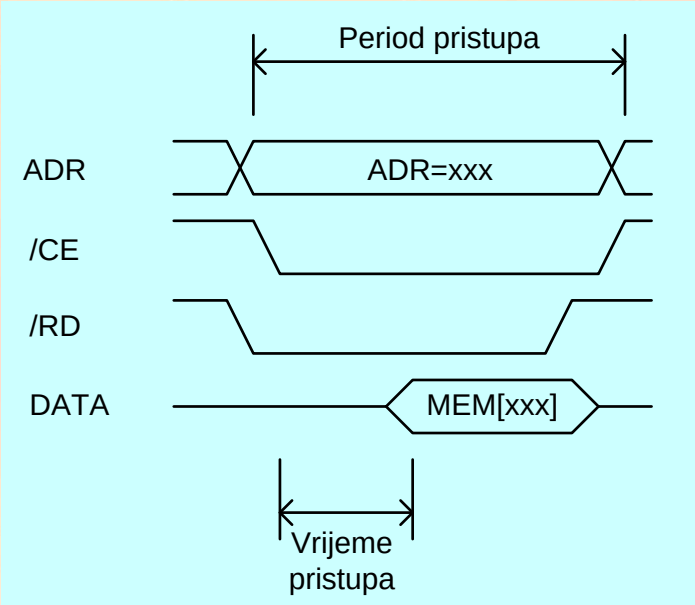
Memorijski sustavi

Definicije i osnovni vremenski dijagram

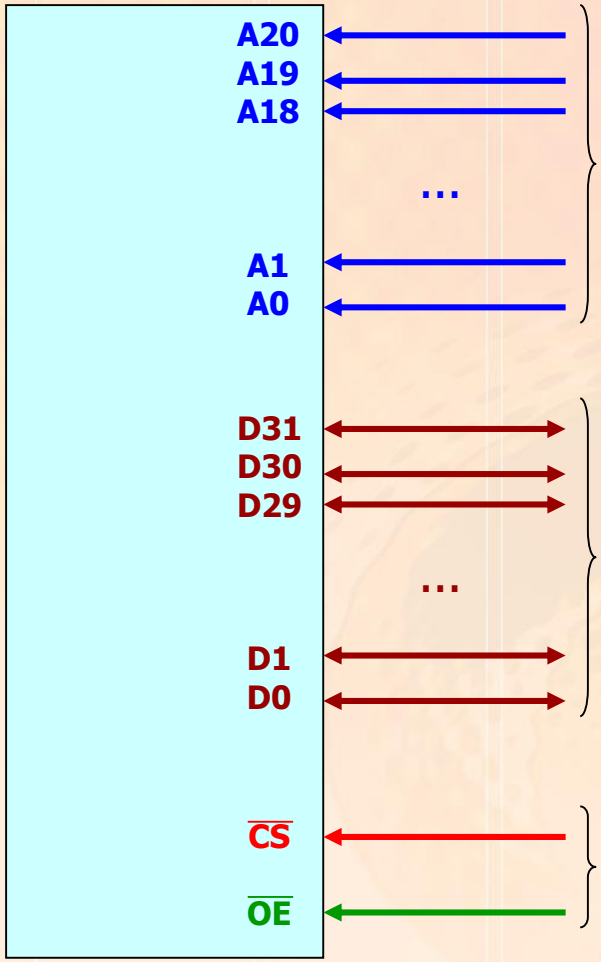
- **Memorijski pristup** (memory access):
  - Operacija čitanja ili pisanja podatka u memoriju
- **Vrijeme pristupa** (access time):
  - Vrijeme potrebno da memorija obavi čitanje ili pisanje podatka nakon što su na ulazu postavljeni svi potrebni adresni i upravljački signali
- **Period pristupa** (access period):
  - Period između iniciranja zahtjeva za pristupom memoriji i trenutka kad je pristup obavljen do kraja te kad je memorija spremna za novi zahtjev



- **Brzina**
  - 1/period pristupa
- **Propusnost** (throughput)
  - Umnožak brzine memorije i širine podataka koje memorija daje na priključcima. Izražava se u b/s

Memorijski signali

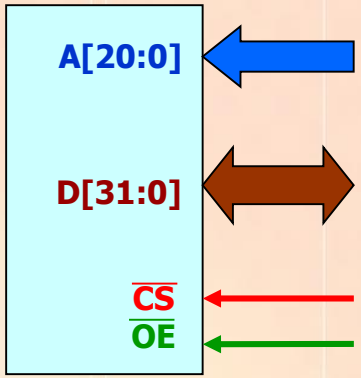
Primjer signala na običnoj statičkoj memoriji (SRAM) od 64Mb (2Mx32b) tj. (2^{21}x32b)



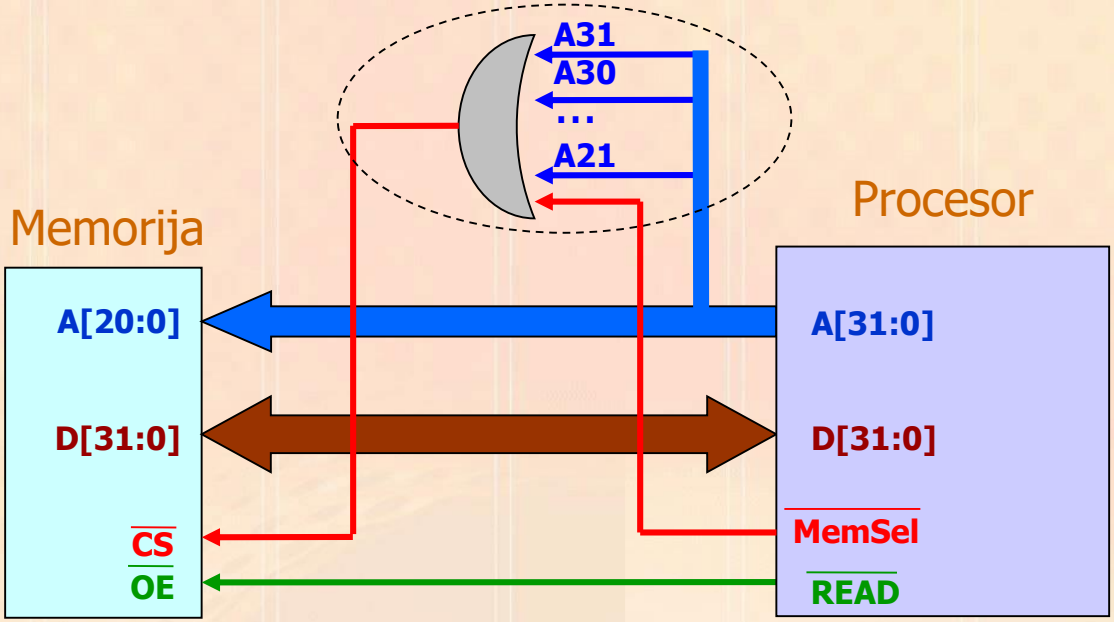
Adresna sabirnica

Podatkovna sabirnica

Upravljačka sabirnica



Povezivanje memorije i procesora



/CS (Chip Select) ili /**CE** (Chip Enable) je signal koji omogućuje dekodiranje adresne sabirnice i ostalih upravljačkih signala unutar memorije

/OE (Output Enable) je signal koji podatke iz internog spremnika podataka u memoriji prosljeđuje na vanjsku sabirnicu podataka (obično se spaja na procesorov priključak /RD)

/WE (Write Enable) je signal koji inicira pisanje podataka sa sabirnice podataka na izabranu lokaciju u memoriji (obično se spaja na procesorov priključak /WR)

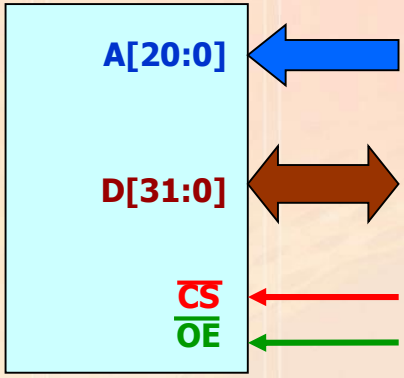
/MemSel (Memory Select) je signal kojim procesor definira da je na adresnoj sabirnici stabilna memorijska adresa (postoje različite izvedbe ovog signala)

