriji ima 2^{m+b} grupa, širina memorijske reči TAG MEMORIJA manjih keš memorija je (*m*+*b*) bitova. Totalni kapacitet keš memorije je, kao i u slučaju direktnog preslikavanja, 2^k blokova, ali su sada oni organizovani u 2^{k-b} setova sa 2^b blokova po setu.

Realizacija keš meorije sa set-asocijativnim preslikavanjem je data na slici 7.

S obzirom da je operativna memorija kapaciteta 2^{m+k+1} reči, da je veličina bloka 2^1 reči i da keš memorija ima 2^k blokova organizovanih u 2^{k-b} seta sa po 2^b ulaza po setu, uzima se kao da je operativna memorija organizovana u 2^{m+b} grupa, od kojih svaka sadrži 2^{k-b} blokova veličine 2^1 reči. Stoga se adresa operativne memorije dužine m+k+l bita može podeliti na sledeći način: najviših m+b bitova označavaju broj grupe, srednjih k-b bitova označavaju broj seta i najnižih lbitova označavaju adresu reči u bloku.

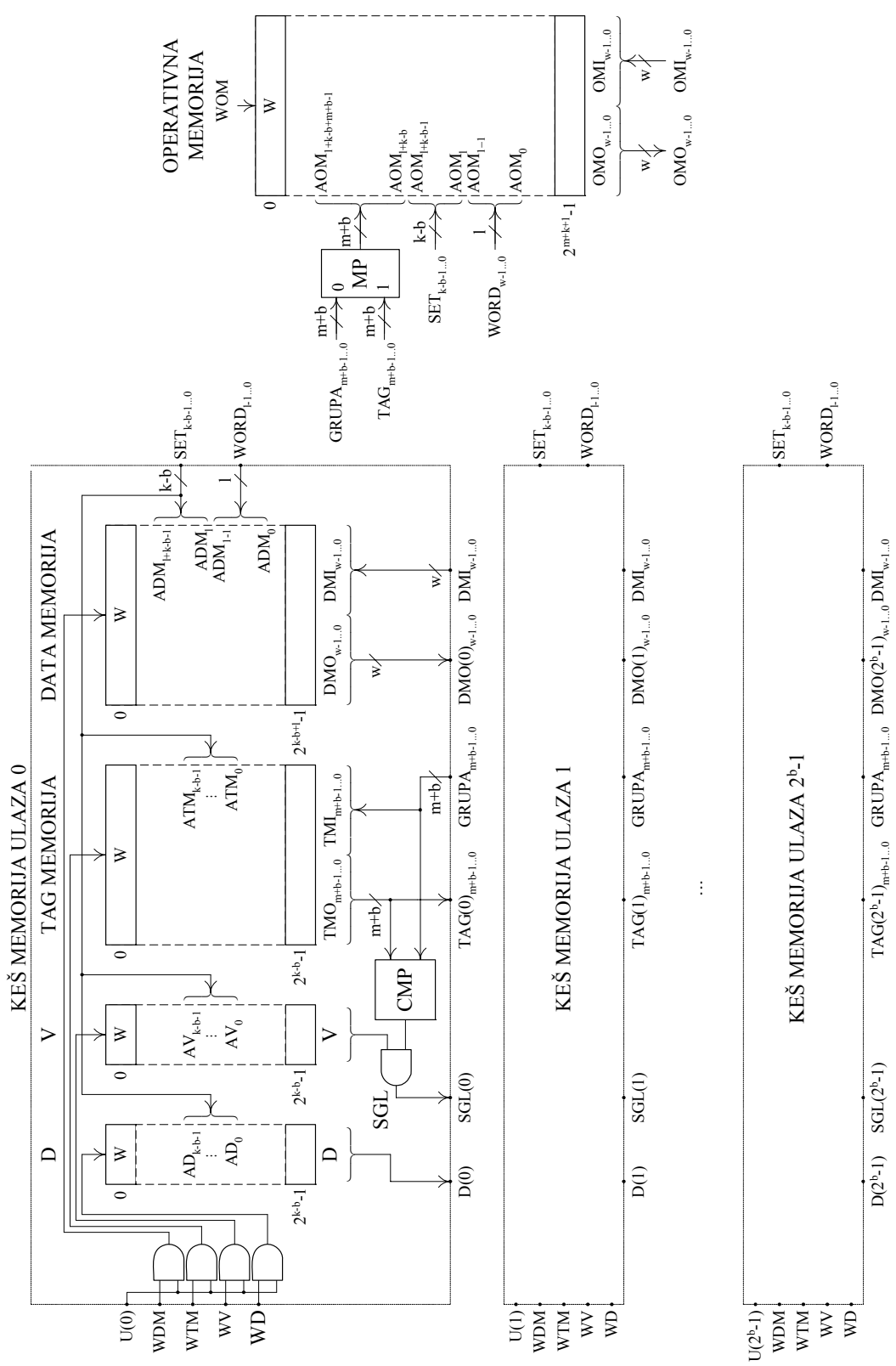
Pri generisanju zahteva za čitanje vrši se provera, i to na isti način kao i u slučaju direktnog preslikavanja, da li u nekoj od TAG MEMORIJA keš memorija ulaza 0 do 2^b-1 postoji saglasnost za set određen generisanom adresom. Zato se iz TAG MEMORIJA keš memorija ulaza 0 do 2^b-1 sa adrese određene sa k-b bitova $A_{l+k-b-1...l}$ broja seta generisane adrese istovremeno čitaju brojevi grupa $DOTM_{m+b-1...0}$ i upoređuju sa m+b bitova $A_{l+k-b+m+b-1...l+k-b}$ broja grupe generisane adrese. Takođe se sa iste adrese iz V indikatora keš memorija ulaza 0 do 2^b-1 čitaju V indikatori svih 2^b ulaza datog seta. Signali saglasnosti SGL(0) do SGL(2^b-1) keš memorija ulaza 0 do 2^b-1 se formiraju na osnovu signala jednakosti sa izlaza komparatora CMP i V indikatora svih 2^b ulaza datog seta. Signal saglasnosti cele keš memorije SGL je ILI funkcija signala saglasnosti SGL(0) do SGL(2^b-1) keš memorija ulaza 0 do 2^b-1, respektivno. Istovremeno se, i to na isti način kao u slučaju direktnog preslikavanja, iz DATA MEMORIJA keš memorija ulaza 0 do 2^b-1 sa adrese određene sa k-b bitova $A_{l+k-b-1...l}$ broja seta generisane adrese i l bitova $A_{l-1...0}$ adrese reči u bloku čitaju sadržaji $DMO(0)_{w-1...0}$ do $DMO(2^b-1)_{w-1...0}$. U slučaju da je signal saglasnosti SGL aktivan, formiraju se, i to na osnovu signala saglasnosti

SGL(0) do SGL(2^b-1) keš memorija ulaza 0 do 2^b-1, signali selekcije ulaza U(0) do U(2^b-1) za keš memorije ulaza 0 do 2^b-1, respektivno. Njima se, kao očitani sadržaj $DMO_{w-1...0}$ iz cele keš memorije, selektuje jedan od očitanih sadržaja $DMO(0)_{w-1...0}$ do $DMO(2^b-1)_{w-1...0}$.

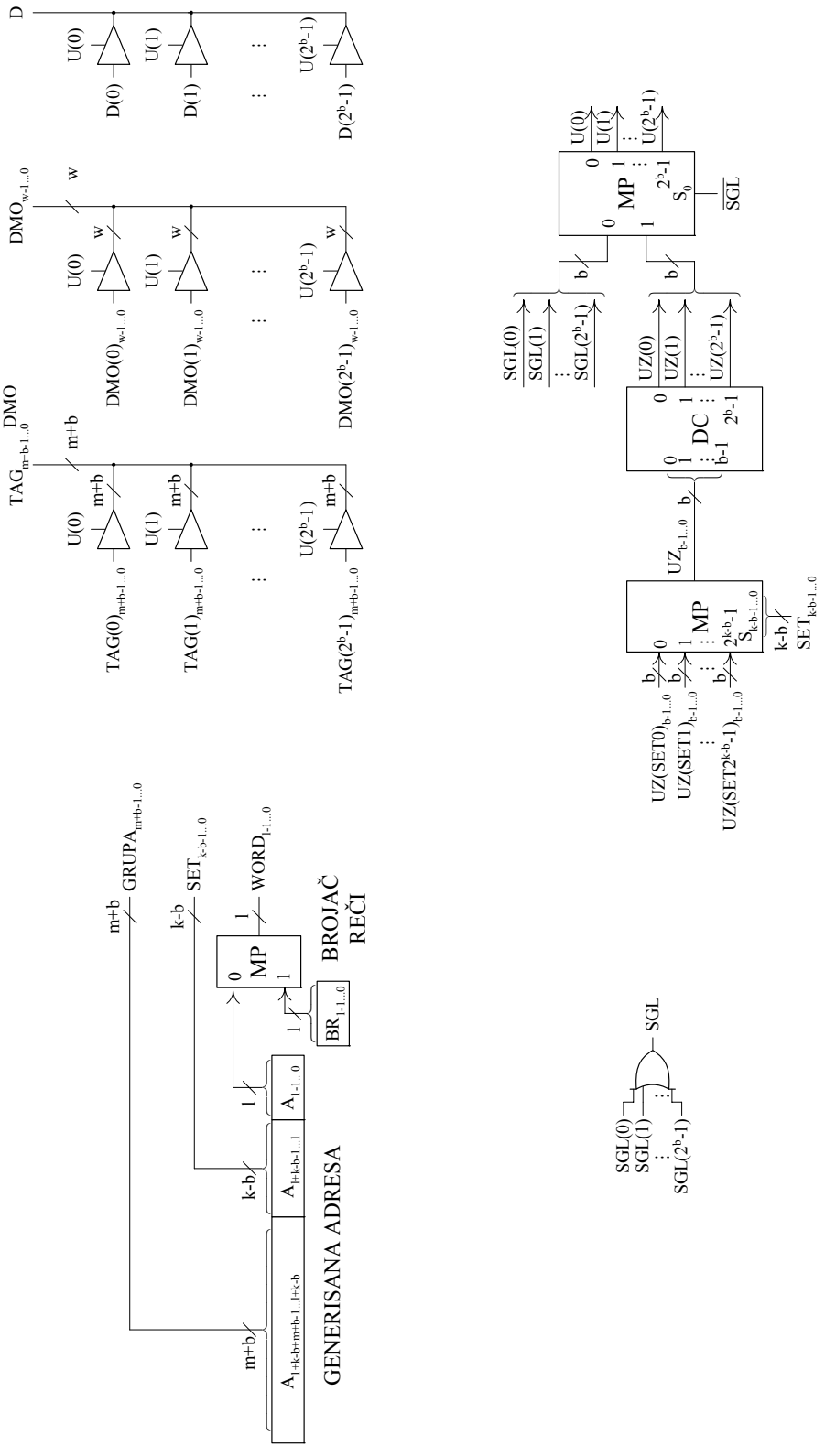
Ako saglasnost ne postoji ni u jednom od 2^b blokova seta određenog sa k-b bitova $A_{l+k-b-1...l}$ broja seta generisane adrese, onda se, na osnovu sadržaja logike za realizaciju nekog od algoritama zamene, za zamenu bira jedan od 2^b blokova datog seta. Algoritam zamene se realizuje nezavisno za svaki od 2^{k-b} setova i brojevi ulaza za zamenu za svaki od 2^{k-b} setova se nalaze u registrima zamene $UZ(SET0)_{b-1...0}$ do $UZ(SET2^{k-b}-1)_{b-1...0}$. Sadržaj jednog od ovih registara se selektuje sa k-b bitova broja seta generisane adrese na izlaznim linijama $UZ_{b-1...0}$ multipleksera. Na osnovu sadržaja $UZ_{b-1...0}$ se na izlazima dekodera DC formira vrednost 1 jednog od signala ulaza za zamenu UZ(0) do UZ(2^b-1) za keš memorije ulaza 0 do 2^b-1, respektivno. Ovi signali se selektuju kroz multiplekser MP za slučaj da nema saglasnosti i pojavljuju kao signali selekcije U(0) do U(2^b-1) za keš memorije ulaza 0 do 2^b-1.

Pre dovlačenja željenog bloka iz operativne memorije u keš memoriju proverava se da li se u ulazu seta koji je odabran za zamenu nalazi blok koji je nekom od operacija upisa modifikovan. Ovo se utvrđuje na osnovu sadržaja njegovog indikatora D. Ukoliko je indikator D datog ulaza 1, blok je modifikovan, pa prvo treba dati blok vratiti u operativnu memoriju pa tek onda dovući željeni blok. Ukoliko je indikator D datog ulaza 0, blok nije modifikovan, pa se željeni blok odmah dovlači. Vrednost indikatora D se dobija na osnovu signala selekcije ulaza U(0) do U(2^b-1) i signala indikatora D(0) do D(2^b-1) za keš memorije ulaza 0 do 2^b-1. Signali D(0) do D(2^b-1) predstavljaju indikatore D za blokove iz ulaza 0 do 2^b-1 seta određenog sa k-b bitova $A_{l+k-b-1...l}$ broja seta generisane adrese.

Prilikom vraćanja 2^l reči bloka odabranog za zamenu iz DATA MEMORIJA keš memorija ulaza 0 do 2^b-1 sa adrese određene sa k-b bitova $A_{l+k-b-1...l}$ broja seta generisane adrese i l bitova brojača $BR_{l-1...0}$ adrese reči u bloku se istovremeno čitaju sadržaji $DMO(0)_{w-1...0}$ do $DMO(2^b-1)_{w-1...0}$. Kao očitani sadržaj $DMO_{w-1...0}$ iz cele keš memorije se, na osnovu signala selekcije ulaza U(0) do U(2^b-1) za keš memorije ulaza 0 do 2^b-1, selektuje jedan od očitanih sadržaja $DMO(0)_{w-1...0}$ do $DMO(2^b-1)_{w-1...0}$. Date reči se upisuju u OPERATIVNU MEMORIJU na adresama formiranih od vrednosti na linijama $TAG_{m+b-1...0}$ koja daje m+b starijih bitova adrese, srednjih k bitova generisane adrese $A_{l+k-1...l}$ koji daju k srednjih bitova adrese i vrednosti $BR_{l-1...0}$ koja je daje l mlađih bitova adrese. Pri tome se iz DATA MEMORIJA keš memorija ulaza 0 do 2^b-1 sa adrese određene sa k-b bitova $A_{l+k-b-1...l}$ broja seta generisane adrese istovremeno čitaju sadržaji $TAG(0)_{m+b-1...0}$ do $TAG(2^b-1)_{m+b-1...0}$. Kao očitani sadržaj $TAG_{m+b-1...0}$ iz cele keš memorije se, na osnovu signala selekcije ulaza U(0) do U(2^b-1) za keš memorije ulaza 0 do 2^b-1, selektuje jedan od očitanih sadržaja $TAG(0)_{m+b-1...0}$ do $TAG(2^b-1)_{m+b-1...0}$.



Slika 7 Keš memorija sa set asocijativnim preslikavanjem



Slika 7 Keš memorija sa set asocijativnim preslikavanjem (nastavak)

Prilikom dovlačenja željenog bloka 2^l reči datog bloka se dovode na ulazne linije podataka DATA MEMORIJA keš memorija ulaza 0 do 2^b-1. Adrese DATA MEMORIJA su formirane kao i u slučaju vraćanja bloka od k-b bitova $A_{l+k-b-1...l}$ broja seta generisane adrese i l bitova brojača $BR_{l-1...0}$ adrese reči u bloku. Date reči se čitaju iz OPERATIVNE MEMORIJE sa adresa formiranih od bitova $A_{l+k-b+m+b-1...l+k-b}$ generisane adrese koji daju m+b starijih bitova adrese, srednjih k-b bitova generisane adrese $A_{l+k-b-1...l}$ koji daju k-b srednjih bitova adrese i bitova $BR_{l-1...0}$ koji daje l mlađih bitova adrese. Upis se realizuje samo u jednu od 2^b DATA MEMORIJA i to onu određenu aktivnom vrednošću jednog od signala selekcije ulaza U(0) do U(2^b-1). Pored toga bitovi $A_{l+k-b+m+b-1...l+k-b}$ i bitovi $A_{l+k-b-1...l}$ generisane adrese se vode na ulazne linije podataka i adresne linije TAG MEMORIJA keš memorija ulaza 0 do 2^b-1. Upis se realizuje samo u jednu od 2^b TAG MEMORIJA i to onu određenu aktivnom vrednošću jednog od signala selekcije ulaza U(0) do U(2^b-1). Za indikatore V i D keš memorija ulaza 0 do 2^b-1 se na ulazne linije podataka dovode vrednosti 1 i 0, respektivno, dok se na adresne linije dovode bitovi $A_{l+k-b-1...l}$ generisane adrese. Upis se realizuje samo u po jedan od 2^b indikatora V i D i to one određenu aktivnom vrednošću jednog od signala selekcije ulaza U(0) do U(2^b-1).

Po dovlačenju bloka podataka u keš memoriju, utvrđuje se da sada postoji saglasnost, pa se na već opisani način obavlja čitanje.

Pri generisanju zahteva za upis od strane procesora, na isti način se ispituje saglasnost sa sadržajem keš memorije kao u slučaju operacije čitanja. Ukoliko postoji saglasnost, na isti način kao i za operaciju čitanja se formira adresa i vodi na adresne linije DATA MEMORIJA keš memorija ulaza 0 do 2^b-1, dok se podatak za upis se vodi na ulazne linije podataka DATA MEMORIJA keš memorija ulaza 0 do 2^b-1. Upis se realizuje samo u jednu DATA MEMORIJU i to onu određenu aktivnom vrednošću jednog od signala selekcije ulaza U(0) do U(2^b-1). Za indikator D keš memorija ulaza 0 do 2^b-1 se na ulazne linije podataka dovodi vrednost 1, dok se na adresne linije dovode bitovi $A_{l+k-b-1...l}$ generisane adrese. Upis se realizuje samo u jedan od 2^b indikatora D i to onaj određen aktivnom vrednošću jednog od signala selekcije ulaza U(0) do U(2^b-1).

Ako saglasnost ne postoji, na identičan način kao i za operaciju čitanja, se, najpre, blok iz ulaza keš memorije određenog za zamenu, vraća u operativnu memoriju, ukoliko je modifikovan, a zatim, u isti ulaz keš memorije, dovlači novi blok iz operativne memorije. Potom se ponovo, na već opisani način, vrši provera da li postoji saglasnost, utvrđuje da postoji saglasnost i realizuje upis.

5.2.3. Zamena blokova keš memorije

Pri generisanju zahteva za upis ili čitanje od strane procesora može se utvrditi da se blok u kome je zahtevana reč ne nalazi u bloku keš memorije koji je predviđen odabranom tehnikom preslikavanja. Tada se jedan blok keš memorije mora vratiti u operativnu memoriju, da bi se u keš memoriji napravio prostor za blok iz operativne memorije u kome se zahtevana reč nalazi. Ovaj blok se određuje korišćenjem jednog od algoritama zamene koji se hardverski realizuju u keš memoriji.

Kod keš memorije sa direktnim preslikavanjem algoritam zamene je trivijalan jer je blok za zamenu određen brojem bloka generisane adrese. Kod keš memorije sa asocijativnim i set-asocijativnim preslikavanjem algoritmom zamene se za zamenu bira jedan od svih blokova keš memorije sa asocijativnim preslikavanjem i jedan od svih blokova seta, određenog brojem seta generisane adrese, keš memorije sa set-asocijativnim preslikavanjem. Stoga se algoritam zamene realizuje za celu keš memoriju sa asocijativnim preslikavanjem, a posebno za svaki set keš memorije sa set-asocijativnim preslikavanjem.

Pri izboru algoritma zamene treba voditi računa o dva zahteva. Prvi je da on treba da obezbedi minimalnu verovatnoću da će blok koji je odabran za zamenu i vraćen iz keš u operativnu memoriju ubrzo morati ponovo da se dovuče iz operativne u keš memoriju. Drugi je da cena hardvera potrebnog za njegovu realizaciju bude što je moguće niža. Ova dva zahteva su kontradiktorna, jer je cena hardvera algoritama zamene koji bolje ispunjavaju prvi zahtev viša u odnosu na cenu hardvera algoritama zamene koji to čine lošije. Ovde se razmatraju četiri algoritma zamene i to RANDOM, FIFO, LRU i PSEUDO LRU i daju njihova moguća realizacija. Razmatranja se odnose na keš memoriju sa asocijativnim preslikavanjem i važe i za jedan set keš memorije sa set-asocijativnim preslikavanjem.

5.2.3.1. RANDOM

RANDOM algoritmom zamene za zamenu se slučajno bira blok korišćenjem jednog od postojećih generatora slučajnih brojeva. Mogući način realizacije je da se koristi brojač po modulu 2^n, gde 2^n predstalja broj ulaza keš memorije sa asocijativnim preslikavanjem i broj ulaza po setu keš memorije sa set-asocijativnim preslikavanjem. Može se odabrati neki proizvoljan signal i koristiti za inkrementiranje brojača. U trenutku kada nema saglasnoti i treba odabrati blok za zamenu, trenutna vrednost brojača određuje ulaz za zamenu. Posle toga se produžava se inkrementiranje brojača do sledećeg tenutka kada nema saglasnosti i kada ponovo na osnovu vrednosti brojača treba odabrati ulaz za zamenu.

Ovaj algoritam ne vodi računa o tome da će blok koji je odabran za zamenu i vraćen iz keš u operativnu memoriju možda ubrzo morati ponovo da se dovuče iz operativne u keš memoriju. Međutim, hardver za njegovu realizaciju je jedostavan.

5.2.3.2. FIFO

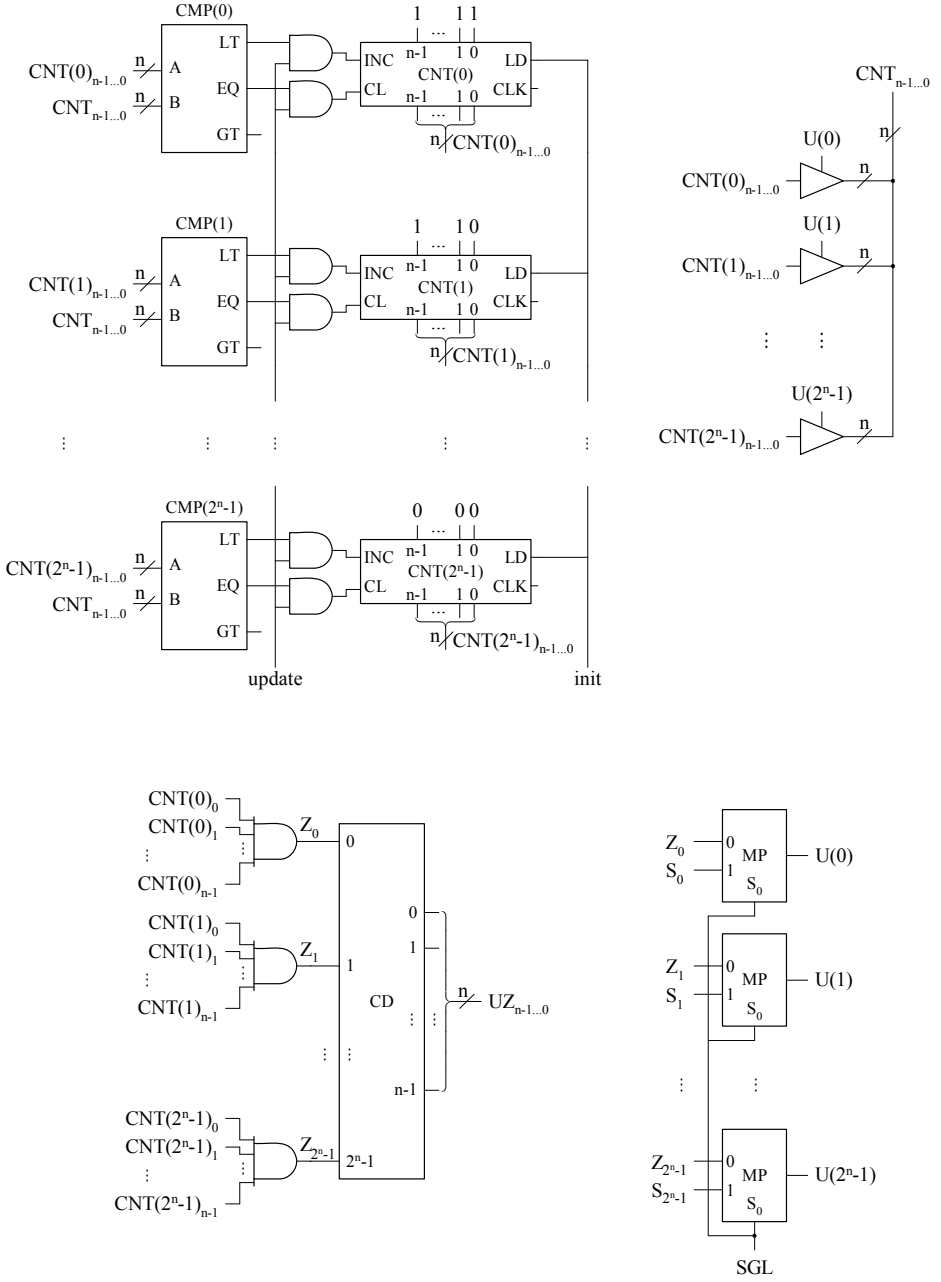
FIFO (*first in—first out*) algoritmom zamene za zamenu se bira blok koji je najranije unet iz operativne memorije u keš memoriju. Mogući način realizacije je da se koristi brojač po modulu 2^n, gde 2^n predstalja broj ulaza keš memorije sa asocijativnim preslikavanjem i broj ulaza po setu keš memorije sa set-asocijativnim preslikavanjem. U trenutku kada nema saglasnoti i treba odabrati blok za zamenu, trenutna vrednost brojača određuje ulaz za zamenu. Pri tome se i vrši inkrementiranje brojača.

Ovaj algoritam ne vodi računa o tome da će blok koji je odabran za zamenu i vraćen iz keš u operativnu memoriju možda ubrzo morati ponovo da se dovuče iz operativne u keš memoriju, već za zamenu bira blokove po onom redosledu po kome su i dovlačeni iz operativne memorije u keš memoriju. Međutim, hardver za njegovu realizaciju je jedostavan.

5.2.3.3. LRU

LRU (*least recently used*) algoritmom zamene za zamenu se bira blok kome se najduže vremena nije pristupalo. Mogući način realizacije je da se koristi 2^n brojača po modulu 2^n, gde 2^n predstalja broj ulaza keš memorije sa asocijativnim preslikavanjem i broj ulaza po setu keš memorije sa set-asocijativnim preslikavanjem. Svakom od 2^n ulaza keš memorije dodeljuje se jedan od 2^n brojača po modulu 2^n. Brojači se tokom rada tako ažuriraju da je u njima uvek 2^n različitih vrednosti i da brojač ulaza kome se najduže vremena nije pristupalo ima sve jedinice i time vrednost 2^n-1. U trenutku kada ima saglasnoti sadržaj brojača ulaza u kome je otkrivena saglasnot se upoređuje sa sadržajima brojača svih preostalih ulaza. Brojači koji imaju manju vrednost od brojača ulaza u kome je otkrivena saglasnost se inkrementiraju, brojači koji imaju veću vrednost od brojača ulaza u kome je otkrivena saglasnost se ne menjaju i brojač ulaza u kome je otkrivena saglasnost se postavlja na nulu. U trenutku kada nema saglasnosti za zamenu se bira ulaz čiji brojač ima sve jedinice i time vrednost 2^n-1, brojači svih preostalih

ulaza imaju manju vrednost od brojača ulaza koji je odabran za zamenu pa se inkrementiraju i brojač ulaza koji je odabran za zamenu se postavlja na nulu. Na početku rada brojači 2^n ulaza treba tako da se inicijalizuju da u njima bude 2^n različitih vrednosti, pri čemu brojače ulaza treba inicijalizovati na vrednosti 2^n-1, 2^n-2, ..., 1 i 0 saglasno redosledu po kome se želi popunjavanje ulaza. Time se i za popunjavanje keš memorije i za zamenu blokova popunjene keš memorije koristi isti mehanizam. Strukturna šema hardvera za realizaciju LRU algoritam zamene po ovom pristupu je data na slici 8.



Slika 8 Realizacija LRU algoritam zamene

Ulazi 0 do 2^n-1 imaju brojače CNT(0) do CNT(2^n-1) i komparatore CMP(0) do CMP(2^n-1). Signalom init se na početku u brojače CNT(0) do CNT(2^n-1) upisuju vrednosti 2^n-1 do 0, čime se određuje da popunjavanje ulaza keš memorije ide redom od ulaza 0 do ulaza 2^n-1. Komparator svakog ulaza upoređuje sadržaj brojača datog ulaza i sadržaj brojača ulaza u

kome je otkrivena saglasnost u situacijama kada se otkrije saglasnost i brojača ulaza koji je odabran za zamenu u situacijama kada nema saglasnoti. Signal LS na izlazu komparatora je aktivan ukoliko je sadržaj sa ulaza A komparatora manji od sadržaja sa ulaza B komparatora, dok je signala EQ aktivan ukoliko su ti sadržaji jednaki i signal GT je aktivan ukoliko je sadržaj sa ulaza A veći od sadržaja sa ulaza B. Pri aktivnoj vrednosti signala update aktivna vrednost signala LS omogućuje inkrementiranje sadržaja odgovarajućeg brojača, dok aktivna vrednost signala EQ postavljanje na vrednost nula.

Sadržaj CNT sa kojim se upoređuju sadržaji brojača odgovarajućih ulaza predstavlja sadržaj brojača ulaza u kome je otkrivena saglasnost u situacijama kada se otkrije saglasnost i brojača ulaza koji je odabran za zamenu u situacijama kada nema saglasnoti. Sadržaj CNT predstavlja sadržaj jednog od brojača CNT(0) do CNT(2^n-1) selektovan preko bafera sa tri stanja aktivnom vrednošću jednog od signala U(0) do U(2^n-1). Signali U(0) do U(2^n-1) predstaljaju signale saglasnosti S_0 do S_{2^n-1} u slučajevima kada je otkrivena saglasnost i signale zamene Z_0 do Z_{2^n-1} u slučajevima kada nema saglasnosti i selektuju se kroz multiplekser aktivnom i neaktivnom vrdnošću signala saglasnosti SGL. Formiranje signala saglasnosti S_0 do S_{2^n-1} i signala zamene Z_0 do Z_{2^n-1} je dato na slikama 3 i 8. Jedan od signala saglasnosti S_0 do S_{2^n-1} je aktivan, čime je i signal SGL aktivan, ukoliko je i jedan od signala M_0 do M_{2^n-1} aktivan i odgovarajući signal V_0 do V_{2^n-1} aktivan (slika 3). Jedan od signala zamene Z_0 do Z_{2^n-1} je aktivan u zavisnosti od toga koji od brojača CNT(0) do CNT(2^n-1) ima sve jedinice (slika 8). Na osnovu signala zamene Z_0 do Z_{2^n-1} na izlazu kodera se formira binarna vrednost ulaza za zamenu $UZ_{n-1...0}$ koja se koristi za formiranje adresa TAG i DATA MEMORIJE, kao i D i V indikatora (slika 3).

Na početku su svi ulazi keš memorije ne važeći pa se svi signali V_0 do V_{2^n-1} postavljaju na neaktivne vrednosti (slika 3). Na početku se signalom init u brojače CNT(0) do CNT(2^n-1) upisuju vrednosti 2^n-1 do 0, čime se određuje da popunjavanje ulaza keš memorije ide redom od ulaza 0 do ulaza 2^n-1 (slika 8). Pri prvom zahtevu za pristup keš memoriji signal saglasnosti SGL je neaktivan jer su zbog neaktivnih vrednosti signala V_0 do V_{2^n-1} i svi signali S_0 do S_{2^n-1} neaktivni. Kako je sadržaj brojača CNT(0) sve jedinice, signal Z_0 je aktivan. Na linijama $UZ_{n-1...0}$ se pojavljuje vrednost nula, pa se zahtevani blok dovlači u ulaz nula keš memorije. Pored toga dobija se aktivna vrednost signala U(0), pa se na komparatorima kao sadržaj CNT pojavljuje sadržaj brojača CNT(0). Komparator CMP(0) daje aktivnu vrednost signala EQ, dok komparatori svih ostalih ulaza daju aktivnu vrednost signala LS. Zbog toga se generisanjem aktivne vrednosti signala update ažuriranje brojača CNT(0) do CNT(2^n-1) vrši tako da se brojač CNT(0) postavi na nulu, dok se sadržaji svih ostalih brojača inkrementiraju. Kako je pri inicijalizaciji brojač ulaza jedan CNT(1) postavljen na 2^n-2, to njegov sadržaj posle inkrementiranja postaje 2^n-1.

Kada se sledeći put utvrdi da u keš memoriji nema saglasnosti sadržaj brojača CNT(1) je sve jedinice, pa je signal Z_1 je aktivan. Na linijama $UZ_{n-1...0}$ se pojavljuje vrednost jedan, pa se zahtevani blok dovlači u ulaz jedan keš memorije. Pored toga dobija se aktivna vrednost signala U(1), pa se na komparatorima kao sadržaj CNT pojavljuje sadržaj brojača CNT(1). Komparator CMP(1) daje aktivnu vrednost signala EQ, dok komparatori svih ostalih ulaza daju aktivnu vrednost signala LS. Zbog toga se generisanjem aktivne vrednosti signala update ažuriranje brojača CNT(0) do CNT(2^n-1) vrši tako da se brojač CNT(1) postavi na nulu, dok se sadržaji svih ostalih brojača inkrementiraju. Na isti način se vrši popunjavanje i ostalih ulaza keš memorije i ažuriranje brojača CNT(0) do CNT(2^n-1).

Kada se u delimično ili potpuno popunjenoj keš memoriji otkrije saglasnost, jedan od signala S_0 do S_{2^n-1} je aktivan (slike 3 i 8). Na linijama $US_{n-1...0}$ se pojavljuje binarna vrednost ulaza u kome je otkrivena saglasnost. Ova vrednost se koristi za adresiranja DATA MEMEORIJE radi čitanja ili upisa podatka i adresiranje D indikatora u koji se upisuje

jedinica u slučaju upisa podatka. Pored toga na osnovu signala S_0 do S_{2^n-1} dobija se aktivna vrednost jednog od signala U(0) do U(2^n-1) koji odgovara ulazu u kome je otkrivena saglasnost. Stoga se na komparatorima kao sadržaj CNT pojavljuje sadržaj jednog od brojača CNT(0) do CNT(2^n-1) koji odgovara ulazu u kome je otkrivena saglasnost. Komparator ulaza u kome je otkrivena saglasnot daje aktivnu vrednost signala EQ, komparatori ulaza čiji brojači imaju manju vrednost od vrednosti brojača ulaza u kome je otkrivena saglasnos daju aktivnu vrednost signala LS, dok komparatori ulaza čiji brojači imaju veću vrednost od vrednosti brojača ulaza u kome je otkrivena saglasnos daju aktivnu vrednost signala GT. Zbog toga se generisanjem aktivne vrednosti signala update ažuriranje brojača CNT(0) do CNT(2^n-1) vrši tako da se brojač ulaza u kome je otkrivena saglasnost postavi na nulu, brojači koji imaju manju vrednost od vrednosti brojača ulaza u kome je otkrivena saglasnost inkrementiraju, dok se sadržaji brojači koji imaju veću vrednost od vrednosti brojača ulaza u kome je otkrivena saglasnost ne menjaju.

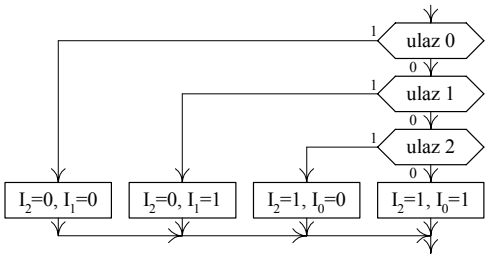
Ovaj algoritam je baziran na pretpostavci da onim blokovima kojima je više pristupano u prošlosti verovatno će biti pristupano i u budućnosti, pa se za zamenu bira blok kome se najmanje skoro pristupalo. Međutim, hardver za njegovu realizaciju je složen.

5.2.3.4. PSEUDO LRU

PSEUDO LRU (*least recently used*) algoritmom zamene se pokušava realizacija principa LRU algoritma zamene sa jednostavnijim hardverom. Mogući način realizacije je dat za keš memorije sa asocijativnim preslikavanjem i četiri ulaza i keš memorije sa set-asocijativnim preslikavanjem i četiri ulaza po setu. Ovaj algoritam zamene često se koristi kod set-asocijativnog preslikavanja kod koga je broj ulaza po setu najčešće dva, četiri ili osam.

Ulazi 0 do 3 su upareni u dve grupe, pri čemu ulazi 0 i 1 jedan čine jednu grupu i ulazi 2 i 3 drugu grupu. Ovim grupama se dodeljuje indikator I_2 koji se postavlja na 0 kad god se pristupi ulazu 0 ili 1 i na 1 kod god se pristupi ulazu 2 ili 3. Ulazima iz grupe koju čine ulazi 0 i 1 dodeljuje se indikator I_1 koji se postavlja na 0 kad god se pristupi ulazu 0 i na 1 kod god se pristupi ulazu 1. Ulazima iz grupe koju čine ulazi 2 i 3 dodeljuje se indikator I_0 koji se postavlja na 0 kad god se pristupi ulazu 2 i na 1 kod god se pristupi ulazu 3. Dijagram toka PSEUDO LRU algoritma zamene pri ažuriranju je dat na slici 9.

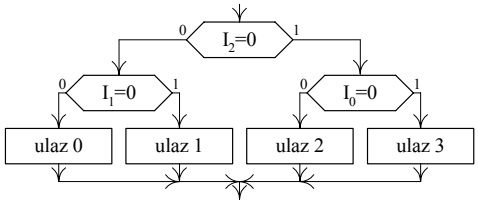
.



Slika 9 Dijagram toka PSEUDO LRU algoritma zamene pri ažuriranju

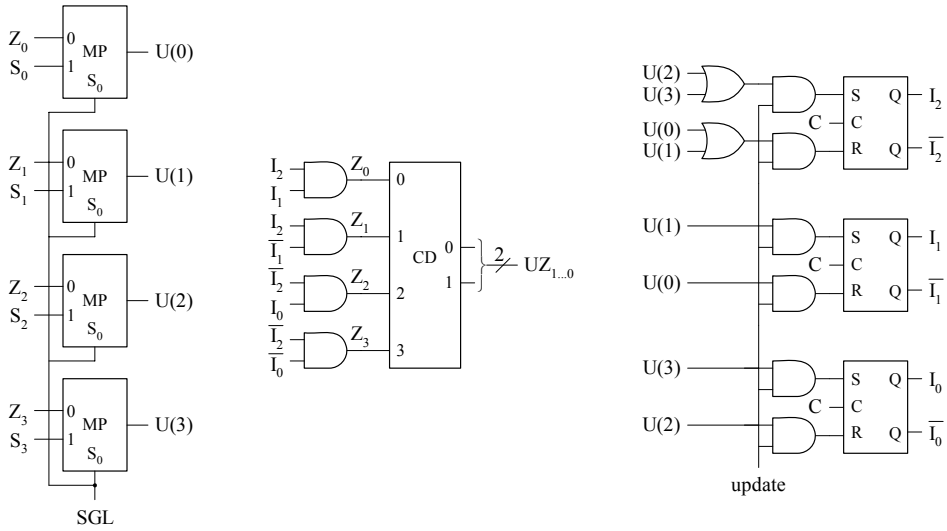
Prilikom odlučivanja koji od četiri ulaza odabrati za zamenu posmatra se najpre indikatora I_2. Ukoliko je I_2 nula, zadnji put se pristupili ulazu 0 ili 1, pa za zamenu treba odabrati ulaz 2 ili 3. Zato se sada proverava indikator I_0. Ukoliko je I_0 nula, zadnji put se, posmatrajući samo ulaze 2 i 3, pristupili ulazu 2, pa za zamenu treba odabrati ulaz 3. Ukoliko je I_0 jedan, zadnji put se, posmatrajući samo ulaze 2 i 3, pristupili ulazu 3, pa za zamenu treba odabrati ulaz 2. S druge strane, ukoliko je I_2 jedan, zadnji put se pristupili ulazu 2 ili 3, pa za zamenu treba odabrati ulaz 0 ili 1. Zato se sada proverava indikator I_1. Ukoliko je I_1 nula, zadnji put se,

posmatrajući samo ulaze 0 i 1, pristupili ulazu 0, pa za zamenu treba odabrati ulaz 1. Ukoliko je I_0 jedan, zadnji put se, posmatrajući samo ulaze 0 i 1, pristupili ulazu 1, pa za zamenu treba odabrati ulaz 0. Dijagram toka PSEUDO LRU algoritma zamene pri određivanju ulaza za zamenu je dat na slici 10.



Slika 10 Dijagram toka pseudo LRU algoritma zamene pri određivanju ulaza za zamenu

Strukturna šema hardvera za realizaciju PSEUDO LRU algoritam zamene po ovom pristupu je data na slici 11.



Slika 11 Realizacija pseudo LRU algoritma zamene

Indikatori I_2, I_1 i I_0 su realizovani pomoću flip-flopova. Indikatori se postavljaju na osnovu vrednosti signala U(0) do U(3) pri aktivnoj vrednosti signala update. Signali U(0) do U(3) predstaljaju signale saglasnosti S_0 do S_3 u slučajevima kada je otkrivena saglasnost i signale zamene Z_0 do Z_3 u slučajevima kada nema saglasnosti i selektuju se kroz multiplekser aktivnom i neaktivnom vrdnošću signala saglasnosti SGL. Formiranje signala saglasnosti S_0 do S_3 i signala zamene Z_0 do Z_3 je dato na slikama 3 i 8. Jedan od signala saglasnosti S_0 do S_3 je aktivan, čime je i signal SGL aktivan, ukoliko je i jedan od signala M_0 do M_3 aktivan i odgovarajući signal V_0 do V_3 aktivan (slika 3). Jedan od signala zamene Z_0 do Z_3 je aktivan u zavisnosti od trenutne vrednosti indikatora I_2, I_1 i I_0 (slika 11). Na osnovu signala zamene Z_0 do $Z_2{}^n{}_{-1}$ na izlazu kodera se formira binarna vrednost ulaza za zamenu $UZ_{1...0}$ koja se koristi za formiranje adresa TAG i DATA MEMORIJE, kao i D i V indikatora (slika 3).

Na početku su svi ulazi keš memorije ne važeći pa se svi signali V_0 do V_3 postavljaju na neaktivne vrednosti (slika 3). Najjednostavnije je da se indikatori I_2, I_1 i I_0 proizvoljno postave na vrednosti 0 i 1 i da se na osnovu postavljenih vrednosti i saglasno algoritmu zamene vrši popunjavanje ulaza keš memorije (slika 11). Pri prvom zahtevu za pristup keš memoriji signal saglasnosti SGL je neaktivan jer su zbog neaktivnih vrednosti signala V_0 do V_3 i svi signali S_0 do S_3 neaktivni. Jedan od signala zamene Z_0 do Z_3 je aktivan u zavisnosti od trenutne vrednosti indikatora I_2, I_1 i I_0 (slika 11). Na osnovu signala zamene Z_0 do Z_3 na izlazu kodera

se formira binarna vrednost ulaza za zamenu $UZ_{1...0}$ koja se koristi za formiranje adresa TAG i DATA MEMORIJE, kao i D i V indikatora (slika 3) pri dovlačenju zahtevanog bloka u keš memoriju. Pored toga dobija se aktivna vrednost jednog od signala U(0) do U(3), pa se indikatori I_2, I_1 i I_0 postavljaju na odgovarajuće vrednosti.

Kada se sledeći put utvrdi da u keš memoriji nema saglasnosti signali S_0 do S_3 su, kao i u kao i u prethodnom slučaju, neaktivni, pa je i signal SGL neaktivan. Jedan od signala zamene Z_0 do Z_3 je, u zavisnosti od trenutne vrednosti indikatora I_2, I_1 i I_0, aktivan (slika 11). Na osnovu signala zamene Z_0 do Z_3 na izlazu kodera se formira binarna vrednost ulaza za zamenu $UZ_{1...0}$ koja se koristi za pri dovlačenje zahtevanog bloka u keš memoriju. Pored toga dobija se aktivna vrednost jednog od signala U(0) do U(3), pa se indikatori I_2, I_1 i I_0 postavljaju na odgovarajuće vrednosti. Na isti način se vrši popunjavanje i ostalih ulaza keš memorije i ažuriranje indikatora I_2, I_1 i I_0.