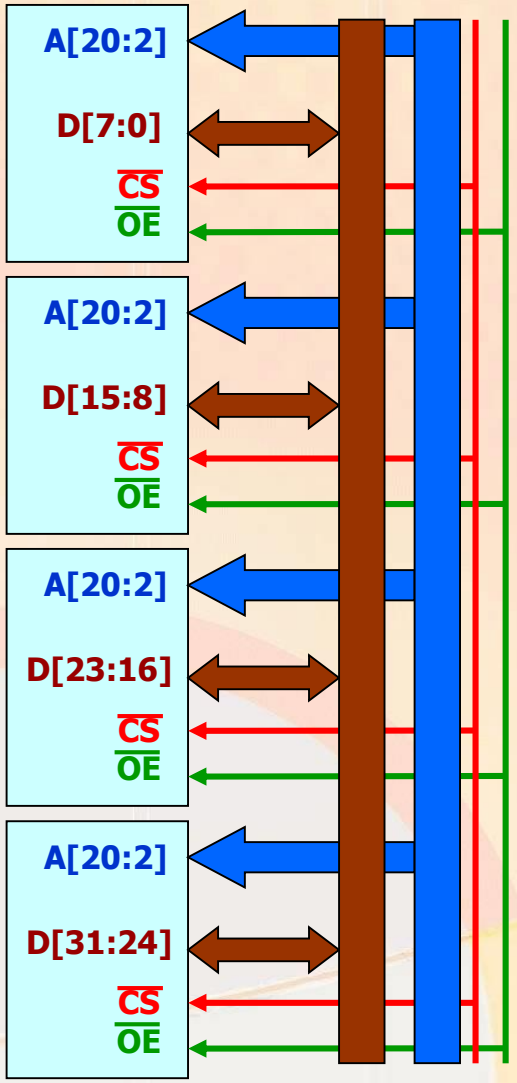
Načini povezivanja u stvarnosti

- Na dosadašnjim slikama memorija je prikazivana kao jedan čip s potrebnim (tj. velikim) brojem adresnih i podatkovnih priključaka
- U stvarnosti se ne proizvode memorijski čipovi s velikim brojem adresnih priključaka i podatkovnih priključaka
- Cijena memorije s manje priključaka je povoljnija
- Memorija se formira kao blok memorijskih čipova manjeg kapaciteta i manje širine riječi*

* manji kapacitet znači manju širinu adresnih priključaka, a manja širina riječi znači manju širinu podatkovnih priključaka

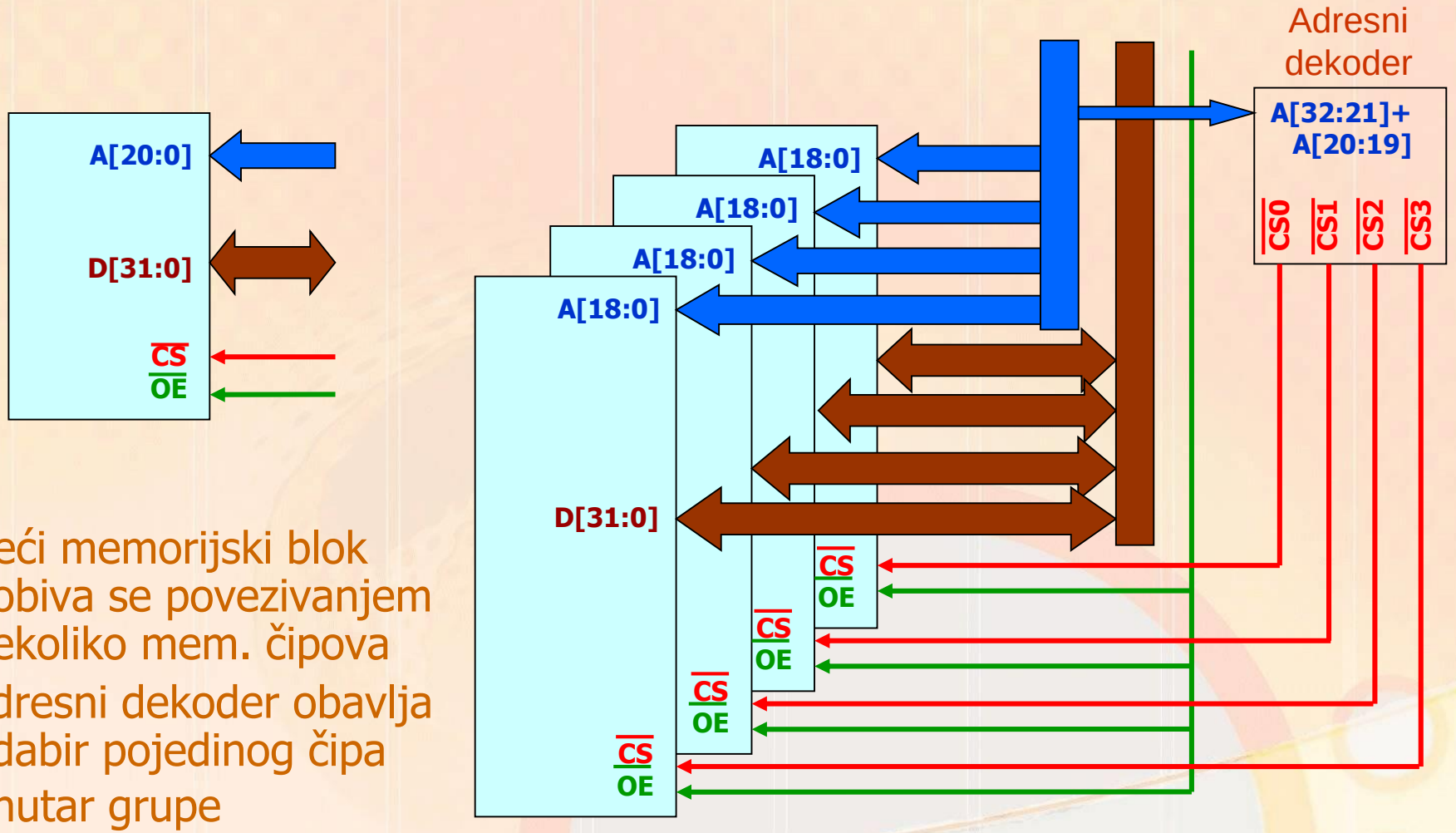
Memorijski blok s čipovima manje širine riječi





- Najniža dva bita adrese često se koriste za izbor bajta unutar riječi ako procesor želi dohvatiti podatak manje preciznosti

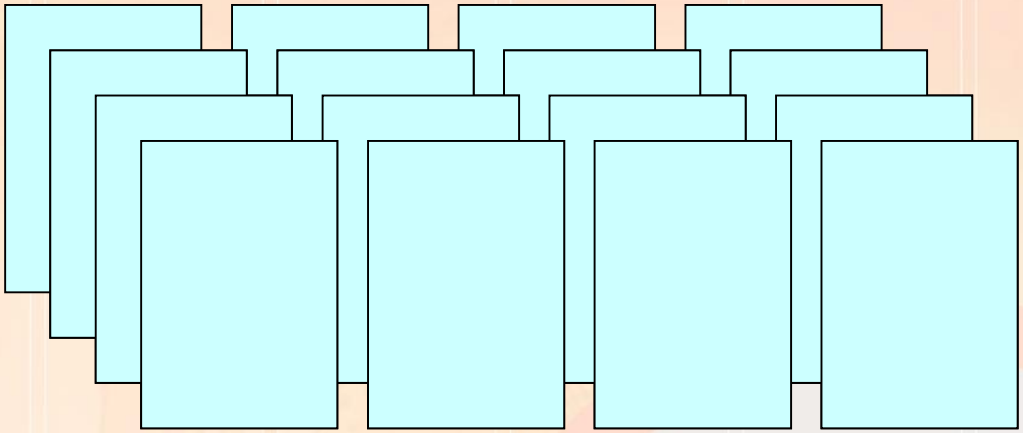
Memorijski blok s čipovima manjeg kapaciteta



- Veći memorijski blok dobiva se povezivanjem nekoliko mem. čipova
- adresni dekoder obavlja odabir pojedinog čipa unutar grupe

Memorijski blok većeg kapaciteta

- Kombinacijom prethodna dva načina povezivanja ostvaruju se memorijski blokovi (memorijski moduli) većeg kapaciteta



Primjeri stvarnih memorija i njihovih cijena

- Ukupna cijena memorijskog modula (načelno):
  - Što je kapacitet memorijskog čipa veći, to mu je cijena po Mb veća (pinovi, kućište, površina silicija, itd.)
  - Cijena memorijskog modula (pločica, montiranje, prostor, itd.) raste s porastom broja memorijskih čipova
  - UKUPNA CIJENA: **mora se tražiti optimum zbroja gornje dvije funkcije**

Ubrzanje memorijskog sustava

Zahtjevi na memoriju

- Pored zahtjeva za što većim kapacitetom, u primjeni je isto tako važan i zahtjev za željenom brzinom memorije
- Brzina memorije značajno utječe na performanse sustava
- Pri razvoju programa uvijek postoji želja ili potreba za velikom količinom **brze** memorije
- Brza memorija je **skupa**, a u nekom trenutku koristi se samo mali dio ukupne memorije u sustavu te je teško opravdati stavljanje velike količine brze memorije u računalni sustav

Ubrzanje memorijskog sustava

- Zbog cijene se u stvarnim računalnim sustavima primjenjuju određene metode organizacije memorijskog sustava kojima se poboljšavaju performanse memorijskog sustava (postiže se ubrzanje)
- Rezultat tih metoda je da se korištenjem jeftinije tehnologije postižu performanse memorijskog sustava koje su bliske izvedbi s brzim, ali i znatno skupljim memorijama
- Ovakve metode organizacije memorije **stvaraju dojam** velike količine brze memorije
- U nastavku ćemo objasniti neke osnovne organizacije memorijskog sustava kojima se poboljšavaju performanse

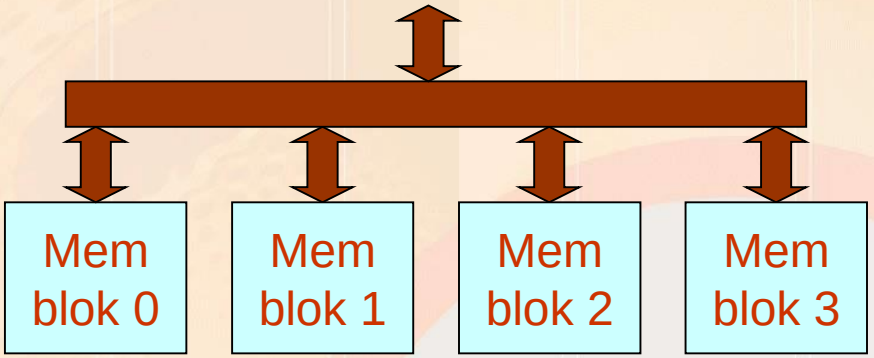
Ubrzanje memorijskog sustava

Preplitanje

Preplitanje - načelo rada

- Povećanje propusnosti mem. sustava može se postići **preplitanjem** (engl. interleaving) memorija

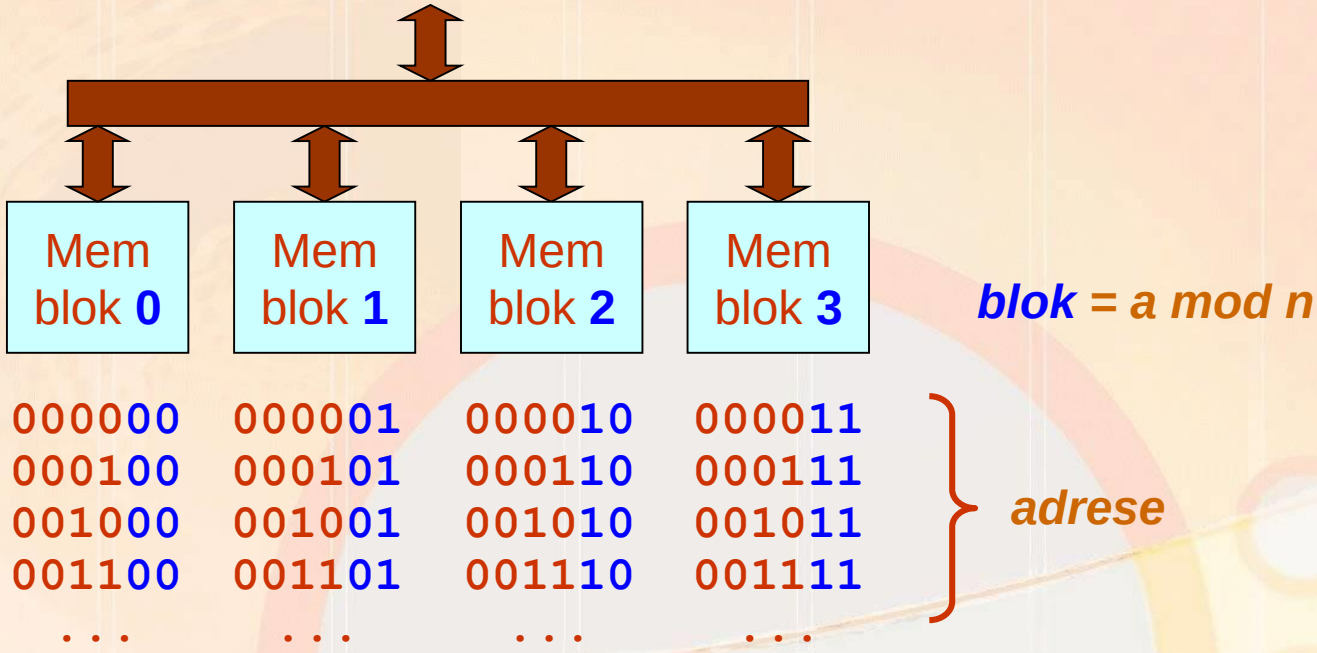
- Adresni prostor dijeli se u ***n*** memorijskih blokova (engl. memory bank) tako da memorijska lokacija s adresom ***a*** bude smještena u memorijski blok (***a mod n***). Operacija ***mod*** jednostavno se izvodi dekodiranjem zadnjih bitova adrese



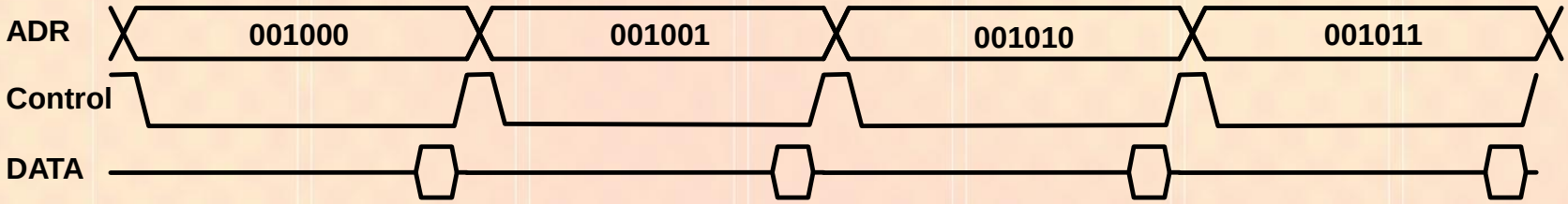
- Takva memorija sa ***n*** blokova se naziva **memorija s n-terostrukim preplitanjem** (engl. n-way interleaved)

Preplitanje - načelo rada

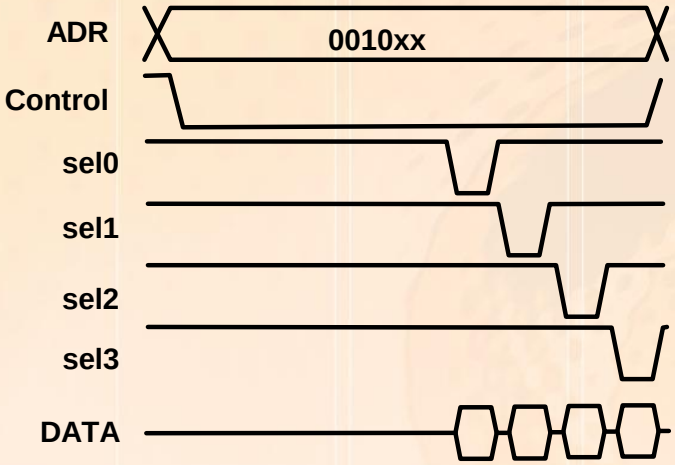
- Na slici su prikazane adrese u pojedinim blokovima (binarno) za **n=4**
- Na taj način je osigurano da se susjedne adrese ne nalaze u istom bloku te se mogu paralelno dohvaćati
- Sustav radi na načelu da se uklanja čekanje na dekodiranje adrese (engl. latency) tako da se dekodiranje obavlja jednom za četiri uzastopne adrese



Primjer uštede za čitanje 4 slijedna podatka



Trajanje običnog pristupa za 4 podatka (uz vrijeme pristupa = 12 clk):
4*(1clk (slanje adrese) +12clk (pristup) +1clk (učitavanje podatka))=56clk



Trajanje pristupa za organizaciju spreplitanjem:
(1+12+1)+ (0+0+1)+ (0+0+1) + (0+0+1) = 17clk

Ubrzanje = 3.3 x

Signali sel0 do sel3 izvode se iz adresnih signala a0 i a1 (običnim dekodiranjem)

Preplitanje

- Vrlo dobro rješenje za sustave s čestim pristupom slijednim lokacijama (npr. programski kod, pristup poljima podataka, ...)
- Efektivno vrijeme pristupa, a time i propusnost, povećava se približno za ***n***, gdje je ***n*** broj memorijskih **blokova**
- Sustav izvrsno radi i za čitanje i pisanje !!!