Kada se u delimično ili potpuno popunjenoj keš memoriji otkrije saglasnost, jedan od signala S_0 do S_3 je aktivan (slike 3 i 8). Na linijama $US_{1...0}$ se pojavljuje binarna vrednost ulaza u kome je otkrivena saglasnost. Ova vrednost se koristi za adresiranja DATA MEMEORIJE radi čitanja ili upisa podatka i adresiranje D indikatora u koji se upisuje jedinica u slučaju upisa podatka. Pored toga na osnovu signala S_0 do S_3 dobija se aktivna vrednost jednog od signala U(0) do U(3) koji odgovara ulazu u kome je otkrivena saglasnost, pa se indikatori I_2, I_1 i I_0 postavljaju na odgovarajuće vrednosti.

Ovaj algoritam je baziran na istoj pretpostavci kao i LRU algoritam i ima za cilj da za zamenu bira blok kome se najmanje skoro pristupalo. Hardver za njegovu realizaciju je jednostavniji u odnosu na hardver LRU algoritma, pa je ovaj algoritam manje precizan u odnosu na LRU algoritam. Ova nepreciznost se javlja, na primer, kada je u paru sa blokom iz ulaza 0, u kome se dosta često otkriva saglasnost, blok iz ulaza 1 kome se dosta dugo nije pristupalo. S druge strane u ulazima 2 i 3 mogu da budu blokovi u kojima se, takođe, dosta često otkriva saglasnost. Uz pretpostavku da se u određenom vremenskom periodu više puta pristupa i otkriva saglasnost za blokove iz ulaza 0, 2 i 3, dok se u datom vremenskom periodu bloku iz ulaza 1 ne pristupa, mogu da nastanu dve situacije, pri čemu je algoritam u prvoj situaciji neprecizan a u drugoj precizan. Pri tome se, kad god se otkrije saglasnost u bloku iz ulaza 0, indikator I_2 se postavlja na 0, čime se određuje da, ukoliko pri sledećem pristupu keš memoriji nema saglasnosti, blok za zamenu treba da bude iz ulaza 2 ili 3. S druge strane, kad god se otkrije saglasnost u bloku iz ulaza 2 ili 3, indikator I_2 se postavlja na 1, čime se određuje da, ukoliko pri sledećem pristupu keš memoriji nema saglasnosti, blok za zamenu treba da bude iz ulaza 0 ili 1.

Ukoliko se sada za neki pristup utvrdi da nema saglasnosti ni u jednom od četiri ulaza, a zadnji pristup je bio za blok iz ulaza 0 za koji je utvrđena saglasnost, indikator I_2 ima vrednost 0. To znači da se za zamenu na osnovu vrednosti indikatora I_0 bira blok iz ulaza 2 ili 3 u kojima je više puta otkrivana saglasnost u datom vremenskom periodu, a ne blok iz ulaza 1 kome uopšte nije pristupano u datom vremenskom periodu. Algoritam je u ovoj situaciji bio neprecizan, jer je trebalo zameniti blok iz ulaza 1, kome se uopšte nije pristupalo u datom vremenskom periodu.

Ukoliko se, međutim, za neki pristup utvrdi da nema saglasnosti ni u jednom od četiri ulaza, a zadnji pristup je bio za blok iz ulaza 2 ili 3 za koji je utvrđena saglasnost, indikator I_2 ima vrednost 1. To znači da se za zamenu na osnovu vrednosti indikatora I_1 bira blok iz ulaza 0 ili 1. Kako je u datom vremenskom periodu u bloku iz ulaza 0 više puta otkrivana saglasnost, dok bloku iz ulaza 1 uopšte nije pristupano, indikator I_1 ima vrednost 0, pa se za zamenu bira blok iz ulaza 1. Algoritam je u ovoj situaciji bio precizan, jer je za zamenu odabran blok iz ulaza 1, kome se uopšte nije pristupalo u datom vremenskom periodu.

5.2.4. Ažuriranje operativne memorije

Ažuriranje operativne memorije određuje kako se kod operacije upisa menja sadržaj u operativnoj memoriji. Pri tome se, kod zahteva za upis, mogu javiti dve situacije. Prva je da u keš memoriji postoji saglasnost, a druga da nema saglasnosti.

Za slučaj kada je u keš memoriji otkrivena saglasnost, postoje dva pristupa i to *upiši skroz* (*write through* ili *store through*) i *vrati nazad* (*write back* ili *copy back*). Kod pristupa *upiši skroz*, pri svakom zahtevu za upis istovremeno se vrši upis i u keš memoriju i u operativnu memoriju. Kod pristupa *vrati nazad*, pri svakom zahtevu za upis vrši se upis samo u keš memoriju, pa odgovarajući sadržaj u operativnoj memoriji nije ažuran. Zbog toga se za svaki blok u keš memoriji vodi evidencija o tome da li je modifikovan ili ne. Ukoliko je kasnije potrebno dovući novi blok iz operativne memorije na mesto nekog bloka u keš memoriji koji je nekim od prethodni upisa modifikovan, potrebno je, najpre, dati blok keš memorije vratiti u operativnu memoriju i time obezbediti da i sadržaj u operativnoj memoriji bude ažuran. Pored toga, kada se nekom procesu oduzima procesor, treba proveriti koji su blokovi u keš memoriji modifikovani, pa ih, radi ažuriranja sadržaja u operativnoj memoriji, vratiti iz keš memorije u operativnu memoriju. Stoga kod keš memorija koje koriste ovaj pristup ažuriranja sadržaja operativne memorije, pored zahteva za čitanje i upis, postoje i zahtevi za selektivno i kompletno vraćanje blokova iz keš memorije u operativnu memoriju (*flush*).

Prednost pristupa *upiši skroz* je u tome da je operativna memorija uvek ažurna čime je obezbeđena konzistentnost sadržaja operativne i keš memorije. Nedostatak ovog pristupa je u obraćanju operativnoj memoriji pri svakom upisu u keš memoriju, čime se bespotrebno opterećuje magistrala upisuivanjem međurezultata u operativnu memoriju.

Prednost pristupa *vrati nazad* je u tome što se operativnoj memoriji i magistrali pristupa samo onda kada se blok vraća iz keš memorije u operativnu memoriju što rezultuje u manjem saobraćaju na magistrali. Nedostatak ovog pristupa je potreba da se blok koji se izbacuje iz keš memorije mora najpre vratiti u operativnu memoriju, pa tek onda dovući novi, što znatno usporava odziv keš memorije u slučaju promašaja.

Ovde se vidi da su sve prednosti jednog pristupa ujedno i nedostaci drugog. Stoga se pristup *vrati nazad* koristi tamo gde je magistrala usko grlo sistema, a pristup *upiši skroz* gde magistrala to nije.

Za slučaj kada je u keš memoriji nije otkrivena saglasnost, postoje dva pristupa i to *dovuci blok* (*write allocate*) i *ne dovlači blok* (*no write allocate*). Kod pristupa *dovuci blok*, blok se dovlači iz operativne u keš memoriju, čime se obezbeđuje da se sada u keš memoriji otkriva saglasnost. Dalji postupak odgovara prethodno opisanoj situaciji za operaciju upisa i otkrivenu saglasnost, u kojoj se upis vrši u keš memoriju, a za ažuriranje sadržaja operativne memorije koristi pristup *upiši skroz* ili *vrati nazad*. Kod pristupa *ne dovlači blok*, blok se ne dovlači iz operativne u keš memoriju, već se upis vrši samo u operativnu memoriju.

Obično se uz pristup *vrati nazad* (*write back* ili *copy back*) koristi pristup *dovuci blok* (*write allocate*), dok se uz pristup *upiši skroz* (*write through* ili *store through*) koristi pristup *ne dovlači blok* (*no write allocate*).

5.2.5. Neka razmatranja u vezi realizacije keš memorije

U osnovni mehanizam funkcionisanja keš memorije moguće je uvesti neka poboljšanja koja skraćuju vreme čitanja iz i upisa u keš memoriju.

Moguće poboljšanje je u tome da keš memorija, ako se radi o operaciji upisa, odmah dozvoli procesoru da produži sa izvršavanjem tekuće instrukcije bez obzira na to da li je upis zaista izvršen ili nije. Time će paralelno keš memorija obavljati upis a procesor izvršavati instrukciju. Keš memorija neće moći da prihvati novi zahtev za upis ili čitanje ukoliko se prethodno započeti upis nije završio.

Poboljšanje je moguće učiniti i u slučaju operacije čitanja kada traženi blok nije u keš memoriji, već ga treba dovući iz operativne memorije. Tada procesor ne mora da čeka da ceo blok bude prenesen iz operativne u keš memoriju i da tek tada dobije traženi sadržaj. Keš memorija može procesoru dostaviti traženi sadržaj čim on stigne iz operativne u keš memoriju. U tom slučaju procesor može ranije da nastavi izvršavanje tekuće instrukcije, a da se paralelno s time ostatak bloka prenese iz operativne u keš memoriju. Pri tome dovlačenje reči bloka treba započeti od reči čije je čitanje zahtevano. Kao u prethodnom slučaju, keš memorija opet ne može prihvatiti novi zahtev za čitanje ili upis sve dok se prenos prethodnog bloka ne obavi do kraja. Ova tehnika naziva se *by-pass*.

Sledeće poboljšanje je moguće ostvariti u slučajevima kada je potrebno izvršiti vraćanje modifikovanog bloka u operativnu memoriju. Da bi se ubrzao taj postupak moguće je u poseban bafer privremeno smestiti ceo blok koji se vraća i odmah preći na dovlačenje novog bloka iz operativne memorije. Tek po završetku dovlačenja novog bloka prelazi se na vraćanje u operativnu memoriju bloka koji se nalazi u baferu. I ovde keš memorija ne može prihvatiti novi zahtev za čitanje ili upis sve dok se cela operacija ne završi do kraja. Ovo poboljšanje se naziva *baferisanje*.

Sva navedena poboljšanja imaju za cilj da se procesor što manje zadržava prilikom obraćanja keš memoriji. Pri tome se pretpostavlja da se procesor vrlo verovatno neće uskoro ponovo obraćati keš memoriji, pa će do sledećeg obraćanja procesora keš memoriji, keš memorija moći da obavi prethodno započetu operaciju do kraja.

U slučaju operacije upisa postoji više načina da se promene sadržaja operativne i keš memorije realizuju. Ako se koristi pristup *vrati nazad* onda se promena u operativnoj memoriji ostvaruje samo u slučaju vraćanja bloka u operativnu memoriju. Što se tiče keš memorije kod ovog pristupa se promena u keš memoriji ostvaruje uvek i to i u slučaju da ima saglasnosti i u slučaju da nema saglasnosti, pri čemu se u drugom slučaju to čini tek pošto se blok prenese iz operativne u keš memoriju. Ako se koristi pristup *upiši skroz* onda se promena u operativnoj memoriji ostvaruje uvek. Što se tiče keš memorije kod ovog pristupa se u slučaju saglasnosti ili upisuje novi sadržaj u keš memoriju ili se ulaz keš memorije proglašava nevažećim. U slučaju da nema saglasnosti u nekim situacijama ažurirani blok operativne memorije se dovlači u keš memoriju, dok se u drugim ažurirani blok operativne memorije ne dovlači u keš memoriju.

U osnovni mehanizam funkcionisanja keš memorije je moguće uvesti neka poboljšanja, koja skraćuju vreme čitanja iz i upisa u keš memoriju. Moguće poboljšanje je u tome da keš memorija, ako se radi o operaciji upisa, upis izvrši u bafer podatka i odmah dozvoli procesoru da produži sa izvršavanjem tekuće instrukcije. Time će se paralelno obavljati upis iz bafera podatka u operativnu memoriju i izvršavati instrukcije procesora. Ove tehnike se nazivaju

baferovanje podatka i *rani start procesora*. Poboljšanje je moguće učiniti i u slučaju operacije čitanja kada traženi blok nije u keš memoriji, već ga treba dovući iz operativne memorije. Tada procesor ne mora da čeka da ceo blok bude dovučen iz operativne memorije u keš memoriju i da tek tada dobije traženi sadržaj. Keš memorija može procesoru dostaviti traženi sadržaj čim on stigne iz operativne u keš memoriju. U tom slučaju, procesor može ranije da nastavi izvršavanje tekuće instrukcije i da se paralelno ostatak bloka prenosi iz operativne u keš memoriju. Pri tome, dovlačenje reči bloka treba započeti od reči čije je čitanje zahtevano. Ove tehnike se nazivaju *prosleđivanje* i *rani start procesora*. Sledeće poboljšanje je moguće ostvariti u slučajevima kada je potrebno izvršiti vraćanje modifikovanog bloka odabranog za zamenu iz keš memorije u operativnu memoriju. Da bi se ubrzao taj postupak, moguće je postaviti bafer bloka, koji će prihvatiti ceo blok koji se vraća, i odmah preći na dovlačenje bloka iz operativne memorije. Tek po završetku dovlačenja bloka iz operativne u keš memoriju, prelazi se na vraćanje bloka iz bafera bloka u operativnu memoriju. Ovo poboljšanje se naziva *baferovanje bloka podataka*.

5.3. VIRTUELNA MEMORIJA I JEDINICA ZA UBRZAVANJE

Kod velikog broja savremenih računara ne postoji jedan prema jedan korespondencija između adresa koje se generišu u programu i adresa operativne memorije. Adrese koje se generišu u programu zovu se virtuelne adrese, a adrese operativne memorije realne adrese. Opseg adresa koje se generišu u programu se zove virtuelni adresni prostor, a opseg adresa operativne memorije realni adresni prostor.

U ovim razmatranjima se uzima da se kompletan virtuelni adresni prostor dodeljuje svakom procesu. Kompletni programi i podaci svakog procesa nalaze se na disku, a samo njihovi delovi za kojima u određenom trenutku postoji potreba dovlače se sa diska i smeštaju u neki deo operativne memorije. Zbog toga kod ovih računara postoji potreba da se za svaki proces vodi evidencija o tome koji se njegovi delovi nalaze u operativnoj memoriji i u kom njenom delu. To se obično realizuje pomoću posebnih tabela koje se formiraju u operativnoj memoriji za svaki proces i koje se nazivaju tabele preslikavanja. U zavisnosti od toga kako se virtuelni adresni prostor radi dovlačenja sa diska u operativnu memoriju deli na delove, razlikuju se tri tipa virtuelnih memorija, i to virtuelne memorije sa straničnom, segmentnom i segmentno-straničnom organizacijom.

Kod virtuelnih memorija stranične organizacije virtuelni adresni prostor se deli na delove fiksne veličine koji se nazivaju stranice, a realni adresni prostor se deli na delove fiksne veličine koji se nazivaju blokovi. Veličina stranice odgovara veličini bloka. Pojedine stranice svih procesa se po potrebi smeštaju u raspoložive blokove operativne memorije. Kada se svi blokovi operativne memorije popune stranicama različitih procesa, neka od tih stranica se vraća na disk da bi se oslobodio blok u operativnoj memoriji za neku novu stranicu.

Kod virtuelnih memorija segmentne organizacije virtuelni adresni prostor se deli na delove promenljive veličine koji se nazivaju segmenti. Pojedini segmenti svih procesa se po potrebi smeštaju u delove operativne memorije koji po veličini odgovaraju veličinama segmenata koji se u njih smeštaju. Kada je kompletna operativna memorija popunjena segmentima različitih procesa, jedan ili više segmenata se vraća na disk da bi se u operativnoj memoriji oslobodio prostor dovoljan za smeštanje segmenta koji se dovlači.

Kod segmentno-stranične organizacije virtuelne memorije virtuelni adresni prostor se deli na delove promenljive veličine koji se nazivaju segmenti, a onda se segmenti dele na delove fiksne veličine koji se nazivaju stranice. Realni adresni prostor se deli na delove fiksne

veličine koji se nazivaju blokovi. Veličina stranice odgovara veličini bloka. Pojedine stranice pojedinih segmenata svih procesa se po potrebi smeštaju u raspoložive blokove operativne memorije. Kada se svi blokovi operativne memorije popune stranicama segmenata procesa, neka od stranica se vraća na disk da bi se oslobodio blok za neku novu stranicu negog segmenta nekog procesa.

Razdvajanjem virtuelnog i realnog adresnog prostora javlja se potreba za preslikavanjem virtuelnih adresa u realne korišćenjem tabela preslikavanja. S obzirom da se sve tabele preslikavanja nalaze u operativnoj memoriji, preslikavanje virtuelnih adresa u realne bi drastično usporilo vreme izvršavanja instrukcija. To je razlog što kod računara sa virtuelnom memorijom postoje posebne jedinice, koje će se u daljem tekstu nazivati jedinice za preslikavanje, čiji je zadatak da ubrzaju postupak preslikavanja virtuelnih adresa u realne.

U daljem tekstu se daju osnovne karakteristike virtuelnih memorija i jedinica za preslikavanje.

5.3.1. Organizacija virtuelne memorije

Postoje tri osnovne vrste organizacije virtuelnih memorija, i to:

stranična,

segmentna i

segmentno-stranična.

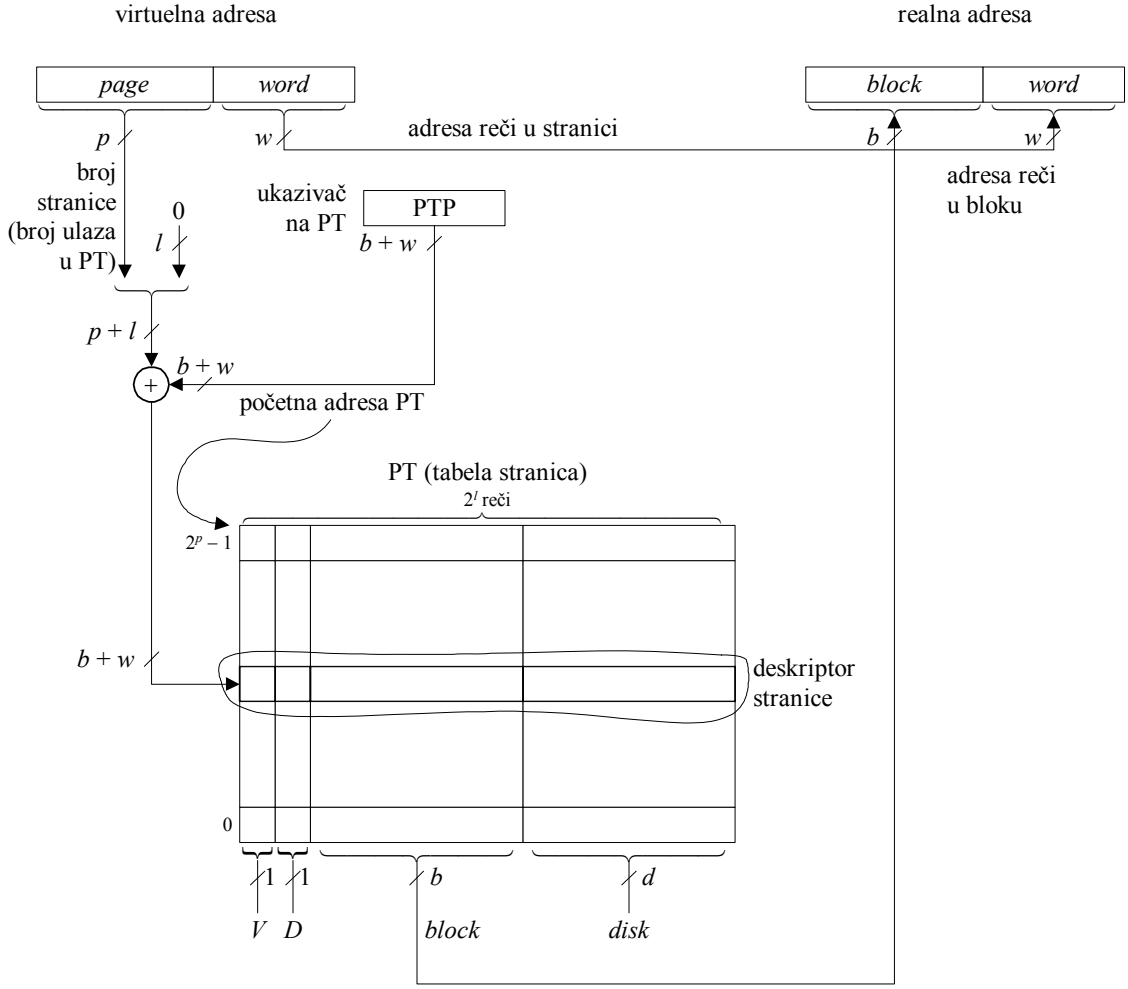
U daljem tekstu se daju samo one osobine svake od navedenih organizacija virtuelne memorije koje su relevantne za realizaciju jedinica za preslikavanje.

5.3.1.1. Stranična organizacija

Kod stranične organizacije virtuelne memorije virtuelna adresa ima dva polja: broj stranice (*page*) dužine p bita i adresa reči u stranici (*word*) dužine w bita. Kako je veličina virtuelnog adresnog prostora 2^{p+w} reči, a stranice 2^w reči, to su veličine polja *page* i *word* p i w bita, respektivno. Kod stranične organizacije virtuelne memorije realna adresa ima dva polja: broj bloka (*block*) dužine b bita i adresa reči u bloku (*word*) dužine w bita. Kako je veličina realnog adresnog prostora 2^{b+w} reči, a bloka 2^w reči, to su veličine polja *block* i *word* b i w bita, respektivno. Struktura virtuelne i realne adrese i dužine u bitovima delova virtuelne i realne adrese su prikazani na slici 10.

Tabela stranica (PT) jednog procesa je data na slici 10. Tabela stranica ima poseban ulaz za svaku stranicu procesa. U njima se nalaze informacije neophodne za preslikavanje stranica virtuelnog adresnog prostora procesa u blokove fizičke memorije. Te informacije se nazivaju deskriptori stranica. S obzirom da u virtuelnom adresnom prostoru procesa ima 2^p stranica, to i tabela stranica ima 2^p ulaza. Početna adresa tabele stranica sadržana je u posebnom registru procesora koji se naziva ukazivač na PT (PTP). Na osnovu sadržaja ukazivača na PT i broja stranice, jedinica za preslikavanje virtuelnih adresa u realne adrese dolazi do deskriptora stranice, a time i do informacija neophodnih za preslikavanje.