Ubrzanje memorijskog sustava

Hijerarhijska organizacija memorije

Hijerarhijska organizacija memorije

- Primjer problema: učenje za ispit iz ARH1 1 u knjižnici
  - **Način 1**:
    - Sve knjige su na policama knjižnice
    - Svaki put kad trebamo neki podatak iz knjige, uzmemo knjigu, pročitamo podatak i knjigu vratimo na policu
    - **SPORO !!!**
  - **Način 2**:
    - Kad zatrebamo neku knjigu, uzmemo je s police i stavimo na stol
    - Ako na stolu više nema mjesta, onda za svaku novu knjigu prvo moramo jednu knjigu sa stola spremiti na policu, a tek onda možemo uzeti novu knjigu za čitanje
    - S obzirom da za proučavanje neke teme ne trebamo sve knjige, već da većinu vremena provedemo čitajući neki manji skup knjiga, nećemo trebati ustajati od stola tako često
    - **BRŽE !!!**

Hijerarhijska organizacija memorije

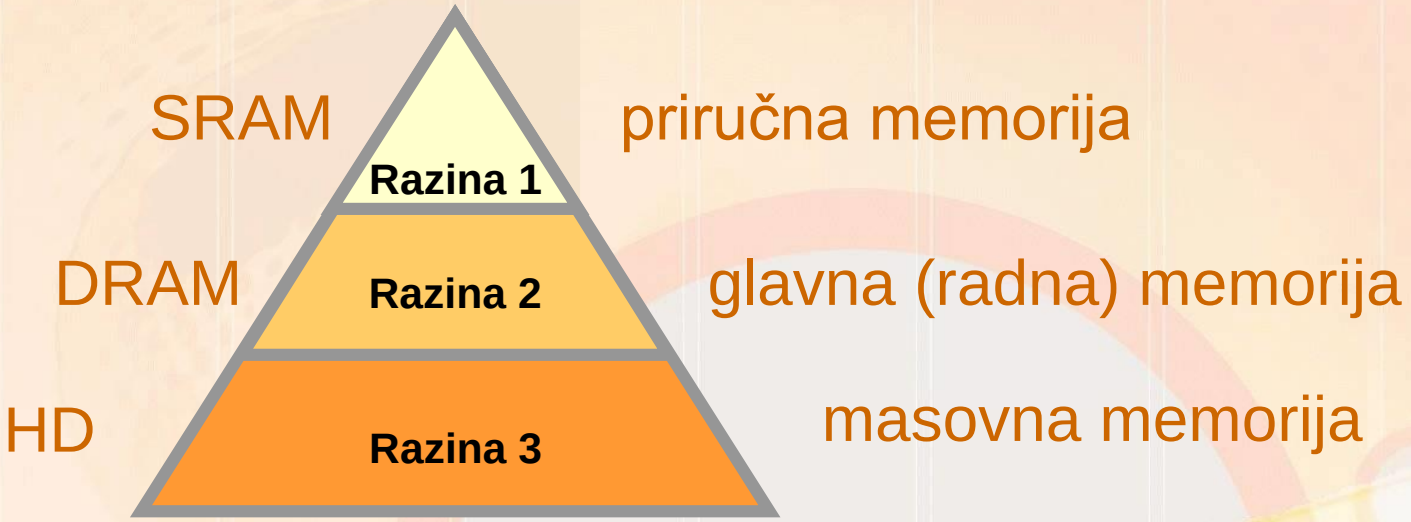
- Prethodni primjer opisuje metodu poboljšanja performansi korištenjem hijerarhijske organizacije
- Često korištena metoda za poboljšanje performansi memorijskog sustava upravo je hijerarhijska organizacija

- Načelo hijerarhijske organizacije memorije određuje da se podaci koji se češće koriste smještaju u manju, ali brzu memoriju, dok se ostali, rjeđe korišteni podaci čuvaju u većoj i sporijoj memoriji

- Kako se može znati koji će se podaci češće koristiti? Podaci u stvarnim primjenama imaju karakteristiku nazvanu **lokalnost podataka**:
  - **Prostorna lokalnost**: ako pretpostavimo da je neki podatak bio potreban, onda je vjerojatno da će i njemu **susjedni podaci uskoro** biti potrebni (npr. naredbe u nekom programu, elementi polja, ...)
  - **Vremenska lokalnost**: ako pretpostavimo da je neki podatak bio potreban, onda je vjerojatno da će **isti podatak uskoro** biti opet potreban (npr. varijable u nekom programu, ...)

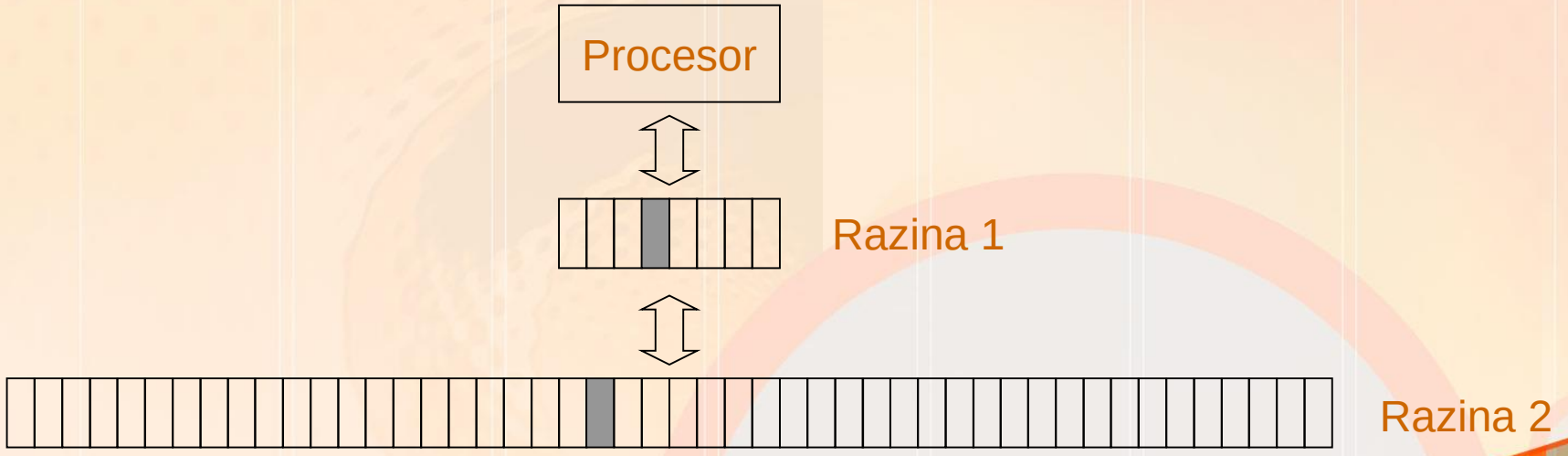
Hijerarhijska organizacija memorije

- **Glavna ili radna memorija** (druga razina) ostvarena je korištenjem DRAM-a, koji predstavlja kompromis između kapaciteta, brzine i cijene
- Između glavne memorije i procesora, na prvu razinu, može se staviti manja količina brze statičke memorije SRAM. Takva memorija naziva se **priručna memorija** (engl. cache)
- Na kraju, u sustav se na treću razinu obično stavlja magnetska memorija u obliku diskova. Ona ima malu cijenu i velik kapacitet, ali je spora



Kako radi sustav s priručnom memorijom

- U okviru Arhitekture računala 1 objasnit ćemo samo neka osnovna načela (stvarne izvedbe mogu biti složenije, koristiti brojne podvarijante i poboljšanja itd.)
- Memorija se dijeli u **blokove**, minimalne količine podataka koji se mogu naći u susjednim razinama



Kako radi sustav s priručnom memorijom

- Ako procesor želi pristupiti nekom podatku, tada se **pretražuje** memorija koja je najbliža procesoru, a to je **priručna memorija**
  - Ako se podatak **nalazi** u priručnoj memoriji, tada se to naziva **pogodak** (engl. hit)
  - Ako se podatak **ne nalazi** u priručnoj memoriji, tada se to naziva **promašaj** (engl. miss). Tada se pristupa glavnoj memoriji kako bi se dohvatio podatak

- **Omjer pogodaka** (engl. hit-rate): broj pristupa memoriji kada je podatak pronađen u višoj razini u odnosu na ukupni broj pristupa
- **Omjer promašaja** (engl. miss-rate): *miss_rate=1-hit_rate*, tj. broj pristupa memoriji kada podatak nije pronađen u višoj razini u odnosu na ukupni broj pristupa

- Budući da procesor stalno pristupa podacima, **pretraživanje mora biti iznimno brzo** (ostvaruje se sklopovski)
  - Pretraživanje ćemo detaljnije objasniti malo kasnije

Obrada čitanja i promašaja kod čitanja

- Ako je traženi podatak u priručnoj memoriji, onda ga se jednostavno čita (pogodak), a operacija čitanja je vrlo brza
- Ako traženi podatak nije u priručnoj memoriji (promašaj), tada jedinica za upravljanje memorijom mora zahtijevati taj podatak iz glavne memorije (zapravo, cijeli blok)
- U tom trenutku rad procesora se mora zaustaviti dok se traženi podatak (tj. blok) ne dohvati i spremi u priručnu memoriju. To dovodi do kašnjenja zbog promašaja (engl. miss penalty)
- Postoje mnogi načini kako se pokušava izbjeći ovaj negativni utjecaj promašaja na efikasnost rada procesora, ali bez obzira na organizaciju, promašaj uvijek degradira performanse

Obrada pisanja

- Pisanje novog podataka, odnosno promjena postojećeg podatka u priručnoj memoriji predstavlja poseban problem, bez obzira što se on već u njoj nalazi, tj. bez obzira što je došlo do pogotka*

- Problem je osiguravanje istovjetnosti (ili koherentnosti) podataka u različitim razinama memorije:
  - Ako procesor mijenja podatak u priručnoj memoriji, onda je potrebno osvježiti i originalni podatak u glavnoj memoriji

* slučaj promašaja kod pisanja obradit ćemo malo kasnije

Obrada pisanja

- Najjednostavniji algoritam je **pisanje s prosljeđivanjem** (engl. write-through)
  - Svaki puta kad procesor promijeni podatak u priručnoj memoriji, taj isti podatak se odmah prosljeđuje i zapisuje i u glavnu memoriju
  - Osnovni nedostatak: znatan gubitak vremena na pisanje u glavnu memoriju
  - Efikasnost se može povećati tako da procesor ne čeka dovršetak upisa u glavnu memoriju, već nastavlja raditi, a pisanje u glavnu memoriju se obavlja preko posebnih međuspremnika neovisno o radu procesora