U operativnoj memoriji postoji posebna tabela stranica svakog procesa. Početne adrese tabela stranica svih procesa čuva operativni sistem. Prilikom prebacivanja procesora sa procesa na procesa operativni sistem, najpre, čuva kontekst procesa kome se oduzima procesor, a potom, restaurira kontekst procesa kome se dodeljuje procesor. Tom prilikom se u ukazivač na PT upisuje početna adresa tabele stranica procesa kome se dodeljuje procesor. Time se obezbeđuje da jedinica za preslikavanje pristupa tabeli stranica procesa kome je dodeljen procesor.



Slika 10 Virtuelna adresa, realna adresa i tabela stranica za straničnu organizaciju virtuelne memorije

Polja deskriptora stranice su:

V (1 bit)—stranica u memoriji,

D (1 bit)—stranica modifikovana,

block (*b* bita)—broj bloka i

disk (*d* bita)—adresa na disku.

Polje *V* označava da li je data stranica u operativnoj memoriji. Postavlja ga operativni sistem prilikom dovlačenja date stranice sa diska u operativnu memoriju. Koristi ga jedinica za preslikavanje i to ako

je postavljen, da formira realnu adresu i

nije postavljen, da generiše prekid.

Polje *D* označava da li je data stranica modifikovana. Postavlja ga jedinica za preslikavanje ako je bilo operacija upisa u datu stranicu. Koristi ga operativni sistem onda kada odabere datu stranicu za zamenu i to ako

je postavljen, da datu stranicu vrati na disk

nije postavljen, da datu stranicu ne vraća na disk.

Polje *block* označava broj bloka operativne memorije u kome se nalazi data stranica. Vrednost u ovom polju ima smisla jedino ako je polje *V* postavljeno. Polje *block* postavlja operativni sistem prilikom dovlačenja date stranice sa diska u operativnu memoriju. Koristi ga jedinica za preslikavanje, i to ako je polje V postavljeno, da formira realnu adresu.

Polje *disk* označava adresu date stranice na disku. Polje *disk* postavlja operativni sistem prilikom formiranja tabele stranice datog procesa. Koristi ga samo operativni sistem i to radi lociranja date stranice na disku prilikom

dovlačenja stranice sa diska u odabrani blok operativne memorije

vraćanja modifikovane stranice iz bloka operativne memorije odabranog za zamenu na disk.

Uzeto je da je veličina adrese stranice na disku *d* bitova.

Preslikavanje virtuelne u realnu adresu za slučaj da je stranica u memoriji realizuje se kompletno hardverski i to pomoću jedinice za preslikavanje. Ovo preslikavanje se realizuje u sledećim koracima:

Polje *page* virtuelne adrese predstavlja broj ulaza u tabelu stranica u kome se nalazi deskriptor date stranice. S obzirom na to da deskriptor stranice zauzima 2^l reči potrebno je broj ulaza u tabelu stranica pretvoriti u pomeraj u odnosu na početak tabele stranica. To se realizuje pomeranjem ulevo za *l* mesta sadržaja polja *page* vitruelne adrese.

Sadržaj registra ukazivač na PT predstavlja početnu adresu tabele stranica. Pomeraj u odnosu na početak tabele stranica formiran u prethodnom koraku se sabira sa sadržajem registra ukazivač na PT i time dobija adresa deskriptora date stranice. Počev od formirane adrese treba očitati odgovarajući broj reči da bi se došlo do polja *V* i *block* koja koristi jedinica za preslikavanje.

Polje *V* deskriptora ukazuje na to da li je data stranica u memoriji. Stoga se, najpre, sa adrese formirane u prethodnom koraku čita prva reč u kojoj se nalazi polje *V* deskriptora i vrši provera da li je ovo polje postavljeno. Za slučaj kada je stranica u memoriji utvrdiće se da je polje *V* postavljeno, pa se produžava sa sledećim koracima.

Polje *block* deskriptora sadrži broj bloka u kome se nalazi data stranica. Stoga se sada čita potreban broj reči deskriptora da bi se do njega došlo. Konkatenacijom polja *block* iz deskriptora stranice i polja *word* iz virtuelne adrese formira se realna adresa.

U slučaju da stranica nije u memoriji realizuju se samo prva tri od četiri koraka za slučaj kada je stranica u memoriji. U koraku tri se u ovom slučaju utvrđuje da polje *V* nije postavljeno, pa se generiše prekid. Svi koraci do generisanja signala prekida, uključujući i njegovo generisanje, realizuju se hardverski pomoću jedinice za preslikavanje. Sve ostalo, što za ovaj slučaj potom treba uraditi, radi se softverski i to radi poseban deo operativnog sistema. Ovo se realizuje u sledećim koracima:

Oduzima se procesor datom procesu, njegov kontekst čuva i proces stavlja u red blokiranih procesa.

Organizuje se dovlačenje date stranice sa diska u neki od blokova operativne memorije.

Dodeljuje se procesor nekom od radnosposobnih procesa. Tada se restaurira njegov kontekst. U okviru toga se u programski brojač upisuje nova vrednost i time kontrola predaje tom novom procesu.

Dovlačenje stranice sa diska u neki od blokova operativne memorije se realizuje u sledećim koracima:

Traži se blok u koji će se smestiti data stranica. Pri tome se izvršava jedan od sledeća dva koraka:

Ukoliko ima slobodnih blokova po nekom algoritmu se odlučuje koji blok treba dodeliti datoj stranici.

Ukoliko nema slobodnih blokova po nekom algoritmu se odlučuje koju stranicu treba izbaciti iz operativne memorije da bi se blok u kome se ona nalazi oslobodio i dodelio datoj stranici. Ukoliko je stranica odabrana za izbacivanje modifikovana mora se vratiti na disk pre nego što se blok u kome se ona nalazi koristi da se u njega dovuče nova stranica. Adresa na disku stranice odabrane za izbacivanje dobija se iz polja *disk* deskriptora date stranice. Po završenom vraćanju date stranice na disk, u polje V deskriptora stranice koja je vraćena na disk upisuje se nula.

Dovlači se data stranica sa diska u odabrani blok operativne memorije. Adresa date stranice na disku dobija se iz polja *disk* deskriptora stranice. Po završenom dovlačenju date stranice sa diska u polje *block* deskriptora stranice upisuje se broj bloka, a u polje V jedinica.

Proces za koji je dovučena stranica prevodi se u red radnosposobnih procesa.

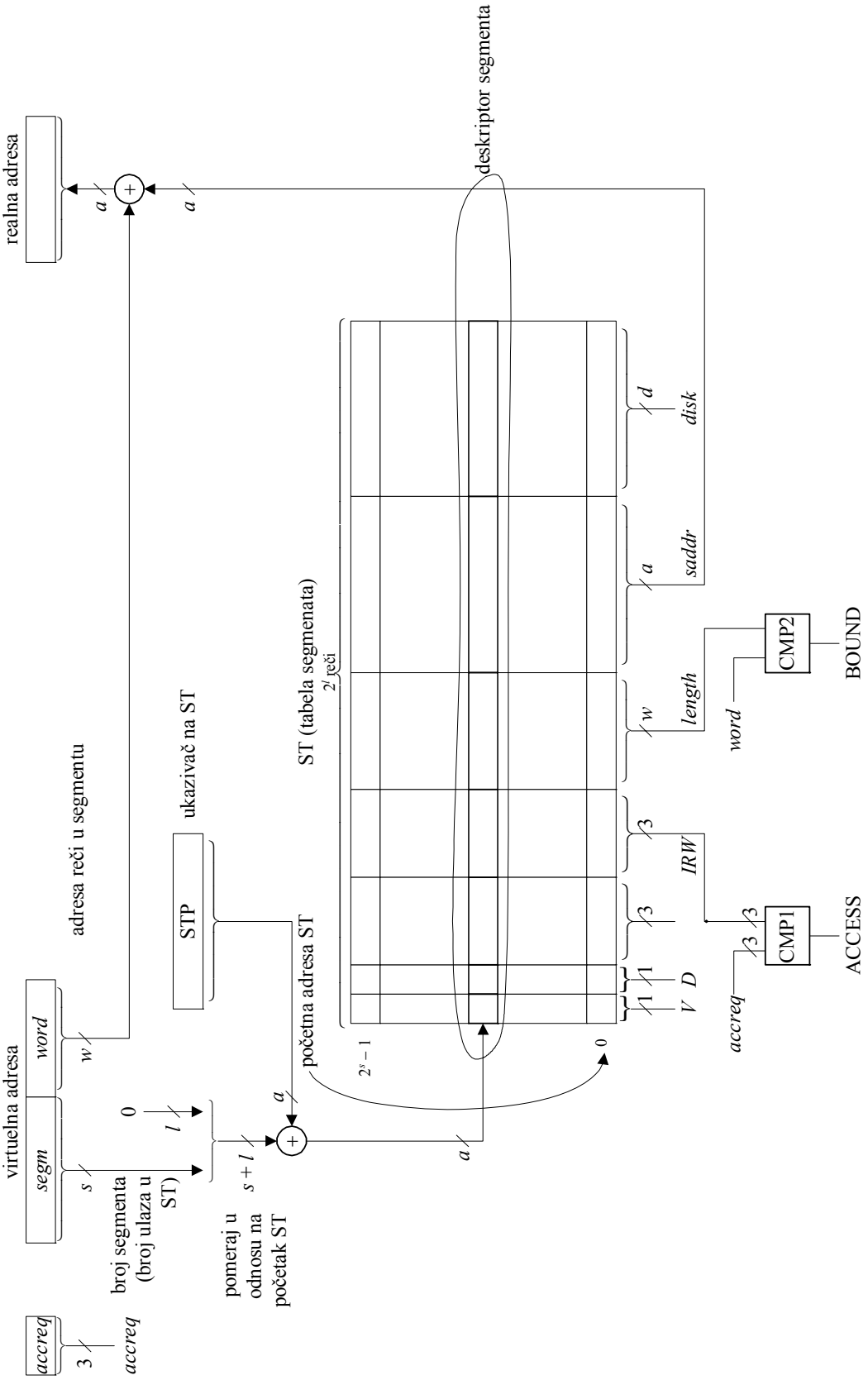
U nekom trenutku dati proces ponovo dobija procesor. Dati proces ponovo generiše adresu za koju je bilo utvrđeno da je iz stranice koja nije bila u memoriji. Pošto je sada data stranica u memoriji ponavljaju se koraci za slučaj kada je stranica u memoriji.

5.3.1.2. Segmentna organizacija

Kod segmentne organizacije virtuelne memorije virtuelna adresa ima dva polja: broj segmenta (*segm*) dužine *s* bita i adresa reči u segmentu (*word*) dužine *w* bita. Kako je veličina virtuelnog adresnog prostora 2^{s+w} reči i sastoji se od 2^s segmenata maksimalne veličine 2^w reči, to su veličine polja *segm* i *word* *s* i *w* bita, respektivno. Kako je veličina realnog adresnog prostora 2^a reči to je veličina realne adrese *a* bita. Struktura virtuelne i realne adrese i dužine u bitovima delova virtuelne i realne adrese su prikazani na slici 11.

Tabela segmenata (ST) jednog procesa je data na slici 11. Tabela segmenata ima poseban ulaz za svaki segment procesa. U njima se nalaze informacije neophodne za preslikavanje segmenata virtuelnog adresnog prostora procesa u delove fizičke memorije veličine segmenata. Te informacije se nazivaju deskriptori segmenata. S obzirom da u virtuelnom adresnom prostoru procesa ima 2^s segmenata, to i tabela segmenata ima 2^s ulaza. Početna adresa tabele segmenata sadržana je u posebnom registru procesora koji se naziva ukazivač na ST (STP). Na osnovu sadržaja ukazivača na ST i broja segmenta, jedinica za preslikavanje virtuelnih adresa u realne adrese dolazi do deskriptora segmenta, a time i do informacija neophodnih za preslikavanje.

U fizičkoj memoriji postoji posebna tabela segmenata svakog procesa. Početne adrese tabela segmenata svih procesa čuva operativni sistem. Prilikom prebacivanja procesora sa procesa na procesa operativni sistem, najpre, čuva kontekst procesa kome se oduzima procesor, a potom, restaurira kontekst procesa kome se dodeljuje procesor. Tom prilikom se u ukazivač na ST upisuje početna adresa tabele segmenata procesa kome se dodeljuje procesor. Time se obezbeđuje da jedinica za preslikavanje pristupa tabeli segmenata procesa kome je dodeljen procesor.



Slika 11 Virtuelna adresa, realna adresa i tabela stranica za segementnu organizaciju virtuelne memorije

Polja deskriptora segmenta su:

V (1 bit)—segment u memoriji,

D (1 bit)—segment modifikovan,

IRW (3 bita)—bitovi prava pristupa,

length (*w* bita)—veličina segmenta,

saddr (*a* bita)—početna adresa segmenta u operativnoj memoriji i

disk (*d* bita)—adresa segmenta na disku.

Polje *V* označava da li je dati segment u operativnoj memoriji. Postavlja ga operativni sistem prilikom dovlačenja datog segmenta sa diska u operativnu memoriju. Koristi ga jedinica za preslikavanje i to ako

je postavljen, da formira realnu adresu i

nije postavljen, da generiše prekid.

Polje *D* označava da li je dati segment modifikovan. Postavlja ga jedinica za preslikavanje ako je bilo operacija upisa u dati segment. Koristi ga operativni sistem onda kada odabere dati segment za zamenu i to ako

je postavljen, da dati segment vrati na disk i

nije postavljen, da dati segment ne vraća na disk.

Polje *IRW* označava dozvoljen pristup datom segmentu. Uzeto je da

postavljen bit *I* označava da iz njega smeju da se čitaju samo instrukcije,

postavljen bit *R* označava da iz njega smeju da se čitaju podaci i

postavljen bit *W* označava da iz njega smeju da se upisuju podaci.

Polje *IRW* formira operativni sistem prilikom formiranja tabele segmenata datog procesa. Koristi ga jedinica za preslikavanje da, upoređivanjem sa sadržajem registra *accreg*, utvrdi da li zahtevani pristup segmentu odgovara dozvoljenom pristupu segmentu.

Polje *length* označava veličinu segmenta. Polje *length* postavlja operativni sistem prilikom formiranja tabele segmenata datog procesa. Koristi ga jedinica za preslikavanje da, upoređivanjem sa sadržajem polja *word* virtuelne adrese, utvrdi da li je adresa reči u segmentu unutar specificirane veličine segmenta.

Polje *saddr* označava početnu adresu dela operativne memorije u kome se nalazi dati segment. Vrednost u ovom polju ima smisla jedino ako je polje *V* postavljeno. Polje *saddr* postavlja operativni sistem prilikom dovlačenja segmenta sa diska u operativnu memoriju. Koristi ga jedinica za preslikavanje, i to ako je polje *V* postavljeno, da formira realnu adresu.

Polje *disk* označava adresu datog segmenta na disku. Polje *disk* postavlja operativni sistem prilikom formiranja tabele segmenata datog procesa. Koristi ga samo operativni sistem i to radi lociranja datog segmenta na disku prilikom

dovlačenja segmenta sa diska u odabrani deo operativne memorije i

vraćanja modifikovanog segmenta iz dela operativne memorije odabranog za zamenu na disk.

Uzeto je da je veličina adrese segmenta na disku *d* bitova.

Preslikavanje virtuelne u realnu adresu za slučaj da je segment u memoriji realizuje se kompletno hardverski i to pomoću jedinice za preslikavanje. Ovo preslikavanje se realizuje u sledećim koracima:

Polje *segm* virtuelne adrese predstavlja broj ulaza u tabelu segmenata u kome se nalazi deskriptor datog segmenta. S obzirom na to da deskriptor segmenta zauzima 2^l reči potrebno je broj ulaza u tabelu segmenata pretvoriti u pomeraj u odnosu na početak tabele segmenata. To se realizuje pomeranjem ulevo za *l* mesta sadržaja polja *segm* vitruelne adrese.

Sadržaj registra ukazivač na ST predstavlja početnu adresu tabele segmenata. Pomeraj u odnosu na početak tabele segmenata formiran u prethodnom koraku se sabira sa sadržajem registra ukazivač na ST i time dobija adresa deskriptora datog segmenta. Počev od formirane adrese treba očitati odgovarajući broj reči, jer se u njima nalaze polja *V*, *IRW*, *length* i *saddr* koja koristi jedinica za preslikavanje.

Polje *V* deskriptora ukazuje na to da li je dati segment u memoriji. Stoga se, najpre, sa adrese formirane u prethodnom koraku čita prva reč u kojoj se nalazi polje *V* deskriptora i vrši provera da li je ovo polje postavljeno. Za slučaj kada je segment u memoriji utvrđuje se da je polje *V* postavljeno, pa se produžava sa sledećim koracima.

Polja *IRW*, *length* i *saddr* deskriptora sadrže bitove prava pristupa, veličinu segmenta i početnu adresu segmenta u fizičkoj memoriji, respektivno. Stoga se sada čita potreban broj reči deskriptora da bi se do njih došlo. Sada se istovremeno:

sabiranjem polja *saddr* iz deskriptora segmenta i polja *word* iz virtuelne adrese formira realna adresa,

upoređivanjem polja *IRW* iz deskriptora segmenta i sadržaja registra *accreg* utvrđuje da li se zahteva pristup koji je dozvoljen za dati segment i

upoređivanjem polja *length* iz deskriptora segmenta i polja *word* iz virtuelne adrese utvrđuje da li je adresa u segmentu unutar specificirane veličine segmenta.

Ukoliko su signali **ACCESS** i **BOUND** neaktivni, zahtevani pristup segmentu i virtuelna adresa su korektni, pa se formirana realna adresa koristi za pristup operativnoj memoriji. U suprotnom nema obraćanja operativnoj memoriji već se generiše prekid. Sve ostalo, što za ovaj slučaj potom treba uraditi, radi operativni sistem i to u sledećim koracima:

Oduzima procesor datom procesu, njegov kontekst čuva i terminira procesa.

Šalje poruku na konzolu da je dati proces terminiran.

Dodeljuje procesor nekom od radnosposobnih procesa. Tada se restaurira njegov kontekst. U okviru toga se u programski brojač upisuje nova vrednost i time kontrola predaje tom novom procesu.

U slučaju da segment nije u memoriji realizuju se samo prva tri od četiri koraka za slučaj kada je segment u memoriji. U koraku 3. se u ovom slučaju utvrđuje da polje *V* nije postavljeno, pa se generiše prekid. Svi koraci do generisanja signala prekida, uključujući i njegovo generisanje, realizuju se hardverski pomoću jedinice za preslikavanje. Sve ostalo, što za ovaj slučaj potom treba uraditi, radi operativni sistem i to u sledećim koracima:

Oduzima procesor datom procesu, njegov kontekst čuva i proces stavlja u red blokiranih procesa.

Organizuje dovlačenje datog segmenta sa diska u neki od delova operativne memorije veličine segmenta.

Dodeljuje procesor nekom od radnosposobnih procesa. Tada se restaurira njegov kontekst. U okviru toga se u programski brojač upisuje nova vrednost i time kontrola predaje tom novom procesu.

Dovlačenje segmenta sa diska u neki od delova operativne memorije se realizuje u sledećim koracima:

Traži se deo operativne memorije u koji će se smestiti dati segment. Pri tome se izvršava jedan od sledeća dva koraka:

Ukoliko ima slobodnih delova operativne memorije po nekom algoritmu se odlučuje koji deo treba dodeliti datom segmentu.

Ukoliko nema slobodnih delova operativne memorije po nekom algoritmu se odlučuje koji segment treba izbaciti iz operativne memorije da bi se deo operativne memorije u kome se on nalazi oslobodio i dodelio datom segmentu. Ukoliko je segment odabran za izbacivanje modifikovan mora se vratiti na disk pre nego što se deo operativne memorije u kome se on nalazi koristi da se u njega dovuče novi segment. Adresa na disku segmenta odabranog za izbacivanje dobija se iz polja *disk* deskriptora segmenta. Po završenom vraćanju datog segmenta na disk, u polje V deskriptora segmenta koji je vraćen na disk upisuje se nula.

Dovlači se dati segment sa diska u odabrani deo operativne memorije. Adresa datog segmenta na disku dobija se iz polja *disk* deskriptora segmenta. Po završenom dovlačenju datog segmenta sa diska u polje *saddr* deskriptora segmenta upisuje se početna adresa dela segmenta u operativnoj memoriji, a u polje V jedinica.

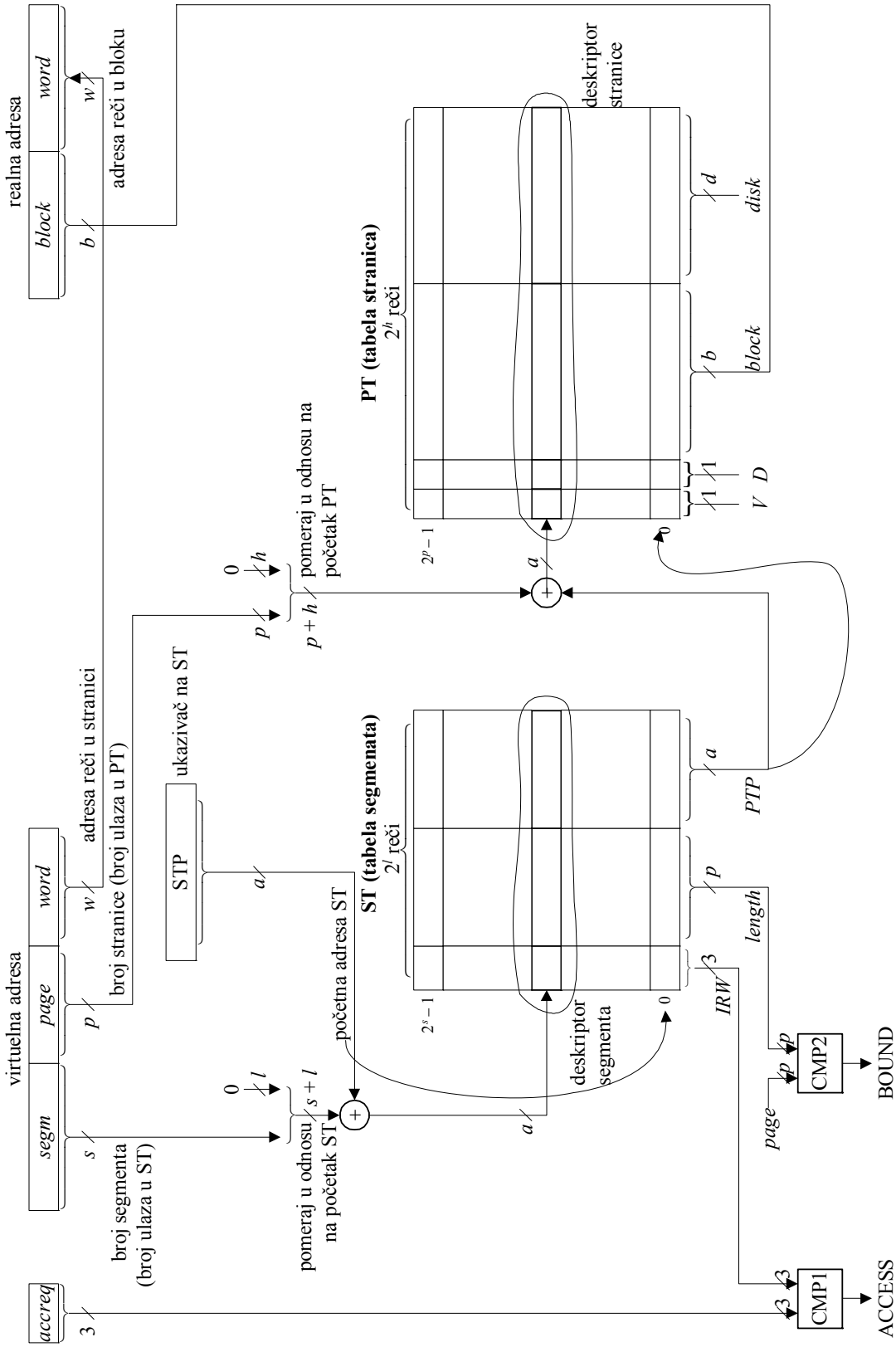
Proces za koga je dovučen segment prevodi se u red radnosposobnih procesa.

U nekom trenutku dati proces ponovo dobija procesor. Dati proces ponovo generiše adresu za koju je bilo utvrđeno da je iz segmenta koji nije bio u operativnoj memoriji. Pošto je sada dati segment u operativnoj memoriji ponoviće se koraci za slučaj kada je segment u memoriji.

5.3.1.3. Segmentno-stranična organizacija

Kod segmentno-stranične organizacije virtuelne memorije virtuelna adresa ima tri polja: broj segmenta (*segm*) dužine *s* bita, broj stranice (*page*) dužine *p* bita i adresa reči u stranici (*word*) dužine *w* bita. Kako je veličina virtuelnog adresnog prostora 2^{s+p+w} reči i sastoji se iz 2^s segmenata maksimalne veličine 2^w reči, a oni su dalje podeljeni na 2^p stranica veličine 2^w reči, to su veličine polja *segm*, *page* i *word* *s*, *p* i *w* bita, respektivno. Kod segmentno-stranične organizacije virtuelne memorije realna adresa ima dva polja: broj bloka (*block*) dužine *b* bita i adresa reči u bloku (*word*) dužine *w* bita. Kako je veličina realnog adresnog prostora 2^{b+w} reči, a bloka 2^w reči, to su veličine polja *block* i *word* *b* i *w* bita, respektivno. Struktura virtuelne i realne adrese i dužine u bitovima delova virtuelne i realne adrese su prikazani na slici 12.

Kod segmentno-stranične organizacije virtuelne memorije svaki proces ima jednu tabelu segmenata i onoliko tabela stranica koliko ima segmenata u virtuelnom adresnom prostoru procesa. Za dati procesor postoji jedna tabela segmenata i maksimalno 2^p tabela stranica.



Slika 12 Virtuelna adresa, realna adresa i tabela stranica za segmentno-straničnu organizaciju virtuelne memorije

Tabela segmenata (ST) jednog procesa je data na slici 12. Tabela segmenata ima poseban ulaz za svaki segment procesa. U njima se nalaze informacije neophodne za preslikavanje segmenata virtuelnog adresnog prostora procesa u blokove fizičke memorije. Te informacije se nazivaju deskriptori segmenata. S obzirom da u virtuelnom adresnom prostoru procesa ima 2^s segmenata, to i tabela segmenata ima 2^s ulaza. Početna adresa tabele segmenata sadržana je u posebnom registru procesora koji se naziva ukazivač na ST (STP). Na osnovu sadržaja ukazivača na ST i broja segmenta, jedinica za preslikavanje virtuelnih adresa u realne adrese dolazi do deskriptora segmenta, a time i do informacija neophodnih za preslikavanje.

U fizičkoj memoriji postoji posebna tabela segmenata svakog procesa. Početne adrese tabela segmenata svih procesa čuva operativni sistem. Prilikom prebacivanja procesora sa procesa na procesa operativni sistem, najpre, čuva kontekst procesa kome se oduzima procesor, a potom, restaurira kontekst procesa kome se dodeljuje procesor. Tom prilikom se u ukazivač na ST upisuje početna adresa tabele segmenata procesa kome se dodeljuje procesor. Time se obezbeđuje da jedinica za preslikavanje pristupa tabeli segmenata procesa kome je dodeljen procesor.

Polja deskriptora segmenta su:

IRW (3 bita)—bitovi prava pristupa,

length (*p* bita)—veličina segmenta i

PTP (*b* + *w* bita)—početna adresa tabele stranica u fizičkoj memoriji.

Polje *IRW* označava dozvoljen pristup datom segmentu. Uzeto je da

postavljen bit *I* označava da iz njega smeju da se čitaju samo instrukcije,

postavljen bit *R* označava da iz njega smeju da se čitaju podaci i

postavljen bit *W* označava da iz njega smeju da se upisuju podaci.

Polje *IRW* formira operativni sistem prilikom formiranja tabele segmenata datog procesa. Koristi ga jedinica za preslikavanje da, upoređivanjem sa sadržajem registra *accreg*, utvrdi da li zahtevani pristup segmentu odgovara dozvoljenom pristupu segmentu.

Polje *length* označava veličinu segmenta u stranicama. Polje *length* postavlja operativni sistem prilikom formiranja tabele segmenata datog procesa. Koristi ga jedinica za preslikavanje da, upoređivanjem sa sadržajem polja *page* virtuelne adrese, utvrdi da li je broj stranice u segmentu unutar specificirane veličine segmenta.

Polje *PTP* označava početnu adresu operativne memorije u kome se nalazi tabela stranica datog segmenta. Polje *PTP* postavlja operativni sistem prilikom formiranja tabele segmenata i tabela stranica. Koristi ga jedinica za preslikavanje da formira realnu adresu.

Uzeto je da se *STP* i *PTP* predstavljaju u brojevima blokova, a *length* u broju stranica.

Tabela stranica (PT) jednog segmenta je data na slici 12. Tabela stranica ima poseban ulaz za svaku stranicu datog segmenta. U njima se nalaze informacije neophodne za preslikavanje stranica segmenta u blokove fizičke memorije. Te informacije se nazivaju deskriptori stranica. S obzirom da segment ima najviše 2^p stranica, to i tabela stranica može da ima najviše 2^p ulaza. Početna adresa tabele stranica sadržana je u polju *PTP* deskriptora datog segmenta. Na osnovu sadržaja polja *PTP* deskriptora datog segmenta i broja stranice, jedinica za preslikavanje virtuelnih adresa u realne adrese dolazi do deskriptora stranice, a time i do informacija neophodnih za preslikavanje.

Polja deskriptora stranice su:

V (1 bit)—stranica u memoriji,

D (1 bit)—stranica modifikovana,

block (*b* bita)—broj bloka i

disk (*d* bita)—adresa na disku.

Polje *V* označava da li je data stranica u operativnoj memoriji. Postavlja ga operativni sistem prilikom dovlačenja date stranice sa diska u operativnu memoriju. Koristi ga jedinica za preslikavanje i to ako

je postavljen, da formira realnu adresu i

nije postavljen, da generiše prekid.

Polje *D* označava da li je data stranica modifikovana. Postavlja ga jedinica za preslikavanje ako je bilo operacija upisa u datu stranicu. Koristi ga operativni sistem onda kada odabere datu stranicu za zamenu i to ako

je postavljen, da datu stranicu vrati na disk

nije postavljen, da datu stranicu ne vraća na disk.

Polje *block* označava broj bloka operativne memorije u kome se nalazi data stranica. Vrednost u ovom polju ima smisla jedino ako je polje *V* postavljeno. Polje *block* postavlja operativni sistem prilikom dovlačenja date stranice sa diska u operativnu memoriju. Koristi ga jedinica za preslikavanje, i to ako je polje V postavljeno, da formira realnu adresu.

Polje *disk* označava adresu date stranice na disku. Polje *disk* postavlja operativni sistem prilikom formiranja tabele stranice datog procesa. Koristi ga samo operativni sistem i to radi lociranja date stranice na disku prilikom

dovlačenja stranice sa diska u odabrani blok operativne memorije

vraćanja modifikovane stranice iz bloka operativne memorije odabranog za zamenu na disk.

Uzeto je da je veličina adrese stranice na disku *d* bitova.

Preslikavanje virtuelne u realnu adresu za slučaj da je stranica u memoriji realizuje se kompletno hardverski i to pomoću jedinice za preslikavanje. Ovo preslikavanje se realizuje u sledećim koracima:

Polje *segm* virtuelne adrese predstavlja broj ulaza u tabelu segmenata u kome se nalazi deskriptor datog segmenta. S obzirom na to da deskriptor segmenta zauzima 2^l reči potrebno je broj ulaza u tabelu segmenata pretvoriti u pomeraj u odnosu na početak tabele segmenata. To se realizuje pomeranjem ulevo za *l* mesta sadržaja polja *segm* vitruelne adrese.

Sadržaj registra ukazivač na ST predstavlja početnu adresu tabele segmenata. Pomeraj u odnosu na početak tabele segmenata formiran u prethodnom koraku se sabira sa sadržajem registra ukazivač na ST i time dobija adresa deskriptora datog segmenta. Počev od formirane adrese treba očitati potreban broj reči, jer se u njima nalaze polja *IRW*, *length* i *PTP* koja koristi jedinica za preslikavanje.

Polja *IRW*, *length* i *PTP* deskriptora sadrže bitove zaštite, veličinu segmenta i početnu adresu tabele stranica u fizičkoj memoriji, respektivno. Stoga se sada čita potreban broj reči da bi se do njih došlo. Sada se istovremeno:

upoređivanjem polja *IRW* iz deskriptora segmenta i sadržaja registra *accreg* utvrđuje da li se zahteva pristup koji je dozvoljen za dati segment i

upoređivanjem polja *length* iz deskriptora segmenta i polja *page* iz virtuelne adrese utvrđuje da li je broj stranice u segmentu unutar specificirane veličine segmenta.

Ukoliko su signali ACCESS i BOUND neaktivni, zahtevani pristup segmentu je korektan, pa se pristupa deskriptoru stranice u tabeli stranica na način prikazan počev od koraka 4. U suprotnom nema obraćanja tabeli stranica već se generiše prekid i prelazi na aktivnosti prikazane počev od koraka 8.

Polje *page* virtuelne adrese predstavlja broj ulaza u tabelu stranica u kome se nalazi deskriptor date stranice. S obzirom na to da deskriptor stranice zauzima 2^h reči potrebno je broj ulaza u tabelu stranica pretvoriti u pomeraj u odnosu na početak tabele stranica. To se realizuje pomeranjem ulevo za *h* mesta sadržaja polja *page* vitruelne adrese.

Sadržaj polja *PTP* koji predstavlja početnu adresu tabele stranica sabira se sa pomerajem u odnosu na početak tabele stranica koji je formiran u prethodnom koraku i time se dobija adresa deskriptora date stranice. Počev od formirane adrese treba očitati potreban broj reči, jer se u njima nalaze polja *V* i *block* koja koristi jedinica za preslikavanje.

Polje *V* deskriptora ukazuje na to da li je data stranica u memoriji. Stoga se, najpre, sa adrese formirane u prethodnom koraku čita prva reč u kojoj se nalazi polje *V* deskriptora i vrši provera da li je ovo polje postavljeno. Za slučaj kada je stranica u memoriji utvrđuje se da je polje *V* postavljeno, pa se produžava sa sledećim koracima.

Polje *block* deskriptora sadrži broj bloka u kome se nalazi data stranica. Stoga se sada čita potreban broj reči deskriptora da bi se do njega došlo. Konkatenacijom polja *block* iz deskriptora stranice i polja *word* iz virtuelne adrese formira se realna adresa.

Ovo je kraj preslikavanja virtuelne u realnu adresu za slučaj da je stranica u memoriji.

U slučaju generisanog prekida, zbog pojave aktivne vrednosti jednog ili oba signala **ACCESS** ili **BOUND**, sve što za ovaj slučaj potom treba uraditi, radi operativni sistem i to u sledećim koracima:

Oduzima procesor datom procesu, njegov kontekst čuva i terminira procesa.

Šalje poruku na konzolu da je dati proces terminiran.

Dodeljuje procesor nekom od radnosposobnih procesa. Tada se restaurira njegov kontekst. U okviru toga se u programski brojač upisuje nova vrednost i time kontrola predaje tom novom procesu.

U slučaju da stranica nije u memoriji realizuju se svi koraci sem koraka 7 kao i za slučaj kada je stranica u memoriji. Moguće su dve situacije. U prvoj situaciji se ide od koraka 1 do koraka 3 u kome se utvrđuje da je jedan od ili oba signala **ACCESS** ili **BOUND** aktivni, pa se prelazi na korak 8. U drugoj situaciji se ide od koraka 1 do koraka 6. U koraku 6 se u ovom slučaju utvrđuje da polje *V* nije postavljeno, pa se generiše prekid. Svi koraci do generisanja signala prekida, uključujući i njegovo generisanje, realizuju se hardverski pomoću jedinice za preslikavanje. Sve ostalo, što za ovaj slučaj potom treba uraditi, radi operativni sistem i to u sledećim koracima:

Oduzima se procesor datom procesu, njegov kontekst čuva i proces stavlja u red blokiranih procesa.

Organizuje se dovlačenje date stranice sa diska u neki od blokova operativne memorije.

Dodeljuje se procesor nekom od radnosposobnih procesa. Tada se restaurira njegov kontekst. U okviru toga se u programski brojač upisuje nova vrednost i time kontrola predaje tom novom procesu.

Dovlačenje stranice sa diska u neki od blokova operativne memorije se realizuje u sledećim koracima:

Traži se blok u koji će se smestiti data stranica. Pri tome se izvršava jedan od sledeća dva koraka:

Ukoliko ima slobodnih blokova po nekom algoritmu se odlučuje koji blok treba dodeliti datoj stranici.

Ukoliko nema slobodnih blokova po nekom algoritmu se odlučuje koju stranicu treba izbaciti iz operativne memorije da bi se blok u kome se ona nalazi oslobodio i dodelio datoj stranici. Ukoliko je stranica odabrana za izbacivanje modifikovana mora se vratiti na disk pre nego što se blok u kome se ona nalazi koristi da se u njega dovuče nova stranica. Adresa na disku stranice odabrane za izbacivanje dobija se iz polja *disk* deskriptora stranice. Po završenom vraćanju date stranice na disk, u polje V deskriptora stranice koja je vraćena na disk upisuje se nula.

Dovlači se data stranica sa diska u odabrani blok operativne memorije. Adresa date stranice na disku dobija se iz polja *disk* deskriptora stranice. Po završenom dovlačenju date stranice sa diska u polje *block* deskriptora stranice upisuje se broj bloka, a u polje V jedinica.

Proces za koji je dovučena stranica prevodi se u red radnosposobnih procesa.

U nekom trenutku dati proces ponovo dobija procesor. Dati proces će ponovo generisati adresu za koju je bilo utvrđeno da je iz stranice koja nije bila u memoriji. Pošto je sada data stranica u memoriji ponoviće se koraci za sličaj kada je stranica u memoriji.

5.3.2. Organizacija jedinica preslikavanja

Postoje tri osnovne vrste jedinica za preslikavanje i to:

jedinica sa asocijativnim preslikavanjem,

jedinica sa direktnim preslikavanjem i

jedinica sa set-asocijativnim preslikavanjem.

Bilo koja vrsta jedinice može da se koristi sa bilo kojom vrstom virtuelne memorije. Međutim, jedinica sa asocijativnim preslikavanjem, prikazana u odeljku **Error! Reference source not found.**, je realizovana sa virtuelnom memorijom stranične organizacije, jedinica sa direktnim preslikavanjem, prikazana u odeljku **Error! Reference source not found.**, je realizovana sa virtuelnom memorijom segmentne organizacije i jedinica sa set-asocijativnim preslikavanjem, prikazana u odeljku **Error! Reference source not found.**, je realizovana sa virtuelnom memorijom segmentno-stranične organizacije. Stoga su opšti principi realizacije tri tipa jedinica za preslikavanje dati za tri različita tipa virtuelnih memorija, saglasno navedenim realizacijama.

5.3.2.1. Jedinica sa asocijativnim preslikavanjem

S obzirom:

da u adresnom prostoru procesa ima 2^p stranica,

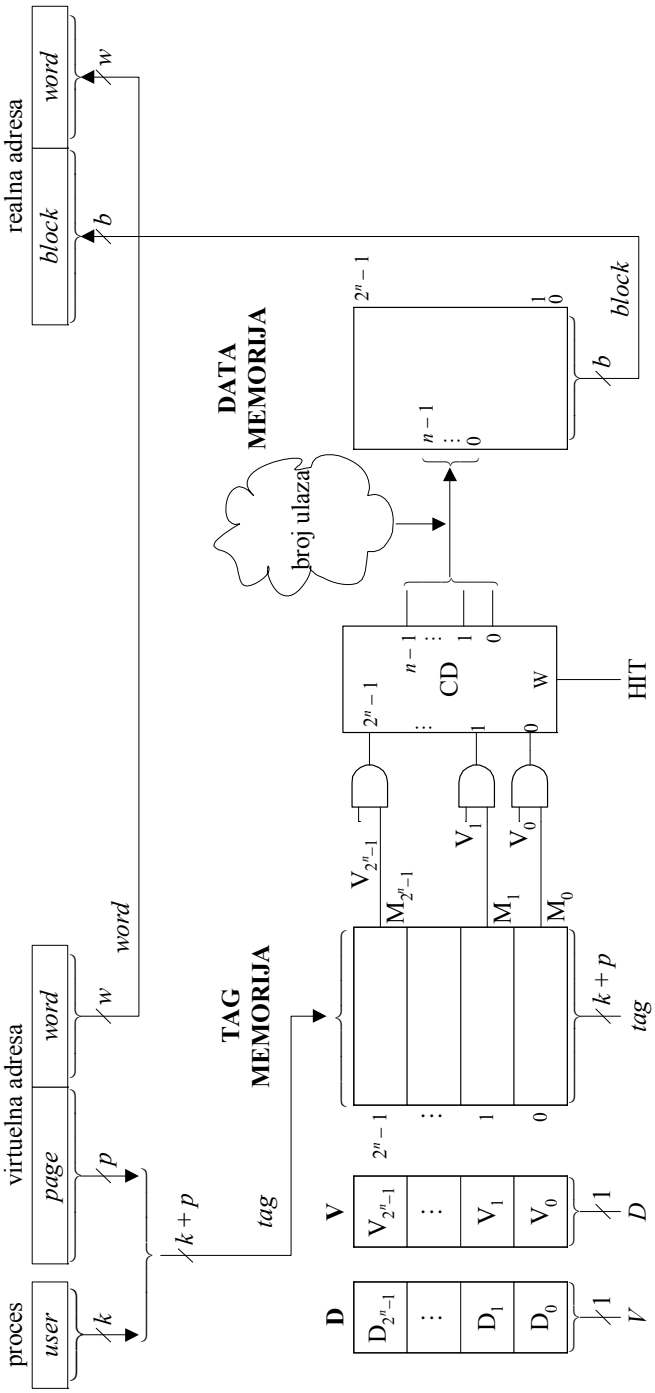
da je broj procesa 2^k i

da se u jedinici za preslikavanje mogu istovremeno čuvati deskriptori stranica različitih procesa,

zaključuje se da je, kao što je prikazano na slici 13, *tag* dužine $k + p$ bita i da je formiran tako da:

k najstarijih bitova predstavljaju broj procesa (*user*) i

p najmlađih bitova predstavljaju broj stranice (*page*) iz virtuelne adrese.



Slika 13 Jedinica sa asocijativnim preslikavanjem za virtuelnu memoriju stranične organizacije

Na osnovu zadatih karakteristika virtuelne memorije i jedinice za preslikavanje dolazi se do strukture jedinice za preslikavanje prikazane na istoj slici.

Jedinica za preslikavanje se sastoji iz sledećih delova:

$D_{0\ldots2^n-1}$ (dirty bitovi)—2^n flip-flopova,

$V_{0\ldots2^n-1}$ (valid bitovi)—2^n flip-flopova,

TAG MEMORIJA—asocijativna memorija kapaciteta 2^n reči širine $k + p$ bitova,

CD—koder i

DATA MEMORIJA—RAM memorija kapaciteta 2^n reči širine b bita.

Dirty bitovi označavaju za svaki od 2^n ulaza jedinice za preslikavanje da li je bilo upisa u neku od lokacija date stranice.

Valid bitovi označavaju za svaki od 2^n ulaza jedinice za preslikavanje da li je odgovarajući ulaz TAG MEMORIJE važeći.

TAG MEMORIJA služi za čuvanje 2^n TAG polja stranica čiji se delovi deskriptora nalaze u odgovarajućim ulazima DATA MEMORIJE i generisanje aktivnih vrednosti signala saglasnosti $\mathbf{M_{0\ldots2^n-1}}$ ukoliko postoji saglasnost TAG polja generisane adrese i sadržaja odgovarajućeg ulaza TAG MEMORIJE.

CD služi za generisanje aktivne vrednosti signala saglasnosti **HIT** i broja ulaza u jedinicu za preslikavanje za koji je otkrivena saglasnost.

DATA MEMORIJA služi za čuvanje 2^n deskriptora stranica.

TAG bitovi ($k + p$) iz generisane adrese se porede sa sadržajima svih 2^n ulaza u TAG MEMORIJI. Ako se sadržaj bilo kojeg ulaza slaže sa TAG bitovima generisane adrese i odgovarajući V bit je postavljen, signal saglasnosti HIT postaje aktivan.