Obrada pisanja

- Napredniji algoritam je **pisanje pri povratku** (engl. write-back)
  - Procesor mijenja podatak SAMO u priručnoj memoriji
  - Podatak se zapisuje u glavnu memoriju SAMO onda kada se blok izbacuje iz priručne memorije
  - Efikasniji algoritam od prethodnog, ali teži za izvedbu:
    - U glavnu memoriju se zapisuju samo oni podaci iz bloka priručne memorije koji su bili mijenjani
    - Za svaki podatak u priručnoj memoriji imamo dodatnu zastavicu u kojoj se pamti je li podatak mijenjan ili nije (engl. dirty bit)
    - Zastavica se postavlja kod svake operacije pisanja. Kod punjenja novog bloka u priručnu memoriju brišu se sve zastavice za taj blok

Obrada promašaja kod pisanja

- **Promašaj pri pisanju**
  - Ako se blok u kojeg se treba upisati vrijednost ne nalazi u priručnoj memoriji, tada postoje razna rješenja
  - Najjednostavnije rješenje je da se traženi blok prvo dohvati u priručnu memoriju, a zatim obrađuje nekim od prije navedenih algoritama

Pretraživanje podataka i preslikavanje

- Problem: **kako efikasno obaviti pretraživanje**, tj. kako odrediti je li podatak u priručnoj memoriji i ako je, gdje se nalazi?
  - Prilikom pristupa podatku zadaje se njegova normalna adresa u radnoj memoriji. Potrebno je odrediti nalazi li se taj podatak u priručnoj memoriji i u kojem bloku.
  - Zapravo treba normalnu adresu podatka **pretvoriti (preslikati)** u adresu podatka/bloka u priručnoj memoriji
  - Budući da se između priručne i glavne memorije ne premještaju pojedini podaci, nego blokovi, onda promatramo adrese blokova

- Postoje mnogi načini organizacije preslikavanja podataka iz niže razine u priručnu memoriju:
  - direktno preslikavanje
  - skupovna asocijativnost
  - puna asocijativnost

Direktno preslikavanje

- Jedan od najjednostavnijih načina preslikavanja je **direktno preslikavanje**
  - Podatak s određene adrese iz glavnoj memoriji može se preslikati **samo na jednu točno definiranu adresu** u priručnoj memoriji.
  - Formula koja obavlja direktno preslikavanje je*:

    ADRBLK_u_priručnoj_mem =

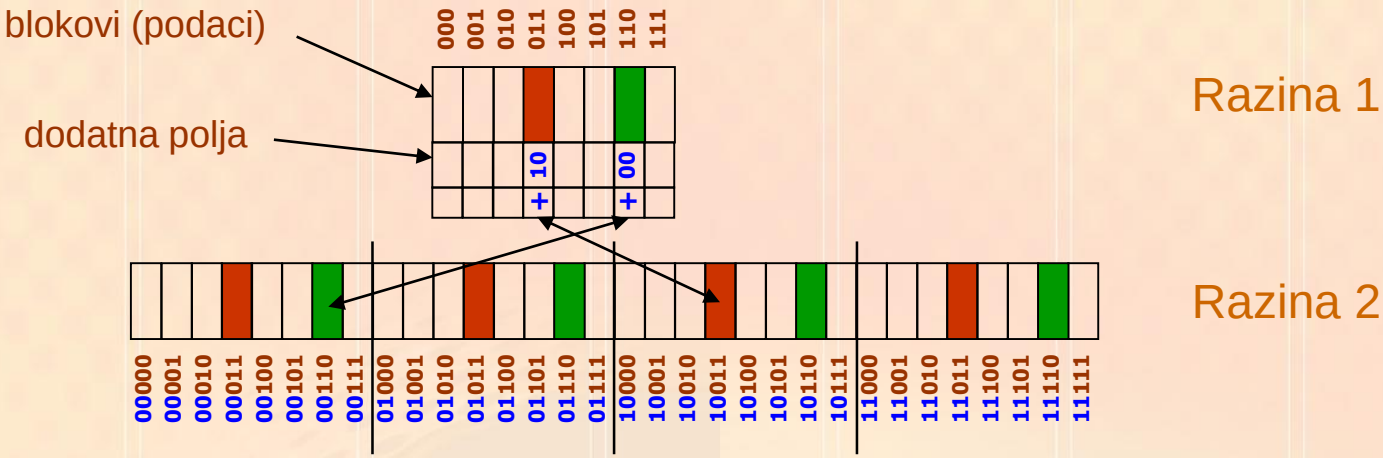
    ADRBLK_u_glavnoj_mem **mod** Broj_blokova_u_priručnoj_mem
  - To znači da se više blokova iz glavne memorije preslikavaju u jedan blok priručne memorije
    - Na primjer, ako priručna memorija ima 8 blokova, onda se u blok 3 u priručnoj memoriji preslikavaju blokovi 3, 11, 19,... glavne memorije (jer vrijedi: 3 = 3 mod 8 = 11 mod 8 = 19 mod 8 = …)
    - Posljedica je da u priručnoj memoriji ne mogu istovremeno biti neki od blokova iz glavne memorije (npr. 3 i 11, 3 i 19, 11 i 19, itd.)

---

* ADRBLK je kratica za *adresa bloka*

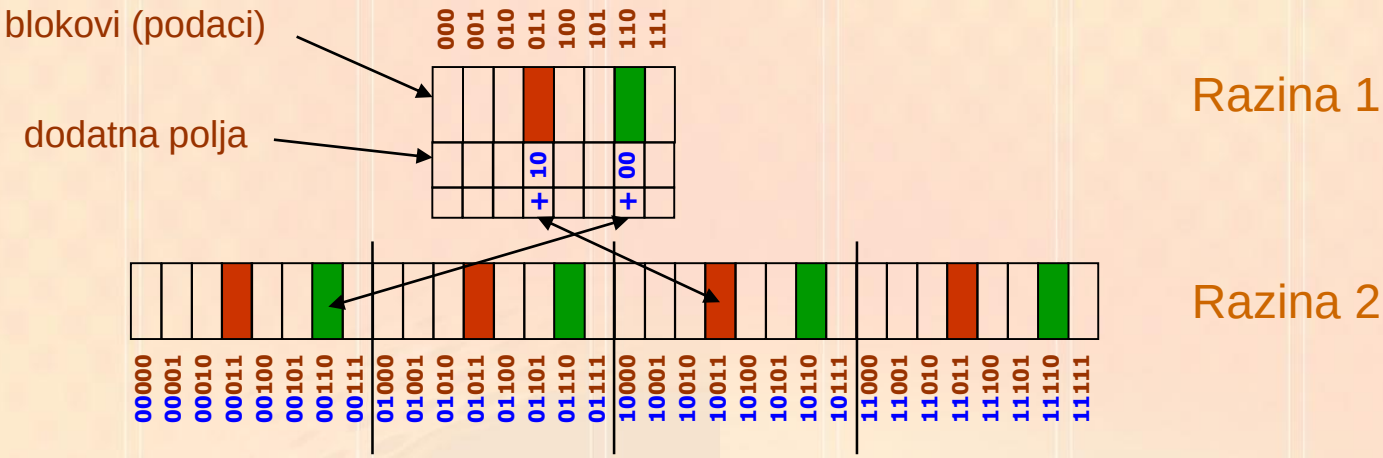
Direktno preslikavanje

Primjer priručne memorije kapaciteta 8 blokova i glavne memorije kapaciteta 32 bloka:



- Zeleni blokovi u glavnoj memoriji mogu se preslikati samo na položaj zelenog bloka u priručnoj memoriji (isto vrijedi i za crvene blokove)
- Na slici je prikazana situacija gdje se u priručnoj memoriji trenutačno nalaze samo dva bloka iz glavne memorije: 00110 i 10011 koji su označeni strelicama (ostali su blokovi priručne memorije neiskorišteni)
- Na slici su prikazane adrese blokova, a ne pojedinih podataka, jer pri preslikavanju radimo s adresama blokova