Bitovi (n) koji označavaju broj ulaza u jedinicu za preslikavanje gde je otkrivena saglasnost dobijeni sa izlaza kodera se koriste kao adresa u DATA MEMORIJI i deskriptor se čita.

Očitano polje *block* daje b najstarijih bitova a polje *word* iz virtuelne adrese w najmlađih bitova realne adrese.

Sve ove aktivnosti se realizuju hardverski.

Da bi se stranice različitih procesa koje imaju isti broj preslikale svaka u svoj blok, u formiranju *tag* polja učestvuje ne samo polje *page* generisane adrese, već i vrednost registra *user* procesora. Kod prebacivanja procesora sa procesa na proces u ovaj registar procesora se upisuje broj procesa kome se dodeljuje procesor.

Ukoliko se utvrdi da se relevantni deskriptor ne nalazi u hardveru, ide se hardverski u tabelu stranica i čita deskriptor na način prikazan u odeljku 5.3.1.1. Ukoliko se utvrdi da se stranica nalazi u operativnoj memoriji, polje *block* deskriptora se dovlači u hardver. Korišćenjem FIFO ili LRU algoritma zamene bira se ulaz jedinice za preslikavanje u koji se smešta deskriptor. U dati ulaz DATA MEMORIJE se upisuje polje *block* deskriptora. U isti ulaz TAG MEMORIJE se upisuje polje *tag* generisane adrese. U isti ulaz *V* flip-flopova se upisuje 1, a u isti ulaz *D* flip-flopova se upisuje 0. Sve ovo se realizuje hardverski. Ukoliko se utvrdi da se stranica ne nalazi u operativnoj memoriji, generiše se prekid. Operativni sistem dovlači datu stranicu sa diska na način prikazan u odeljku 5.3.1.1.

5.3.2.2. Jedinica sa direktnim preslikavanjem

S obzirom:

da u virtuelnom adresnom prostoru procesa ima 2^s segmenata,

da je broj procesa 2^k,

da se u jedinici za preslikavanje mogu istovremeno čuvati deskriptori segmenata različitih procesa,

da je jedinica za preslikavanje realizovana u tehnici direktnog preslikavanja i

da se u jedinici za preslikavanje mogu istovremeno čuvati relevantni delovi deskriptora 2^m najčešće korišćenih segmenata do 2^k procesa,

onda je 2^s segmenata svih 2^k procesa grupisano u 2^m grupa sa po 2^{k+s-m} segmenata u grupi. Zaključuje se, kao što je prikazano na slici 14, da su:

polje *tag* dužine $k + s - m$ bita i

polje *ulaz* dužine m bita.

Na osnovu zadatih karakteristika virtuelne memorije i jedinice za preslikavanje dolazi se do strukture jedinice prikazane na istoj slici.

Jedinica za preslikavanje se sastoji iz sledećih delova:

TAG MEMORIJA—RAM memorija kapaciteta 2^m reči širine $1 + 1 + (k + s - m) + 3 + w + a$ bita, koja se sastoji iz polja:

V (valid bitovi) dužine 1 bit,

D (dirty bitovi) dužine 1 bit,

tag dužine $k + s - m$ bita,

IRW dužine 3 bita,

length dužine w bita i

addr dužine a bita,

CMP—$(k + s - m)$-bitni komparator,

CMP1—3-bitni komparator i

CMP2—w-bitni komparator.

TAG MEMORIJA služi za čuvanje delova *IRW*, *length* i *addr* 2^m deskriptora segmenata zajedno sa odgovarajućim *V*, *D* i *tag* bitovima. Značenje ovih bitova je sledeće:

V bitovi označavaju za svaki od 2^m ulaza jedinice za preslikavanje da li je odgovarajući ulaz TAG MEMORIJE važeći,

D bitovi označavaju za svaki od 2^m ulaza jedinice za preslikavanje da li je bilo upisa u neku od lokacija datog segmenta,

tag bitovi služe za čuvanje 2^m *tag* polja segmenata čiji se delovi deskriptora nalaze u poljima *IRW*, *length* i *addr* TAG MEMORIJE,

IRW bitovi označavaju dozvoljen pristup datom segmentu,

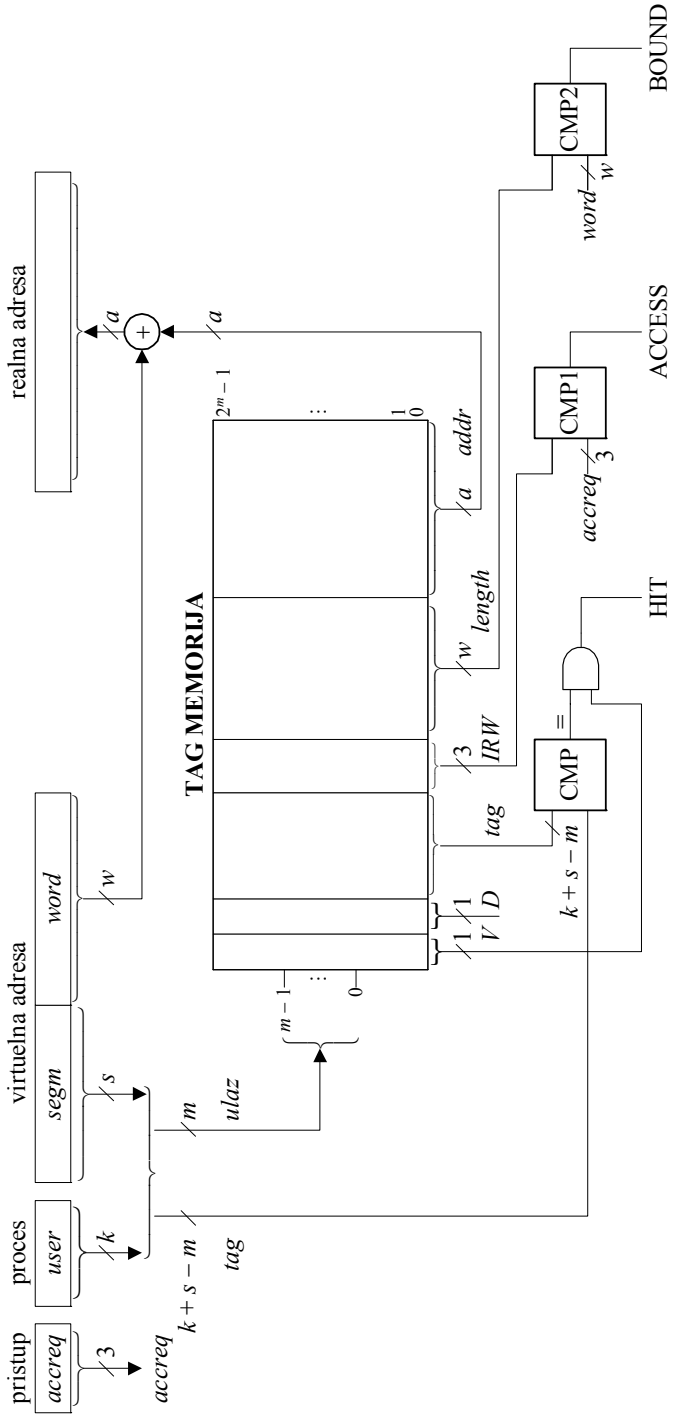
length bitovi označavaju dužinu segmenta i

addr bitovi označavaju početnu adresu dela operativne memorije u kome se nalazi dati segment.

CMP služi za generisanje aktivne vrednosti signala saglasnosti **HIT** ukoliko:

tag polje generisane adrese je isto kao sadržaj *tag* polja TAG MEMORIJE adresiran poljem *ulaz* generisane adrese i

V bit adresiran poljem *ulaz* generisane adrese je postavljen.



Slika 14 Jedinica sa direktnim preslikavanjem za virtuelnu memoriju segmentne organizacije

CMP1 služi za generisanje aktivne vrednosti signala **ACCESS** ukoliko je sadržaj registara *accreg* isti kao sadržaj polja *IRW* TAG MEMORIJE adresiran poljem *ulaz* generisane adrese.

CMP2 služi za generisanje aktivne vrednosti signala **BOUND** ukoliko je sadržaj polja *word* virtuelne adrese manji od sadržaja polja *length* TAG MEMORIJE adresiran poljem *ulaz* generisane adrese.

Bitovi *ulaz* iz generisane adrese se koriste kao adresa za TAG MEMORIJU. Bitovi *tag* očitani iz TAG MEMORIJE se porede sa bitovima *tag* iz generisane adrese. Ako se otkrije da postoji saglasnost i da je *V* bit, adresiran bitovima *ulaz* generisane adrese, postavljen, signal saglasnosti **HIT** postaje aktivan.

Očitano polje *addr* i polje *word* iz virtuelne adrese se sabiraju i daju *a* bitova realne adrese.

Sve ove aktivnosti se realizuju hardverski.

Da bi se segmenti različitih procesa koji imaju isti broj preslikali svaki u svoj deo memorije, u formiranju *tag* polja učestvuje ne samo polje *segm* generisane adrese, već i vrednost registra *user* procesora. Kod prebacivanja procesora sa procesa na proces u ovaj registar procesora se upisuje broj procesa kome se dodeljuje procesor.

Ukoliko se utvrdi da se relevantni delovi deskriptora ne nalaze u jedinici za preslikavanje, ide se hardverski u tabelu segmenata i čita deskriptor na način prikazan u odeljku 5.3.1.2. Ukoliko se utvrdi da se segment nalazi u operativnoj memoriji, polja *IRW*, *length* i *addr* deskriptora se dovlače u jedinicu za preslikavanje. U ulaz DATA MEMORIJE određen *ulaz* bitovima virtuelne adrese se upisuju polja *IRW*, *length* i *addr* deskriptora. U isti ulaz TAG MEMORIJE se upisuje polje *tag* generisane adrese. U isti ulaz *V* bitova se upisuje 1, a u isti ulaz *D* bitova se upisuje 0. Sve ovo se realizuje hardverski. Ukoliko se utvrdi da se segment ne nalazi u operativnoj memoriji, generiše se prekid. Operativni sistem dovlači dati segment sa diska na način prikazan u zadatku 5.3.1.2.

5.3.2.3. Jedinica sa set-asocijativnim preslikavanjem

Jedinica sa set-asocijativnim preslikavanjem se razmatra za slučaj da postoje dva ulaza po setu, jer je to slučaj koji se najčešće sreće u praktičnim realizacijama.

S obzirom da u adresnom prostoru procesa ima 2^{s+p} stranica, da je broj procesa 2^k, da se u jedinici za preslikavanje mogu istovremeno čuvati deskriptori stranica različitih procesa, da je broj setova u jedinici za preslikavanje 2^m i da postoje dva ulaza po setu (2^1 ulaza), onda je 2^{s+p} stranica svih 2^k procesa grupisano u $2^{k+s+p-m}$ grupa sa po 2^m stranica u grupi. Zaključuje se da je polje TAG dužine $k + s + p - m$ bita i da je polje *set* dužine m bita kao što je prikazano na slici 15.

Na osnovu zadatih karakteristika virtuelne memorije i jedinice za preslikavanje dolazi se do strukture jedinice prikazane na istoj slici.

Jedinica za preslikavanje se sastoji iz sledećih delova:

TAG MEMORIJA ULAZA 0—RAM memorija kapaciteta 2^m reči širine $1 + 1 + (k + s + p - m) + 3 + b$ bita, koja se sastoji iz polja:

D (dirty bitovi) dužine 1 bit,

V (valid bitovi) dužine 1 bit,

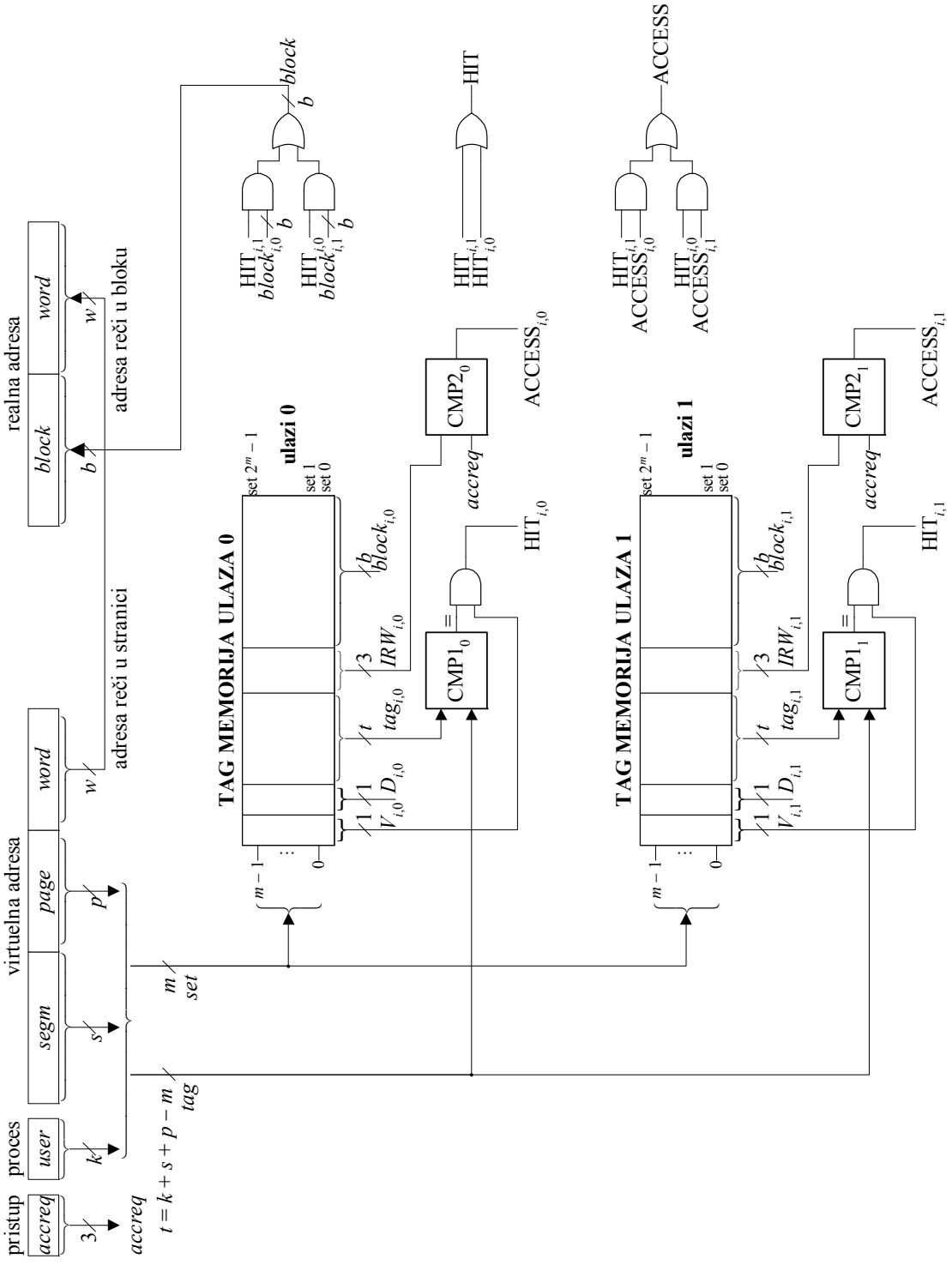
tag dužine $k + s + p - m$ bita,

block dužine m bita i

TAG MEMORIJA ULAZA 1—RAM memorija kapaciteta 2^m reči širine $1 + 1 + (k + s + p - m) + 3 + b$ bita, koja se sastoji iz istih polja kao i TAG MEMORIJA ULAZA 0.

CMP_0—komparator TAG MEMORIJE ULAZA 0.

CMP_1—komparator TAG MEMORIJE ULAZA 1.



Slika 15 Jedinica sa set-asocijativnim preslikavanjem za virtuelnu memoriju segmentno-stranične organizacije

TAG MEMORIJA ULAZA 0 služi za čuvanje 2^m *block* polja deskriptora stranica zajedno sa odgovarajućim *V*, *D* i *tag* bitovima setova SET_0 do SET_{2^m-1} smeštenih u ulaze 0 jedinice za preslikavanje.

Bitovi *D* označavaju za svaki od 2^m setova ulaza 0 jedinice za preslikavanje da li je bilo upisa u neku od lokacija date stranice.

Bitovi *V* označavaju za svaki od 2^m setova ulaza 0 jedinice za preslikavanje da li je odgovarajući ulaz TAG MEMORIJE ULAZA 0 važeći.

Bitovi *tag* služe za čuvanje 2^m *tag* polja stranica čija se *block* polja deskriptora nalaze u odgovarajućim *block* poljima TAG MEMORIJE ULAZA 0.

Bitovi *block* služe za čuvanje 2^m *block* polja deskriptora stranica koja se nalaze u ulazima 0 jedinice za preslikavanje.

CMP_0 služi za generisanje aktivne vrednosti signala saglasnosti $\mathbf{HIT_{i,0}}$ za ulaze 0 jedinice za preslikavanje ukoliko:

tag polje generisane adrese je isto kao sadržaj $tag_{i,0}$ polja TAG MEMORIJE ULAZA 0 adresiran poljem *set* generisane adrese i

$V_{i,0}$ bit adresiran poljem *set* generisane adrese je postavljen.

TAG MEMORIJA ULAZA 1 služi za čuvanje 2^m *block* polja deskriptora stranica zajedno sa odgovarajućim *V*, *D* i *tag* bitovima setova SET_0 do SET_{2^m-1} smeštenih u ulaze 1 jedinice za preslikavanje. Funkcije polja *V*, *D*, *tag* i *block* su iste kao u slučaju TAG MEMORIJE ULAZA 0.

CMP_1 služi za generisanje aktivne vrednosti signala saglasnosti $\mathbf{HIT_{i,1}}$ za ulaze 1 jedinice za preslikavanje na isti način kao signal saglasnosti $\mathbf{HIT_{i,0}}$.

Bitovi *set* iz generisane adrese se koriste kao adresa za TAG MEMORIJU ULAZA 0 i TAG MEMORIJU ULAZA 1. Bitovi $tag_{i,0}$ očitani iz TAG MEMORIJE ULAZA 0 se porede sa bitovima *tag* iz generisane adrese. Ako se otkrije da postoji saglasnost i da je bit $V_{i,0}$, adresiran bitovima *set* generisane adrese, postavljen, signal saglasnosti $\mathbf{HIT_{i,0}}$ postaje aktivan. Bitovi $tag_{i,1}$ očitani iz TAG MEMORIJE ULAZA 1 se porede sa bitovima *tag* iz generisane adrese. Ako se otkrije da postoji saglasnost i da je bit $V_{i,1}$, adresiran bitovima *set* generisane adrese, postavljen, signal saglasnosti $\mathbf{HIT_{i,1}}$ postaje aktivan. Signal saglasnosti **HIT** je aktivan ukoliko je aktivan jedan od signala $\mathbf{HIT_{i,0}}$ i $\mathbf{HIT_{i,1}}$.

U formiranju *b* najstarijih bitova realne adrese učestvuju bitovi $block_{i,0}$ iz TAG MEMORIJE ULAZA 0 ukoliko je signal $\mathbf{HIT_{i,0}}$ aktivan, odnosno bitovi $block_{i,1}$ iz TAG MEMORIJE ULAZA 1 ukoliko je signal $\mathbf{HIT_{i,1}}$ aktivan. Polje *word* iz virtuelne adrese daje *w* najmlađih bitova realne adrese.

Sve ove aktivnosti se realizuju hardverski.

Da bi se stranice različitih procesa koje imaju isti broj preslikale svaka u svoj blok, u formiranju polja *tag* učestvuje i vrednost registra *user* procesora. Kod prebacivanja procesora sa procesa na proces u ovaj registar procesora se upisuje broj procesa kome se dodeljuje procesor.

Ukoliko se utvrdi da se relevantni deskriptor ne nalazi u hardveru, ide se hardverski u tabelu stranica i čita deskriptor na način prikazan u odeljku 5.3.2.3. Ukoliko se utvrdi da se stranica nalazi u operativnoj memoriji, polje *block* deskriptora se dovlači u hardver. Polje *set* generisane adrese određuje u koji od 2^m seta se smešta polje *block* deskriptora. Pošto za svaki set postoje ulazi 0 i 1, korišćenjem FIFO ili LRU algoritma zamene u okviru datog seta bira se

ulaz jedinice za preslikavanje u koji se smešta deskriptor. U polje *block* datog ulaza jedinice za preslikavanje se upisuje polje *block* deskriptora. U isti ulaz se upisuje polje *tag* generisane adrese. U isti ulaz *V* bitova se upisuje 1, a u isti ulaz *D* bitova se upisuje 0. Sve ovo se realizuje hardverski. Ukoliko se utvrdi da se stranica ne nalazi u operativnoj memoriji, generiše se prekid. Operativni sistem dovlači datu stranicu sa diska na način prikazan u odeljku 5.3.2.3.

5.4. POVEZIVANJE KEŠ MEMORIJE, JEDINICE ZA UBRZAVANJE, U/I UREĐAJA, GLAVNE I SEKUNDARNE MEMORIJE

U računarima sa keš memorijom pažnju zaslužuju još neka pitanja od kojih su tri, razmatrana u ovom odeljku, najinteresantnija. Prvo pitanje se odnosi na povezivanje keš memorije i TLB jedinice. Ovde je moguće, prvo, u TLB jedinici prevesti virtuelnu u realnu adresu, a onda se realnom adresom pristupiti kešu. Moguće je i sa virtuelnom adresom pristupati keš memoriji, pa tek kada treba pristupiti memoriji, prevoditi virtuelnu u realnu adresu. Drugo pitanje se odnosi na mogućnost da, zbog toga što podaci mogu da budu i u keš memoriji i u memoriji, procesor ili U/I uređaji mogu da rade sa neažurnom kopijom podatka. Treće pitanje se odnosi na izbor interfejsa za pristup memoriji, koji će omogućiti efikasan rad keša.

5.4.1. Keš memorija i virtuelna memorija

Kod procesora sa virtuelnom memorijom adrese koje se generišu su virtuelne pa postoji potreba za njihovim prevođenjem u fizičke adrese svaki put kada se pristupa fizičkoj memoriji. Ukoliko u procesoru postoji i keš memorija postupak prevođenja virtuelnih u realne adrese i pristup keš memoriji je moguće realizovati na dva načina:

najpre se protiupa keš memoriji, pa se tek kod pristupa operativnoj memoriji vrši prevođenje virtuelnih u realne adrese, ili

najpre se vrši prevođenje virtuelnih u realne adrese, pasž se tek posle toga pristupa keš memoriji.