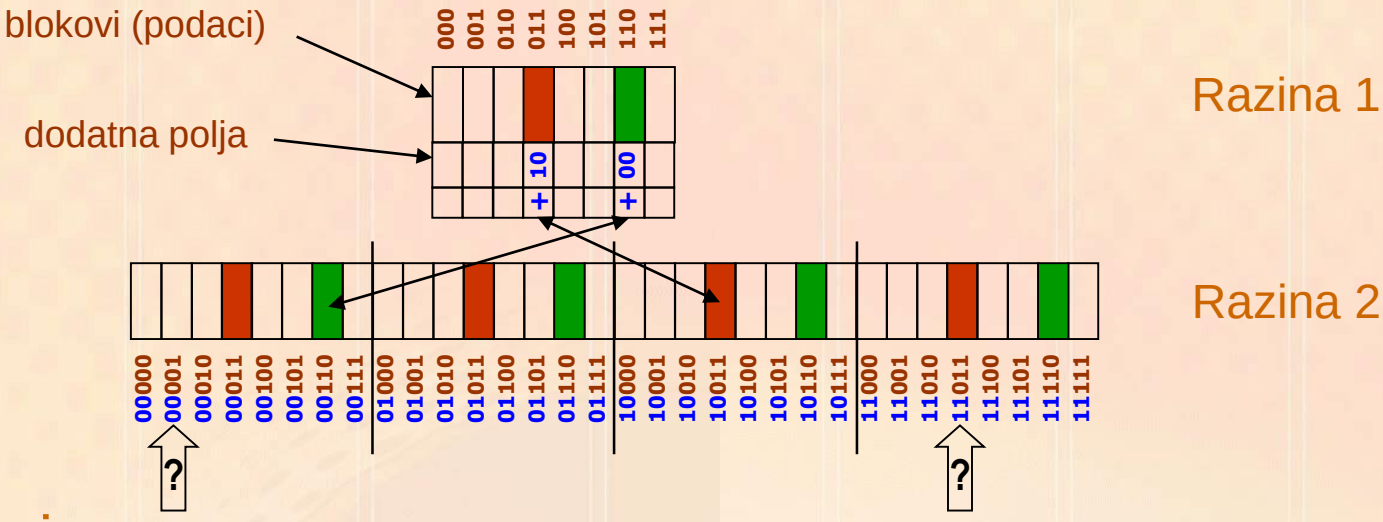
Direktno preslikavanje



- Na temelju nižih bita adrese bloka (tri smeđa bita na slici - to je zapravo operacija *mod 8*) zna se gdje bi podatak mogao biti u priručnoj memoriji

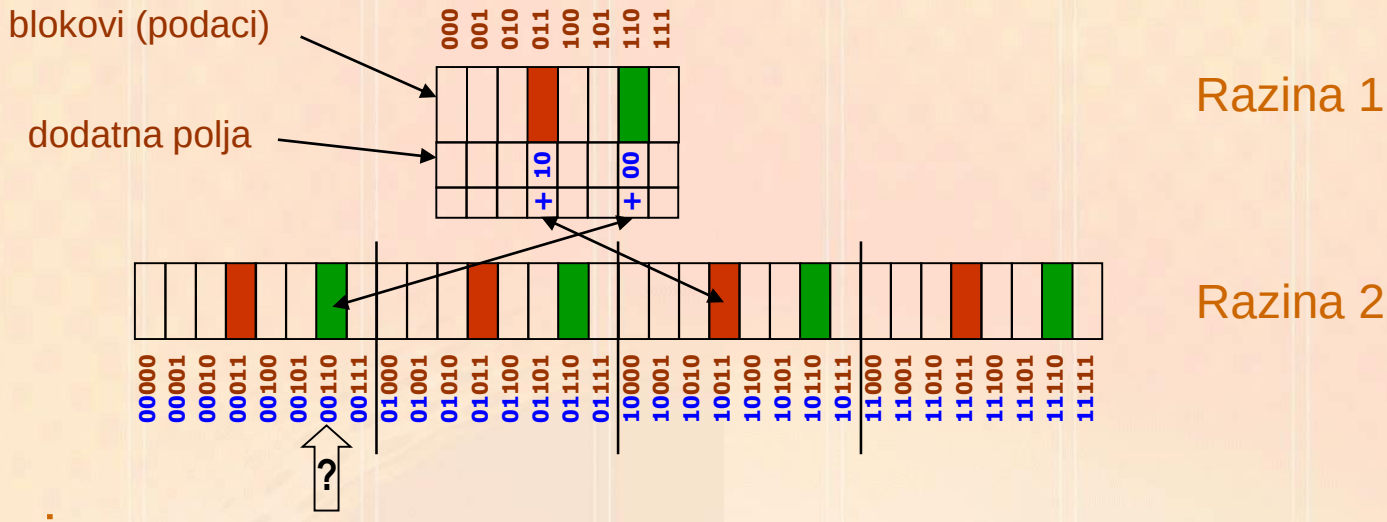
- Svakom bloku u priručnoj memoriji pridruženo je dodatno polje (engl. tag). U njima su informacije o višim bitovima adrese (dva plava bita na slici) te o prisutnosti/valjanosti bloka (znak + na slici). Ispitivanje je li traženi podatak prisutan ili nije obavlja se provjerom dodatnih polja za blok odabran pomoću preslikavanja

Direktno preslikavanje



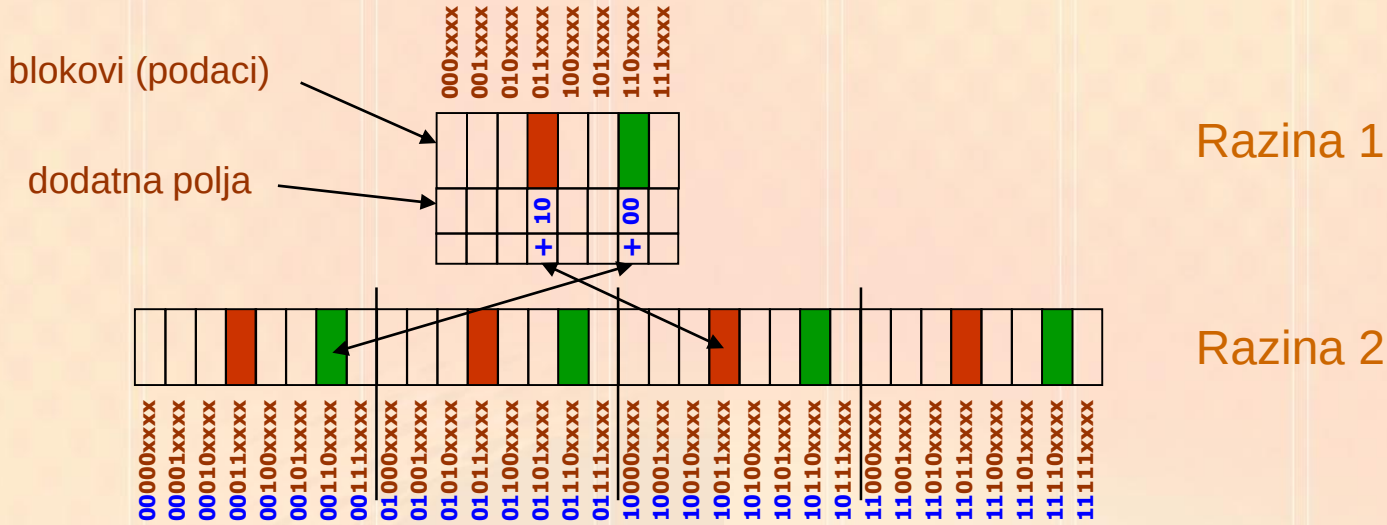
- Na primjer:
  - Ako se zatraži blok 00001, onda on (zbog direktnog preslikavanja) može biti u priručnoj memoriji na adresi 001. Za blok 001 se u dodatnom polju prvo provjerava oznaka valjanosti pomoću koje se ustanovi da blok 00001 sigurno nije u priručnoj memoriji, jer na toj poziciji nema niti jednog bloka (promašaj)
  - Ako se zatraži blok 11011, onda on može biti u priručnoj memoriji na adresi 011 pa se prvo provjerava oznaka valjanosti pomoću koje se ustanovi da u tom bloku priručne memorije postoji neki blok (ima oznaku +), ali se ne zna koji. Zato se dalje provjerava dodatno polje, ali u tom polju piše 10 što nije isto kao viši dio adrese traženog bloka 11 pa znači da se tu ne nalazi traženi blok (promašaj)

Direktno preslikavanje



- Na primjer:
  - Ako se zatraži blok 00110, onda on može biti u priručnoj memoriji na adresi 110 pa se prvo provjerava oznaka valjanosti pomoću koje se ustanovi da u tom bloku priručne memorije postoji neki blok, ali se ne zna koji. Zato se dalje provjerava dodatno polje, a u tom polju piše 00 što je isto kao viši dio adrese traženog bloka 00110 pa znači da se traženi blok nalazi u priručnoj memoriji (pogodak)

Direktno preslikavanje



- Kad bi umjesto adresa blokova htjeli prikazati adrese pojedinih podataka, to bi izgledalo kao na ovoj slici, gdje je pretpostavljeno da svaki blok sadrži 16 podataka (za čije adresiranje trebaju 4 bita)
- Oznake xxxx predstavljaju moguće adrese (od 0 do 15) pojedinih podataka unutar bloka od 16 podataka: vidimo da se iz adrese podatka vrlo lako dobiva adresa bloka – potrebno je samo odbaciti određen broj najnižih bitova (u ovom primjeru treba odbaciti 4 najniža bita)

Direktno preslikavanje - svojstva

- Prednost: direktno preslikavanje je jednostavno
  - Pretraživanje podatka/bloka je brzo
  - Za svaki blok iz glavne memorije jednostavno se i jednoznačno određuje gdje se u priručnu memoriju treba smjestiti
  - Neki drugi blok, koji se tu već nalazi, treba izbaciti iz priručne memorije
  - Jednostavna sklopovska izvedba