U prvom slučaju u keš memoriji se nalaze virtuelne adrese, pa se ovakva realizacija keš memorije naziva i virtuelna keš memorija. U drugom slučaju u keš memoriji se nalaze realne adrese, pa se ovakva realizacija keš memorije naziva i realna keš memorija.

Posmatrano sa aspekta performansi keš memorije virtuelni keš je bolji nego realni keš. Tada u slučaju “hit”-a koji se daleko češće javlja nego “miss,” ne bi bilo prevođenja virtuelne u realnu adresu. U suprotnom slučaju “hit access time” bi trebalo stalno uvećavati za vreme prevođenja virtuelne u realnu adresu. Virtuelni keš, međutim, unosi određene probleme za koje su potrebna određena rešenja.

5.4.1.1. Virtuelni keš

Prvi problem je posledica činjenice da različiti korisnici imaju iste virtuelne adrese koje se preslikavaju na različite fizičke adrese. Na primer, svaki korisnik u slučaju stranične organizacije virtuelne memorije ima stranice broj nula, jedan, itd. koje se, međutim, preslikavaju u različite blokove. Isto to važi i za segmente sa brojevima nula, jedan itd. koji se, međutim, preslikavaju u različite delove memorije. Stoga bi po porebacivanju procesora sa korisnika na korisnika novi korisnik mogao da utvrdi da se potreban blok, recimo stranica

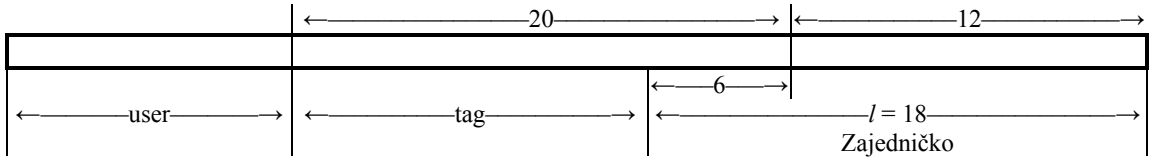
nula, nalazi u kešu i da mu pogrešno pristupi, kako je to blok sa istim brojem stranice nula koji je u kešu ostao od prethodnog korisnika. **Tu postoje dva rešenja. Prvo** je da se svi ulazi keš memorije proglase za nevažeće pri prebacivanju procesora sa korisnika na korisnika. **Drugo** je da se **tag** proširi sa PID (proces identifikator) brojem. U prvom slučaju se kvari "miss rate," a u drugom povećava dužina "tag"-a.

Drugi problem sa virtuelnim kešom se sastoji u tome da različiti korisnici mogu da koriste različite virtuelne adrese za istu fizičku adresu. To se dešava u slučaju stranične organizacije virtuelne memorije u slučaju deljivih segmenata. Ove dvostruke adrese, nazvane **sinonimi** ili **aliasi**, mogu da dovedu do toga da se u virtuelnom kešu pojave dve kopije istog podatka. U slučaju realnog keša ovo ne može da se desi, zato što bi različite virtuelne adrese uvek bile prevedene na istu fizičku adresu.

Ovaj problem se može rešiti kombinacijom hardvera i softvera, korišćenjem tehnike bojenja stranica (page colouring). Kod **page colouring**-a softverski se obezbeđuje da aliasi imaju određen broj bitova virtuelnih adresa identične. Ako je taj broj l i ako je kapacitet keš memorije 2^l bajta ili manji i koristi direktno preslikavanje, tada će blokovi stranica aliasa zauzimati isti ulaz u virtuelnom kešu. Stoga se ne može desiti da se u virtuelnom kešu nađu dve kopije fizičkog podatka.

Napomena: ovde je u kešu jedan blok za različite virtuelne adrese, ali se nepotrebno prebacuje iz keš u operativnu memoriju i nazad, pri promeni virtuelne adrese alias-a.

Moguće je poboljšanje ovoga (strana 425 nejasna).



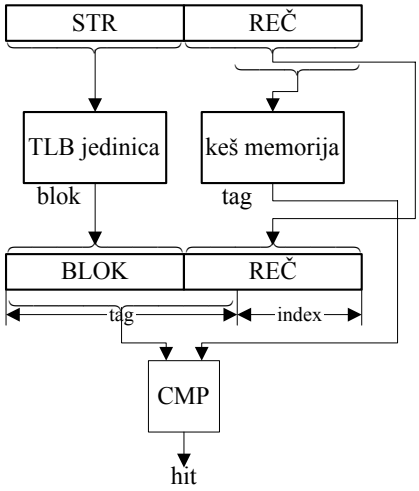
Kod virtuelnog keša sa direktnim preslikavanjem i kapaciteta 256 KB ili manje preslikavaju se u isti ulaz, a razlikuju u **user** i **tag** bitovima.

Poseban problem kod keš memorija sa virtuelnim adresama je vezan za ulaz/ozlaz. Kod ulaza/izlaza se obično radi sa fizičkim adresama, pa bi za interakciju sa virtuelnim kešom bilo potrebno njihovo preslikavanje u virtuelne adrese. Ovo pitanje se detaljno obrađuje u poglavlju sa magistralama.

5.4.1.2. Realni keš

U slučaju realnog keša pre pristupa kešu mora, najpre, da se izvrši prevođenje virtuelnih u realne adrese. To bi povećalo "hit time" i usporilo rad procesora. Stoga se u procesorima sa realnim kešom koriste određene tehnike kojima se ovaj negativan efekat ublažava.

Kod jedne od tih tehnika se istovremeno i vrši preslikavanje virtuelne u realnu adresu i pristupa realnom kešu (slika 16). Da bi to moglo da se realizuje uvodi se ograničenje da veličina keš memorije ne bude veća od veličine stranice. To omogućava da se istovremeno pristupa i jedinici za ubrzavanje preslikavanja i keš memoriji. Jedinici za ubrzavanje preslikavanja se pristupa na osnovu broja stranice da bi se dobio broj bloka realne adrese. Keš memoriji se pristupa na osnovu adrese reči da bi se dobio "tag" koji je realan. Ove operacije se realizuju paralelno. Po njihovom kompletiranju vrši se upoređivanje realnog "tag"-a dobijenog iz realne keš memorije i "tag" bitova realne adrese dobijene iz jedinice za ubrzavanje preslikavanja. Ako postoji saglasnost pristupa se keš memoriji, a ako ne postoji, anulira se sadržaj keš memorije.

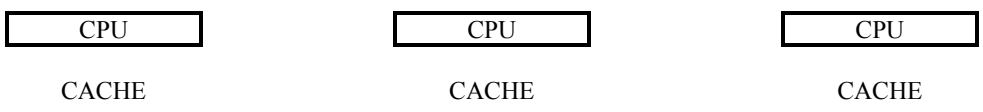


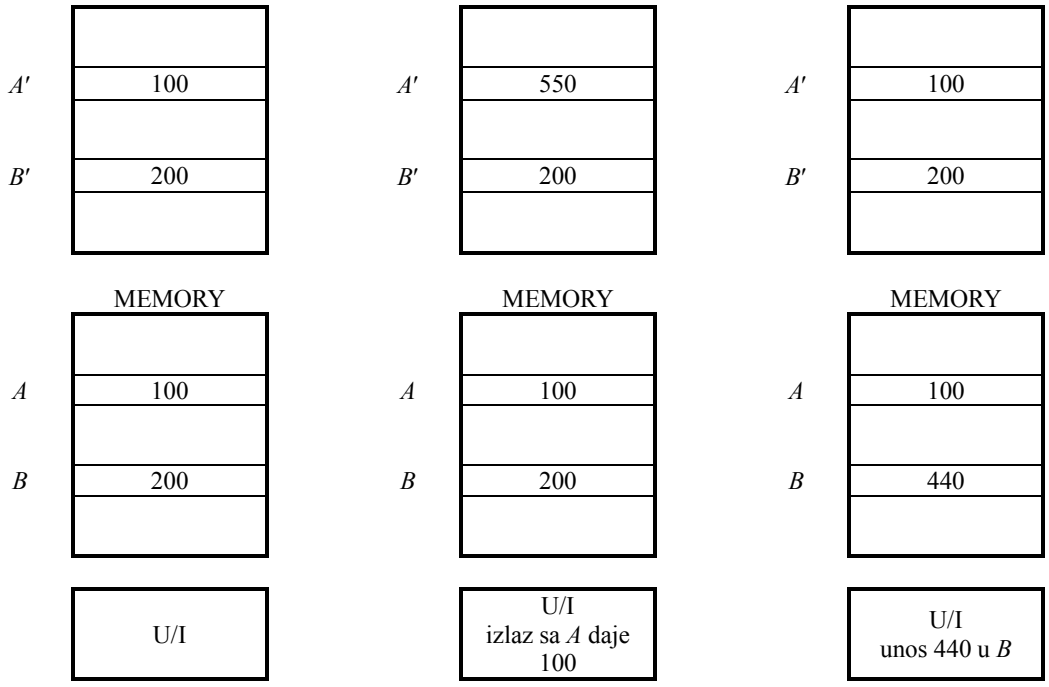
Slika 16 Istovremeno preslikavanje i pristup keš memoriji

Još jedno moguće ubrzanje pristupa realnom kešu je da se u jedinici za ubrzavanje preslikavanja irtuelnih u realne adrese po prvom prevođenju u dva posebna registra upamte broj stranice i broj bloka. Kod svake sledeće virtuelne adrese ide se paralelno sa prevođenjem broja stranice u broj bloka i proverava da li je ovo pristup istoj stranici, upamćenoj u registru stranice, kao i kod prethodnog pristupa. Ukoliko se otkrije da jeste, ne čeka se da se prevođenje kompletira u jedinici za ubrzavanje preslikavanja, već se na osnovu broja bloka, upamćenog u registru bloka, kao i kod prethodnog pristupa formira realna adresa i pristupa realnoj keš memoriji. Ukoliko se otkrije da nije, čeka se da se prevođenje kompletira u jedinici za ubrzavanje preslikavanja, pa se paralelno pristupa realnoj keš memoriji i pamte broj stranice i broj bloka u odgovarajućim registrima.

5.4.2. Keš memorija i U/I uređaji

U procesorima sa keš memorijom podaci mogu da se nađu u memoriji i kešu. U konfiguracijama računara kod kojih je jedino procesor uređaj koji može da menja i čita podatke, a keš se nalazi između procesora i memorije nema opasnosti da procesor radi sa starom ili bajatom (stale) kopijom podatka. Međutim u konfiguracijama računara sa U/I procesorima i DMA kotrolerima koji paralelno sa procesorom pristupaju memoriji može se desiti da kopija nekog podatka u kešu sa kojom radi procesor ne bude konzistentna sa kopijom u memoriji sa kojom radi U/I uređaj. Ovaj problem koji se naziva problem **keš koherencije** je ilustrovan na slici 17. Na slici su sa *A'* i *A* označene kopije istog podatka u kešu i memoriji, respektivno. Isto to važi i za *B*' i *B*. Na slici a) su prikazani keš i memorija u koherentnom stanju. Na slici b) je dato stanje keš i operativne memorije pošto je procesor izvršio upis vrednosti 550 u *A*. Međutim zbog pretpostavke da se koristi algoritam ažuriranja "write-back," upis je izvršen samo u keš a ne i u memoriju. Zbog toga je u keš memoriji na adresi *A'* nova korektna vrednost 550, a u memoriji na adresi *A* stara ili bajata nekorektna vrednost 100. Ako sada neki izlazni uređaj koristi vrednost sa adrese *A* memorije dobiže staru ili bajatu vrednost. Na slici c) je dato stanje u kešu i memoriji pošto ulazni uređaj unese vrednost 440 u memoriju na adresi *B*. Zbog toga će na lokaciji *B*' keša da bude stara ili bajata kopija.





a) keš i memorija su koherentni
$A' = A$ i $B' = B$

b) keš i memorija nisu koherentni
$A' \neq A$ (A'bajata)

b) keš i memorija nisu koherentni
$B' \neq B$ (B'bajata)

Slika 17 Keš memorija i u/i uređaji

Da li će ovaj problem keš koherencije postojati ili ne zavisi od toga kako se ulaz/izlaz u računaru odvija. Moguće je da se on odvija ili između U/I uređaja i keš memorije (slika ******) ili između U/I uređaja i memorije (slika ******). Ukoliko se kod ulaza podaci stavljaju u keš, a kod izlaza uzimaju iz keša, tada i ulazno/izlazni uređaji i procesor rade sa istom kopijom podatka. U tom slučaju problem keš koherencije je rešen. Sa aspekta problema keš koherencije pristup na slici ****** ima prednost nad pristupom na slici ******. Međutim nedostatak pristupa na slici ****** u odnosu na pristup na slici ****** je da on utiče na performanse procesora. Sada se U/I uređaji zajedno sa procesorom bore za pristup kešu, pa se može dešavati da će procesor morati da sačeka dok U/I uređaji pristupaju kešu. Pored toga ulaz može izazvati izbacivanje nekih blokova potrebnih procesoru iz keša radi ubacivanja ulaznih podataka kojima procesor neće skoro pristupati. Ako se uzme u obzir činjenica da se sada keševi stavljaju u isto integrisano kolo sa procesorom, onda postaje još teže povezivanje U/I uređaje preko keša.

Ukoliko se kod ulaza podaci stavljaju u memoriju, a kod izlaza uzimaju iz memorije (slika ******), tada nema gore navedenih problema ali se mora sprečavati problem bajatih podataka.

Ako se koristi **write-through** keš onda se prilikom upisa od strane procesora ažurira i operativna memorija. Stoga su sadržaji u kešu i memoriji identični, pa prilikom slanja podataka iz memorije u izlazni uređaj ne postoji problem sa bajatim podacima. Na primer na slici ****** b) i u *A* a ne samo u *A'* bi bila vrednost 550, pa bi se posle u izlazni uređaj slala korektna ažurna vrednost 550, a ne bajata 100. Zbog toga se i pored određenih nedostataka dosta često koristi **wrtie-through** tehnika ažuriranja sadržaja operartivne memorije.

Ako se koristi **write-back** algoritam situacija je nešto složenija i rešava se ili softverski (operativni sistem) ili hardverski. **Softversko rešenje** zahteva da operativni sistem pre nego što startuje izlaz na izlaznu jedinicu prebaci iz keša u memoriju sve one blokove koji su u

kešu modifikovani a nalaze se u opsegu adresa bafera memorije za izlaz. Time se obezbeđuje da se u izlaznu jedinicu je šalju bajati podaci. Ova operacija se naziva ispiranje keša (*flush*) i oduzima određeno vreme pošto, čak i da podaci nisu u kešu, mora se za datu adresu izvršiti sekvencijlna provera po svim ulazima keša. **Hardversko rešenje** zahteva da poseban hardver keš memorije pri izbacivanju adrese memorije od strane DMA radi čitanja, proveri "tag"-ove keša. Ukoliko otkrije da postoji saglasnost i da je blok u kešu modifikovan, hardver keša:

- generiše signal koji sprečava da memorija reaguje na upravljački signal čitanja
- šalje dati sadržaj iz keša u izlaznu jedinicu.

Ovo hardversko rešenje bi moglo da uspori pristup kešu od strane procesora, jer "tag"-ovima sada pristupa i procesor pri svakom obraćanju kešu i dodatni hardver pri operaciji čitanja iz memorije. Da bi se izbeglo usporavanje pristupa kešu, kod ovakvih hardverskih rešenja udvajaju se "tag"-ovi, pa jednom setu "tag"-ova pristupa procesor a drugom dodati hardver.

Kod ulaza sa ulaznih jedinica problem se rešava ili softverski ili hardverski na isti način kao i kod "write-back" keša i izlaza. **Softversko rešenje** zahteva da operativni sistem, po kompletiranju ulaza sa ulazne jedinice, očisti keš od svih onih blokova koji se nalaze u opsegu adresa bafera memorije za ulaz. Ova operacija se naziva ispiranje keša (flush) i iz istih razloga kao i kod izlaza oduzima određeno vreme. **Hardversko rešenje** zahteva da poseban hardver keš memorije pri izbacivanju adrese memorije od strane DMA radi upisa proveri "tag"-ove keša. Ukoliko se otkrije da postoji sagasnost ulaz keša gde je to otkriveno proglašava se za nevažeći. I ovde se javlja problem usporavanja keša i njegovo rešavanje udvostručavanjem "tag"-ova.

Realizacija hardverskih tehnika za održavanje keš koherencije zahteva realni, a ne virtuelni keš.

Problemi keš koherencije kod ulaza/izlaza su isti kao i problemi keš koherencije kod multiprocesora.

5.4.3. Keš memorija i glavna memorija

Glavna memorija je sledeći niži nivo u hijerarhiji memorija. Podaci sa ulaznih uređaja, najpre, idu u glavnu memoriju, a iz nje, zatim, u keš memorije procesora i obratno. Stoga glavna memorija mora da bude projektovana na takav način, da zadovolji s jedne strane zahteve keš memorije, a sa druge strane zahteve U/I uređaja. Tradicionalno, kašnjenje (*latency*) glavne memorije koje utiče na "miss penalty" keša je bio predmet interesovanja keša, a propusna moć (bandwidth) glavne memorije je bio predmet interesovanja U/I uređaja. Sa rastom popularnosti keš memorija drugog nivoa propusna moć (bandwidth) glavne memorije postaje predmet interesovanja i keša, jer ona omogućuje veću veličinu bloka keša drugog nivoa.

U ovom odeljku se razmatraju neki zahtevi o kojima se vodi računa pri projektovanju glavne memorije, sa aspekta njenog povezivanja sa kešom. U glavi ****** se to isto čini, ali sa aspekta njenog povezivanja sa U/I uređajima.

5.4.3.1. Tehnike za povećanje propusne moći glavne memorije

Tri najviše korišćene tehnike povećanja propusne moći glavne memorije su:

- veća širina memorijske reči,
- memorija sa preklopljenim pristupom modulima i

- memorija sa nezavisnim modulima.

5.4.3.1.1. Veća širina memorijske reči

Širina keš memorije se određuje tako da odgovara onoj širini reči sa kojom procesor najčešće radi. S druge strane prenos blokova između keš memorije i obratno bi se ubrzao ikoliko bi se povećala širina memorijske reči. Idealno bi bilo da ona odgovara veličini bloka. Zbog ove razlike u širinama reči keša i glavne memorije treba usvojiti jedno od sledeća dva rešenja. Prvo rešenje je da se usvoji da i širina keša odgovara širini glavne memorije. Tada se upis iz glavne memorije i keša i obratno realizuje sa širinim reči glavne memorije. Nezgoda je da kada se procesor obraća kešu mora da vrši multipleksiranje potrebne reči procesora iz široke reči. Drugo rešenje je da se kešu pristupa sa širinom reči procesora, a da se široka reč memorije dekomponuje na reči širine keša i takve reči jedna za drugom upisuju. Isto važi i za obrnuti smer prenosa.

U sistemima sa dva nivoa keša keš prvog nivoa ima reč širine reči procesora, a keš drugog nivoa širine glavne memorije. Multipleksiranje šire na užu reč se realizuje kod transfera između keša prvog i drugog nivoa.

Nedostatak velike širine memorijske reči je pre svega velika širina magistrale podataka.

U komercijalnim sistemima ima rešenja sa širinom reči keša drugog nivoa, magistrale podataka i širinom reči glavne memorije od 256 bita (32×8 bita = 32 bajta).

5.4.3.1.2. Memorija sa preklopljenim pristupom modulima

Ovde su širine reči procesora, keša i glavne memorije isti. Veća propusna moć se postiže, tako što se pristupi za različite module realizuju paralelno. Ovde se pretpostavlja da su susedne adrese u susednim modulima.

5.4.3.1.3. Memorija sa nezavisnim modulima

Ovde je ideja ista kao i kod prethodnog rešenja. I ovde su širine reči procesora, keša i glavne memorije isti. Razlika je u tome da svaki memorijski modul ima posebne adresne linije i linije podataka.

5.5. PERFORMANSE KEŠ MEMORIJE

Moguća mera performanse keš memorije je *miss rate* koja predstavlja je vrednost procentualnih promašaja. Ona je privlačna jer ne zavisi od brzine hardvera i obično je manja od 5%. Kao moguća mera performanse keš memorije je i prosečno vreme pristupa memorije (*average memory access time*). Ono zavisi od:

- vremena pristupa keš memoriji pri "hit"-u (*hit time*),
- procenta promašaja (*miss rate*) i
- prosečnog gubitka vremena pri promašaju (*miss penalty*)

i dato je formulom:

$$average\ memory\ access\ time = hit\ time + miss\ rate * miss\ penalty \quad (1)$$

Ona odražava brzinu hardvera i po njoj je poboljšanje performansi keš memorije moguće ostvariti smanjivanjem sva tri elementa. Tehnike kojima je to moguće realizovati su predmet daljih razmatranja.

5.5.1. Smanjenje *hit time*-a

Hit-time je veoma važan element kod projektovanja procesora jer ne utiče samo na prosečno vreme pristupa memorije već i na periodu signala takta procesora. Ovo je posledica činjenice da se u toku izvršavanja jedne instrukcije procesor više puta obraća memoriji i da ta obraćanja memoriji čine značajan deo ukupnog vremena izvršavanja instrukcije. Stoga se perioda signala takta tako bira da odgovara hit-time-u ili da neki njen umnožak bude hit-time. Ukoliko je *hit-time* manji moguće je periodu signala takta finije podešavati. *Hit-time* je moguće smanjiti korišćenjem male i jednostavne keš memorije i izbegavanjem prevođenja virtuelne u realnu adresu.

5.5.1.1. Mala i jednostavna keš memorija

Na *hit time* utiče vreme potrebno da se deo generisane adrese uzme i na osnovu njega izvrši provera **tag**-ova u keš memoriji i očita podatak. To je najbrže kod keš memorije sa direktnim, nešto sporije kod keš memorije sa set-asocijativnim, a najsporije kod keš memorije sa asocijativnim preslikavanjem.

5.5.1.2. Izbegavanje prevođenja virtuelnih u realne adrese

Čuvati virtuelne adrese.

To stvara problem.

5.5.2. Smanjenje *miss rate*

Miss-evi nastaju iz tri razloga.