- Nedostatak: nema izbora gdje će se u priručnu memoriju smjestiti blok iz glavne memorije
  - Postoji relativno velika vjerojatnost da je potrebno istodobno koristiti dva ili više blokova iz glavne koji se preslikavaju u isti blok priručne memorije
  - Posljedica su učestalo izbacivanje i učitavanje blokova uslijed čestih promašaja što može znatno narušiti performanse

  - Za poboljšanje performansi može se primijeniti drugačiji mehanizam preslikavanja >>>>

Skupovna asocijativnost

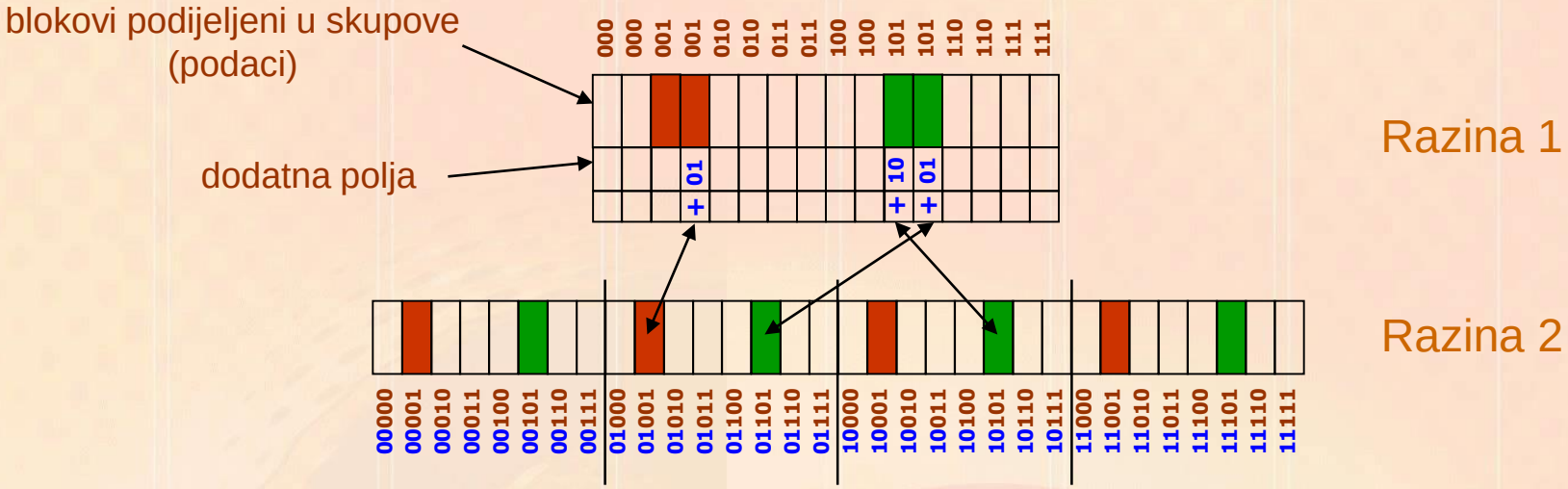
- Radi smanjenja broja promašaja i poboljšanja performansi uvodi se **skupovna asocijativnost**
  - To je mogućnost da priručna memorija može unutar jednog **skupa blokova** pohraniti **nekoliko blokova** iz glavne memorije (koji bi u direktnom preslikavanju spadali u isti blok)
  - Formula koja obavlja skupovno asocijativno preslikavanje je:

    Adresa_skupa__priručna_memorija =

    ADRBLK__glavna_memorija mod Broj_skupova_u_priručnoj_mem
  - Blok glavne memorije uvijek se preslikava u isti skup, ali u skupu ima mjesta za više blokova
  - U isti skup preslikava se više blokova glavne memorije, ali u skupu ima mjesta za više blokova
  - Radi efikasnosti izvedbe skupovi obično sadrže 2, 4, 8, ... blokova
    - Na primjer, ako skup ima dva bloka onda se za priručnu memoriju kaže da ima dvostruku asocijativnost (engl. two-way set-associative cache)

Skupovna asocijativnost

Primjer priručne memorije s dvostrukom asocijativnošću (tj. sa po dva bloka u skupu) i sa 8 skupova



- Zeleni blokovi u glavnoj memoriji mogu se preslikati na jedan od dva položaja u zelenom skupu priručne memorije (isto vrijedi za crvene blokove)
- Na slici je prikazana situacija gdje se u priručnoj memoriji trenutačno nalaze tri bloka iz glavne memorije: 01001 u crvenom skupu te 01101 i 10101 u zelenom skupu koji su označeni strelicama (ostali su blokovi priručne memorije neiskorišteni)
- Za glavnu memoriju su prikazane adrese blokova, a za priručnu memoriju adrese skupova

Skupovna asocijativnost

- Kad se traži neki blok, onda se prvo obavi preslikavanje njegove adrese u adresu skupa
- Za razliku od direktnog preslikavanja gdje je preslikavanje dalo jednoznačni položaj bloka, ovdje moramo ispitati sve blokove u skupu, jer traženi blok može biti u bilo kojem od njih
  - Ovo ispitivanje se odvija slično kao kod direktnog preslikavanja (valjanost i viši bitovi adrese), ali se to radi za svaki blok u skupu pomoću specijalnih sklopova (sklopovi za usporedbu i multipleksori)
- Što je veći broj blokova u skupu, to je manja vjerojatnost da će istodobno trebati upravo oni blokovi iz glavne memorije koji su svi u istom skupu i da će ih biti toliko puno da ne stanu u skup
  - Posljedica je smanjenje broja promašaja i znatno povećanje performansi memorijskog sustava

Skupovna asocijativnost

- Ako traženi podatak/blok nije u priručnoj memoriji, mora se izbaciti jedan od postojećih blokova iz skupa kojemu traženi blok pripada
- Kod direktnog preslikavanja je postojao samo jedan blok, pa se on izbacivao i zamjenjivao traženim blokom. Ovdje postoji **više potencijalnih blokova** koji se mogu izbaciti što **komplicira** zamjenu blokova.

- Postoje brojni **algoritmi zamjene** kojima se izabire koji će se blok izbaciti da bi se ubacio blok s traženim podatkom:

  - Najjednostavniji takav algoritam zamjenjuje blok koji se najdulje vrijeme nije koristio (engl. LRU - Least Recently Used)
    - Izvedba algoritma je jednostavna za skupove s 2 bloka. Dodaje se samo jedan bit u dodatna polja bloka. Taj bit se postavlja kad procesor zatraži podatak iz tog bloka, a bit u susjednom bloku se briše. Za izbacivanje se odabire blok u kojemu bit nije postavljen.
    - Za skupove s više blokova izvedba se komplicira

  - Alternativni algoritam je zamjena slučajno odabranog bloka. Ovo je jednostavan algoritam, ali daje veći omjer promašaja

Skupovna asocijativnost - svojstva

- Prednost: veća efikasnost zbog manjeg broja promašaja
  - Položaj blokova je fleksibilniji nego kod direktnog preslikavanja pa je manja vjerojatnost da će trebati u isti skup staviti više blokova koji su istovremeno potrebni
  - Ako je za neku primjenu veća vjerojatnost da istovremeno trebaju u priručnoj memoriji biti blokovi iz istog skupa, može se upotrijebiti organizacija s većim brojem blokova u skupu

- Nedostatak: složenija i skuplja izvedba
  - Potrebni su dodatni algoritmi izbacivanja koji ne postoje u direktnom preslikavanju
  - Što ima više blokova u skupu, izvedba sklopova za traženje i zamjenu postaje sve kompliciranija, složenija i skuplja
  - Što ima više blokova u skupu, postupci traženja postaju sve sporiji. Time se povećava vrijeme pristupa priručnoj memoriji (vrijeme pogotka, engl. hit time) što smanjuje njene performanse

Puna asocijativnost

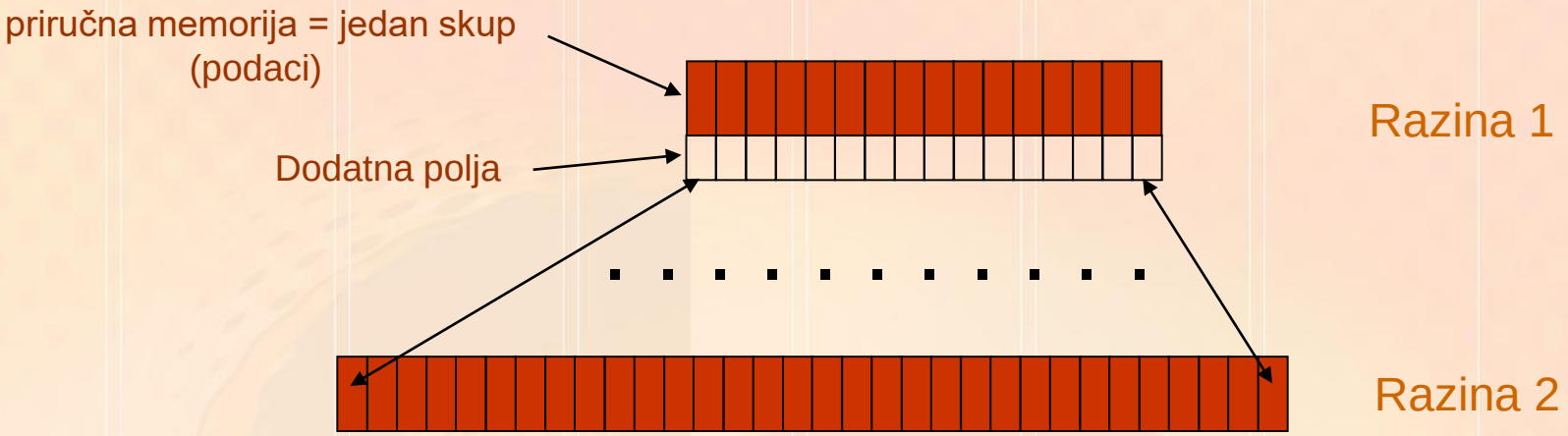
- Ako proširimo ideju skupovne asocijativnosti na cjelokupnu priručnu memoriju, tako da cijela priručna memorija predstavlja samo **jedan skup** u koji možemo pohranjivati blokove, dolazimo do najfleksibilnijeg i najefikasnijeg načina pohranjivanja blokova*
- Ovo se naziva **puno asocijativno preslikavanje**
- Na ovaj način broj promašaja maksimalno se smanjuje
- Na žalost, cijena sklopova za provjeru dodatnih polja je izuzetno velika, kao i njihova složenost pa vrijeme traženja bloka također raste
- Ova organizacija koristi se zato samo kod manjih priručnih memorija koje trebaju maksimalne performanse

---

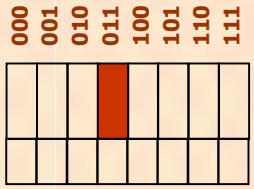
* barem teorijski, a u praksi postoje neka ograničenja

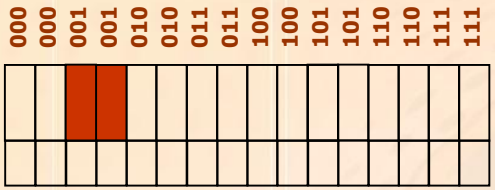
Puna asocijativnost

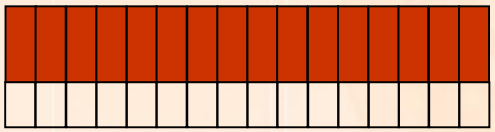
- Svaki blok iz glavne memorije može doći na bilo koji položaj u priručnoj memoriji



Rekapitulacija preslikavanja



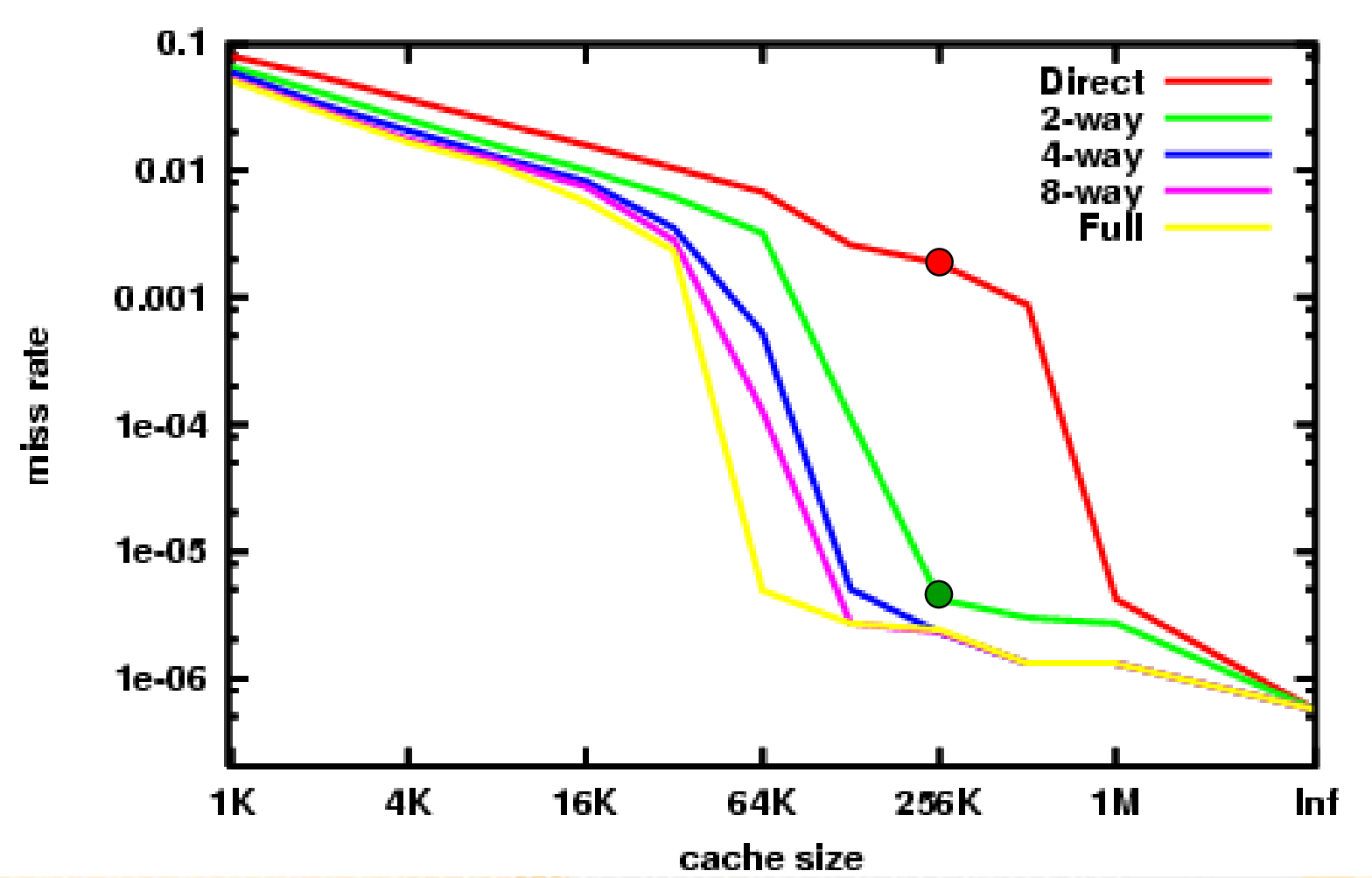




- **Direktno preslikavanje**
  - **Jedan blok samo na jedno točno određeno mjesto**
  - **Velik broj promašaja**
  - **Jednostavna i jeftina organizacija**
- **Skupovna asocijativnost s n-blokova**
  - **Jedan blok može se pohraniti na *n* mjesta unutar skupa**
  - **Značajno smanjenje broja promašaja**
  - **Potreban je algoritam zamjene blokova**
  - **Dodatna cijena sklopova za usporedbu**
- **Puna asocijativnost**
  - **Jedan blok bilo gdje u priručnu memoriju**
  - **Maksimalno smanjenje broja promašaja**
  - **Potreban je algoritam zamjene blokova**
  - **Vrlo visoka složenost sklopva i visoka cijena**
  - **Praktično samo za priručne memorije manjeg kapaciteta**

Usporedba organizacija priručne memorije

- Izvor: http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/
- Npr. miss rate za 256K cache: direct~0.002 2way~0.000003



Višerazinska organizacija priručne memorije

- Kod modernih procesora razlika između brzine priručne memorije i brzine glavne memorije je izuzetno velika te se uvode dodatne razine priručne memorije

- Tako se npr. osim priručne memorije prve razine (L1 cache), može staviti i druga razina priručne memorije (L2 cache) koja je nešto sporija od razine L1, no još uvijek brža od glavne memorije. Zatim se može staviti i priručna memorija treće razine (L3 cache) itd.

Virtualna memorija

- Povijesno su postojala dva razloga za uvođenje virtualne memorije:
  1. Glavna memorija može se gledati kao neka vrsta "priručne" memorije u odnosu na znatno sporiju masovnu memoriju (posebno važno u nekadašnje vrijeme kad su glavne memorije bile relativno malog kapaciteta)
  2. Efikasno dijeljenje i upravljanje memorijom za više programa koji se istovremeno izvode (današnja funkcija virtualne memorije)
- Svakom programu dodjeljuje se zaseban **adresni prostor** (niz virtualnih memorijskih adresa rezerviranih samo za taj program)
- Memorijski sustav obavlja **translaciju** programskog adresnog prostora (virtualnih adresa) u stvarne, fizičke adrese
- Na taj način se obavlja **zaštita** adresnog prostora nekog programa ili OS-a

Virtualna memorija

- Iz povijesnih razloga blok virtualne memorije naziva se stranica (page)
- Ako stranica nije u glavnoj memoriji, dolazi do greške na stranici (page fault), što je analogno promašaju kod priručnih memorija
- Tablica stranica (page table) sadrži translacije virtualnih adresa u fizičke. Čuva se u memoriji za svaki program
- Procesori imaju i posebnu priručnu tablicu (translation lookaside buffer, TLB) koja sadrži najčešće referencirane virtualne adrese (16-512 adresa) tako da se ne troši vrijeme na pristup tablici stranica