- **Obavezni**—prvi pristup bloku uvek rezultira promašajem. Ovo se naziva **cold start miss** ili **first reference miss**.
- **Kapacitet**—keš memorija ima ograničen kapacitet i ne može da drži sve blokove koji su potrebni za vreme izvršavanja programa. **Promašaji zbog kapaciteta** nastaju zbog toga što se blokovi vraćaju iz keš u operativnu memoriju i posle ponovo dovlače iz operativne u keš memoriju.
- **Konflikti**—oni nastaju samo kod direktnog i set-asocijativnog preslikavanja. **Promašaji zbog konflikta** će se dešavati zato što neki blok može da bude vraćen iz keš u operativnu memoriju i kasnije ponovo dovučen u keš memoriju, ako se suviše mnogo blokova operativne memorije (koji pripadaju različitim grupama) preslikava na isti blok keš memorije (direktno preslikavanje) ili isti set keš memorije (set-asocijativno preslikavanje).

Kako poboljšati *miss rate*

- **Obavezni**—*moglo bi da se poboljša povećanjem veličine bloka*. Međutim veći blokovi povećavaju druge tipove *miss*-eva (na primer promašaje zbog kapaciteta i zbog konflikata).
- **Kapacitet**—jedino rešenje je ovde povećanje kapaciteta.
- **Konflikt**—*sa povećanjem asocijativnosti smanjuju se ovi konflikti* (direktna, set-asocijativna sa dva, četiri, itd. bloka po setu i na kraju potpuno asocijativna). Međutim, rešenja sa povećanom asocijativnošću su hardverski skuplja, a povećava se i *hit time*. (Analizirati kako se dolazi do podatka iz keša pri **hit**-u.)
  - **Direktno** (čita se **tag** i **data** i čim **tag** da **hit** odmah se koristi podatak).
  - **Set-asocijativno** (čita se **tag** i **data**, formiraju **hit**-ovi za svaki ulaz seta, pa selekcija podatka i formiranje ILI funkcije **hit**-ova, pa tek tada podatak).
  - **Asocijativno** (formiranje match bitova (komparator), broja ulaza (koder), pa tek tada čitanje podatka).

Opšti zaključak je da mnoge tehnike koje smanjuju *miss rate* povećavaju *hit time* i/ili *miss penalty*. Zbog toga pri odlučivanju i primeni neke tehnike za smanjenje *miss rate*-a mora se voditi računa o efektima drugih elemenata koji, takođe, utiču na *prosečno vreme pristupa*.

5.5.2.1. Veća veličina bloka

Veća veličina bloka smanjuje **obavezne promašaje**. Ovde se iskorišćava prostorni *lokalitet*. Međutim, veći blokovi povećavaju *miss penalty* (treba više vremena za veći blok da se dovuče u keš i vrati iz keša). Pored toga veći blok može da poveća *conflict miss*-evi i *capacity miss*-evi.

Ovde mogu da se jave interesantni efekti sa povećanjem veličine bloka:

- može se čak povećati *miss rate* (zbog povećanja **conflict** i **capacity** misses),
- može se tako povećati *miss penalty* da ima većeg uticaja od smanjenja obaveznih (**cold**) promašaja.

5.5.2.2. Veća asocijativnost

Bolja je veća asocijativnost ali se povećava hit-time.

5.5.2.3. Victim-keš (keš žrtava)

Ova tehnika se koristi da smanji **conflict misses** (direktno i set-asocijativno). Kešu se dodaje jedan mali potpuno asocijativan keš (victim keš). On sadrži blokove koji su bili žrtve (victims) pri **miss**-u. Pri **miss**-u se najpre vrši provera u **victim**-kešu. Ako jeste, onda **keš blok** i **victim blok** menjaju mesto. Ako nije u **victim** kešu tek onda se ide u operativnu memoriju.

5.5.2.4. Pseudo asocijativni keš

Ovo je pokušaj da se dobije **miss rate** set-asocijativnih keš memorija i **hit-time** direktnog preslikavanja.

Koristi se keš memorija sa direktnim preslikavanjem. Provera da li ima **hit** i ako je hit pristup kao kod uobičajenog direktnog preslikavanja. Ako **nije hit** još jedan ulaz u keš memoriji se proverava da li ima saglasnosti. Najjednostavniji način je da se invertuje najstariji bit u adresi ulaza da bi se izvršila provera u drugom bloku ovog **pseudoseta**.

Pseudo asocijativna keš memorija ima sada **brzi** i **spori hit**.

Opasnost je da može da se desi da mnogi **fast hit times** postanu **slow hit times** u pseudo asocijativnoj keš memoriji.

5.5.2.5. Prefetching (unapred očitavanje bloka)

Ovde se čita blok iz operativne memorije, pre nego što je potreban i smešta u bafer. Pri **miss**-u dovlači se dati i prvi sledeći blok. Dati blok ide u keš memoriju a sledeći u bafer. Ukoliko se pri **miss**-u desi da je zahtevan blok u baferu on ide iz bafera u keš, a prvi sledeći se dovlači u bafer.

5.5.3. Smanjenje “miss penalty”

Smanjenje “cache misses” je tradicionalno bilo predmet istraživanja u oblasti keš memorija radi poboljšanja performansi keš memorija. Međutim formula ****** pokazuje da pri ovim razmatranjima treba voditi računa i o “miss penalty” i o “hit time.” Ovome ide u prilog i činjenica da je od 1980. na ovamo trend u tehnologiji takav da se brže povećava brzina

procesora nego RAM memorija (Hennessy, fig. 5.1). Umajući to u vidu *miss penalty* sve značajnije utiče na performanse keš memorija.

U ovom odeljku se razmatraju neke od tehnika čijom primenom se može smanjiti *miss penalty* i time poboljšati performansa keš memorije.

5.5.3.1. Davanje prioriteta operaciji čitanja nad operacijom upisa

Kod keš memorija koje koriste "write-through" tehniku ažuriranja sadržaja operativne memrije kao po pravliku koriste se baferi za upis. Kod takvih keš memorija procesor čeka samo da se izvrši upis u bafer za upis, a ne i u samu operativnu memoriju. Time se preklapa aktivnost keš memorije upis iz bafera za upis u operativnu memoriju, sa drugim aktivnostima procesora. Ovde treba voditi računa o dimenzionisanju bafera za upis da bi u slučaju sukcesivnih operacija upisa svi ti zahtevi mogli da budu smešteni u bafer za upis. Međutim, ovo može da komplikuje stvari u situaciji kada zbog promašaja pri čitanju (read miss) treba iz operativne u keš memoriju dovući blok, koji do tog trenutka nije ažuriran jer se vrednost za upis još uvek nalazi u baferu za upis.

Ova situacija može nastati pri sledećoj sekvenci instrukcija:

```
SW   512(R0), R3        ; M[512] ← R3
LW   R1,      1024(R0)  ;     R1 ← M[1024]
LW   R2,      512(R0)   ;     R2 ← M[512]
```

ukoliko se koristi keš memorija sa direktnim preslikavanjem sa "write-through" tehnikom ažuriranja sadržaja operativne memorije, baferom za upis i takvim kapacitetom keš memorije da se memorijske lokacije 512 i 1024 preslikavaju na isti nulti ulaz (blok) keš memorije. Uzeti da je pri upisu sadržaja registra R3 u lokaciju 512 instrukcijom SW bita saglasnost, pa se upis vrši u ulaz nula keš memorije i samo u bafer upisa i to u poslednji, četvrti ulaz. Uzeti da je potom pri izvršenju prve instrukcije LW kojom se sadržaj lokacije 1024 upisuje u registar R1 nije otkrivena saglasnost u nultom ulazu keš memorije pa se sada blok operativne memorije sa lokacijom 1024 dovlači u nulti ulaz keš memorije. Pri izvršenju druge instrukcije LW kojom se sadržaj lokacije 512 upisuje u registar R2 neće biti saglasnosti u nultom ulazu keš memorije, pa se sada blok operativne memorije sa lokacijom 512 dovlači u nulti ulaz keš memorije.

Ako iz bafera za upis nije izvršen upis u memorijsku lokaciju 512 iniciran instrukcijom SW, onda će se u ulaz nula keš memorije kao rezultat druge instrukcije LW dovući blok koji sadrži staru, a ne ažurnu vrednost lokacije 512, pa će time i sadežaj registra R2 biti pogrešan i različit od R3.

Najjednostavniji način da se ovaj problem izbegne je da se pri promašaju kod čitanja (read miss) ne kreće sa dovlačenjem bloka iz operativne u keš memoriju sve dok se bafer za upise ne isprazni. Međutim, time se povećava "miss penalty" jer je vrlo izvesno da će pri promašaju kod čitanja (read miss) u baferu za upis biti neki sadržaj koji čeka na upis. Rešenje ovoga je da se pri promašaju kod čitanja proverava da li se u baferu za upis nalazi sadržaj koji treba upisati u blok koji treba dovući. Ako se otkrije da jeste, čeka se da se izvrši upis, pa se tek onda dovlači. Ako se otkrije da nije, odmah se kreće sa dovlačenjem bloka, pa se tek posle toga produžava sa upisom iz bafera za upis u operativnu memoriju.

Miss penalty se može smanjiti i u slučaju keš memorija koje koriste "write-back" algoritam ažuriranja sadržaja operativne memorije. Pri promašaju se, najpre, u operativnu memoriju vraća blok samo ako je "dirty." Međutim, umesto da se "dirty" blok najpre vrati u operativnu memoriju, pa tek onda dovuče novi blok iz operativne memorije, moguće je "dirty" blok prebaciti u bafer, zatim dovući potreban blok i tek onda prebaciti "dirty" blok iz bafera u

operativnu memoriju. Time će se operacija čitanja (ili upisa) zbog koje procesor čeka brže obaviti.

Slično kao i kod write-through algoritma ažuriranja sadržaja operativne memorije, ako se desi promašaj pri čitanju (ili upisu) pa treba dovući novi blok, moguć je konflikt ako u baferu ima sadržaj koji treba upisati u operativnu memroiju. Rešenje je i ovde slično rešenju iz prethodnog slučaja. Moguće je zaustavljanje dovlačenja bloka dok se ne isprazni bafer. Ovo je najjednostavnije rešenje ali je "miss penalty" značajan. Poboljšanje bi bilo da se izvrši provera da li blok koji treba da se dovuče čeka ažuriranje, jer se nalazi u nekom od bafera za upis. Ukoliko je to slučaj čeka se da se izvrši upis, pa se tek tada dovlači. U suprotnom odmah se dovlači, pa se tek posle toga produžava sa upisom u operativnu memoriju iz bafera za upis.

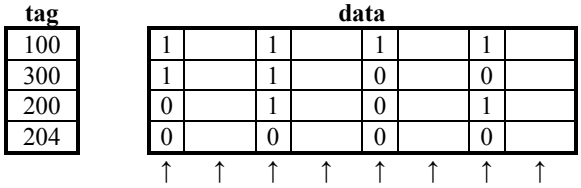
5.5.3.2. Keš memorija sa podblokovima

U nekim situacijama postoji potreba da se tako isprojektuje keš memorija da je moguće **data** deo određenog kapaciteta staviti na čip procesora. Može se, takođe, desiti da se, uzimajući u obzir prethodna razmatranja, odabere takva veličina bloka, da za nju potreban kapacitet **tag** memorije ne može da stane na čip. Jedan pristup je da se ide sa tom veličinom bloka ali da se smanji kapacitet **data** dela na kapacitet koji omogućava raspoloživi prostor na čipu za **tag** deo. Ovo sigurno povlači povećanje **miss rate**-a. Drugo rešenje je da se ide na veću veličinu bloka, i to do one koju dozvoljava raspoloživi prostor za **tag** deo, ne smanjujući totalni kapacitet **data** dela keš memorije. Ovim rešenjem će se u odnosu na prvo poboljšati "miss rate," ali će se povećati "miss penalty" zbog velikog bloka.

Rešenje koje se susreće u ovakvim situacijama je da se veliki blok čija je veličina diktirana dozvoljenim kapacitetom **tag** dela, podeli na više podblokova, čija veličina odgovara optimalnoj veličini bloka. U keš memoriji **tag** se dodeljuje celom bloku, ali se pri promašaju samo dovlači potreban podblok. Zbog toga za svaki blok mora da postoji onoliko **valid** bitova koliko ima podblokova. Svaki **valid** bit bloka označava da li se dati podblok nalazi u kešu ili ne. Stoga da bi se za neku reč utvrdilo da li postoji saglasnost potrebno je ne samo da se slažu **tag**-ovi, već i da **valid** bit podbloka date reči bude postavljen. "Miss penalty" memorije kod koje su blokovi podeljeni na podblokove je manji nego keš memorije kod koje se radi sa kompaktnim blokovima.

Primer keš memorije sa direktnim preslikavanjem na nivou bloka i četiri podbloka u bloku je dat na slici 18. U slučaju bloka sa **tag**-om 100 sva četiri podbloka su dovučena jer su **valid** bitovi za sva četiri podbloka jedinice. U slučaju bloka sa **tag**-om 300 samo su nulti i prvi podblok dovučeni pa su njihovi **valid** bitovi jedinice, dok drugi i treći podblok nisu dovučeni pa su njihovi **valid** bitovi nule. U slučaju bloka sa **tag**-om 200 dovučeni su podblokovi jedan i tri, a za blok sa **tag**-om 204 ni jedan pidblok nije dovučen. Vidi se sa slike da je u slučaju ove keš memorije sa četiri podbloka u bloku, potrebno čuvati četiri **tag**-a. U slučaju keš memorije koja bi imala isti kapacitet kao keš memorija sa podblokovima, bilo bi 16 blokova, pri čemu bi sada veličina bloka odgovarala veličini podbloka, i bio bi potreban **tag** deo za čuvanje 16 **tag**-ova.

Kod keš memorije sa podblokovima jedinica preslikavanja je blok, a jedinica transfera između operativne i **data** dela keš memorije je podblok.



V P V P V P V P

Slika 18 Keš memorije sa podblokovima

Legenda: V—**valid** bitovi podblokova; P—podblokovi.

5.5.3.3. Kritičnu reč prvo i rani start procesora

Dovlači se iz bloka prvo reč koja treba, pa tek onda ostatak.

Ta reč se odmah daje procesoru, iako da je još uvek u toku prenos preostalih reči bloka.

5.5.3.4. Neblokirajući keš

Ovo kod pipeline-a. Ako je **miss** po jednom zahtevu pa se dovlači blok, dozvoliti da iz drugog stepena može da stigne zahtev (nema blokiranja), pa ako je **hit** po njemu reagovati, iako po prethodnom **miss**-u još nije kompletirano.

5.5.3.5. Keš memorija drugog nivoa

Razmatrane tehnike za smanjenje "miss penalty" bile su usredsređene na interfejs između procesora i keš memorije. Ova tehnika se usredsređuje na interfejs između keš memorije i operativne memorije, radi smanjenja "miss penalty." Njom se rešava kontradiktorni zahtev kod projektovanja keš memorija i to:

- keš memorija treba da bude što je moguće **brža**, a to znači i manja, da bi se približila brzini procesora, i
- keš memorija treba da bude što je moguće **veća** da bi u njoj mogli da se nalaze skoro svi blokovi operativne memorije i da se time ponaša kao operativna memorija.

Kao moguće rešenje kojim se zadovoljavaju ovi kontradiktorni zahtevi je da se doda još jedna keš memorija između postojeće keš memorije i operativne memorije. Postojeća keš memorija, koja se obično naziva keš memorija prvog nivoa, se tada projektuje da bude što je moguće brža, a to znači i manja, da bi se približila brzini procesora. Novoubačena keš memorija, koja se obično naziva keš memorija drugog nivoa, se tada projektuje da bude što je moguće veća tako da u njoj keš memorija prvog nivoa nađe skoro sve blokove za koje bi inače trebalo ići u operativnu memoriju. Ovim se smanjuje "miss penalty" keš memorije prvog nivoa, jer se sada potreban blok dovlači iz keš memorije drugog nivoa, koja je brža od operativne memorije.

Konceptualno ovo je jednostavno rešenje, mada je analiza složena.

$$\textit{prosečno vreme pristupa} = \textit{hit time}_{L1} + \textit{miss rate}_{L1} * \textit{miss penalty}_{L1},$$

$$\textit{miss penalty}_{L1} = \textit{hit time}_{L2} + \textit{miss rate}_{L2} * \textit{miss penalty}_{L2}.$$

Vidi se da:

- brzina keš memorije prvog nivoa utiče na clock procesora, dok
- brzina keš memorije drugog nivoa utiče samo na "miss penalty" keš memorije prvog nivoa.

Nekoliko pitanja vezanih za keš memoriju drugog nivoa:

Veličina. Treba da bude veća, a i niža po ceni jer se realizuje sa memorijskim elementima koji su sporiji nego memorijski elementi kep memorije prvog nivoa, a brži nego operativna memorija. Praktično treba toliko da je velika da "miss"-eva po kapacitetu nema, već samo **obaveznih** (compulsory) i zbog **konflikata** (conflicts).

Asocijativnost. Mada se zbog većeg kapaciteta keš memorije drugog nivoa blokovi bolje distribuiraju, "miss"-evi zbog konflikata smanjuju, još uvek ih ima dosta kod direktnog preslikavanja, pa je bolje ići na veću asocijativnost.

Veličina bloka. Bolja je veća veličina bloka. Ona ovde ne utiče značajno na povećanje "miss" konflikata jer je dosta velika. Efekat je da su blokovi prvog nivoa manji (16 do 32 bajta), a drugog veći (i do 256 bajtova).

Obično se blokovi prvog nivoa nalaze u blokovima drugog nivoa:

- prvi 32 do 64 KB,
- drugi 256 do 512 KB.

5.6. OPERATIVNA MEMORIJA

5.6.1. Tehnologija keš i glavne memorije

Kašnjenje (latency) memorije se obično izražava sa dve veličine: vreme pristupa memorije (access time) i vreme ciklusa (cycle time) memorije. Vreme pristupa je vreme koje protekne od trenutka kada se startuje operacija čitanja do trenutka kada podaci postanu raspoloživi. Vreme ciklusa je minimum vremena koje mora da protekne između dva obraćanja memroiji. Kod nekih memorija vreme ciklusa je isto kao i vreme pristupa, a kod nekih vreme ciklusa je veće od vremena pristupa.

Memorije se realizuju sa RAM memorijama kojih ima dva tipa: dinamičke RAM memorije (DRAM—Dynamic RAM) i statičke RAM memorije (SRAM—Static RAM). Za realizaciju glavne memorije obično se koristi DRAM, a za realizaciju keš memorije SRAM. Neke karakteristike ovih memorija relevantne za njihovo korišćenje pro realizaciji glavne i keš memorije razmatraju se u daljem tekstu.

DRAM koristi jedan tranzistor za smeštanje jednog bita i zahteva povremeno osvežavanje (refreshing) da ne bi došlo do gubitka informacije. Interno je memorijski čip organizovan kao jedna pravougaona matrica, koja se sastoji od određenog broja vrsta i kolona. Struktura adrese je takva da pola adresnih bitova predstavlja adresu vrste, a pola adresu kolone. Sa porastom kapaciteta DRAM čipova, rastao je i broj potrebnih adresnih linija, a time i broj pinova na čipu. Veći broj pinova na čipu povećava cenu čipa. Zbog toga se obično vrši multipleksiranje adrese, što smanjuje na pola broj pinova adrese. Tada se najpre generisanjem signala RAS (row access strobe) šalje prva polovina adrese koja predstavlja broj vrste, a zatim, generisanjem signala CAS (column access strobe), šalje druga polovina adrese koja predstavlja broj kolone. Osvežavanje DRAM se može realizovati tako da se osvežavaju istovremeno svi bitovi u vrsti. Pošto je memorijska matrica pravougaonik ili kvadrat, a osvežavanje je na nivou vrsta, broj koraka za osvežavanje odgovara broju vrsta, a to je približno kvadratni koren kapaviteta DRAM. Ako se uzme da se kolona mora osvežavati na svakih 10 ms, da je kapacitet čipa 1 Mbit, što znači da ima 10^3 vrsta i 10^3 kolona, i da je vreme pristupa 100 ns, dobija se totalno vreme osvežavanja čipa 100 μs. To znači da grubo gledano vreme osvežavanja (100 μs) odnosi 1% ukkupnog vremena (10 ms).

Zbog periodičnih osvežavanja memorija realizovana sa DRAM-om povremeno nije raspoloživa. Uz sve ovo dinamička priroda kola u DRAM-u zahteva da se posle svakog čitanja pročitani podatak ponovo upiše. Kao posledica ovoga kod DRAM je vreme ciklusa duže od vremena pristupa. Ovome treba još dodati i vreme potrebno za osvežavanje.

SRAM koristi četiri do šest tranzistora za smeštanje jednog bita, čime se sprečava kvarenje sadržaja bita pri čitanju. Stoga i nema potrebe za novim upisom po čitanju, niti za osvežavanjem. Zbog toga je kod SRAM-a vreme ciklusa i vreme pristupa isto.

Kod DRAM-a cilj je što je moguće veći kapacitet pa tek onds brzina, dok je kod SRAM-a cilj što je moguće veća brzina pa tek onds kapacitet. Kao posledica ovakvih ciljeva adresne linije su multipleksirane kod DRAM, dok kod SRAM nema multipleksiranja adresnih linija. Za memorije projektovane u uporedivim tehnologijama, kapacitet DRAM-ova je 4 do 8 puta veći od kapaciteta SRAM-a. Vreme pristupa SRAM-ova je 8 do 16 puta kraće nego DRAM-ova, ali je i njihova cena 8 do 16 puta veća.

Za realizaciju glavne memorije obično se koristi DRAM, a za realizaciju keš memorije SRAM.

Jedno od pravila kojeg se pridržavaju projektanti računara je da kapacitet glavne memorije treba da se povećava linearno sa povećanjem brzine procesora, da bi se procesor i memorija ponašali kao dobro izbalansiran sistem. Takve zahteve može da ispuni, kada je reč o ceni, jedino DRAM. Nažalost, kapaciteti DRAM-a rastu sporije nego što se povećavava brzina procesora. Ovaj problem se ublažava korišćenjem keš memorije između procesora i glavne memorije. Međutim, povećavanje kapaciteta keša ili uvođenje više nivoa keševa je sa aspekta cene koštanja prihvatljivo samo do određene granice. Zbog toga se koriste određene tehnike organizacije glavne memorije kojima se kroz povećanje propusne moći (bandwidth) glavne memorije poboljšavaju performanse keš memorije i time ptoblem dispariteta u rastu brzine procesora i kapaciteta DRAM-a ublažava. Te tehnike su predmet daljih razmatranja.

5.6.2.

5.6.3.