1. PRIJENOS PODATAKA IZ REGISTRA U REGISTAR

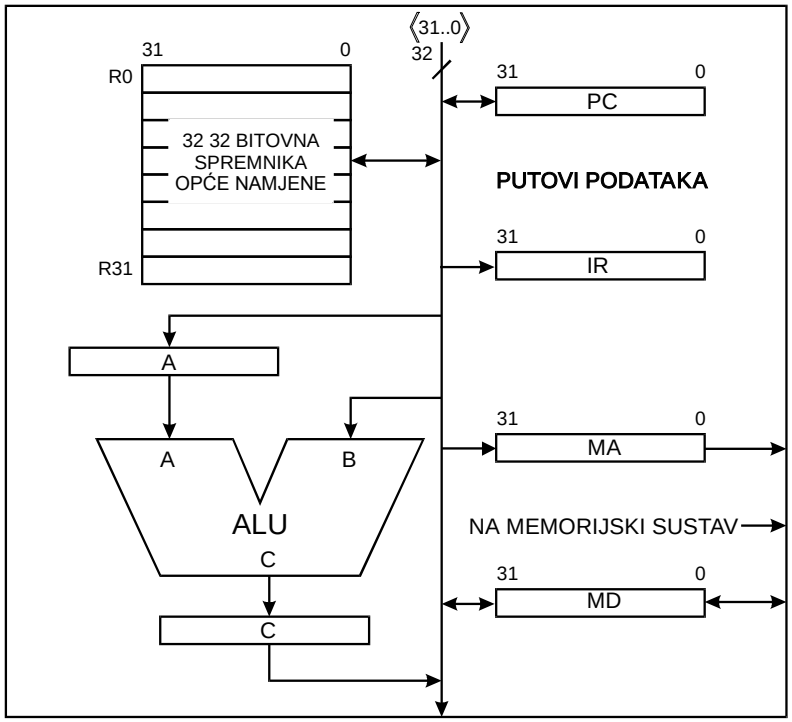
- **nacrtat shemu**
- **objasniti ciklus dohvaæanja i izvoðenja naredbi LOAD, ADD, STORE, itd.**
- **jednosabirnièka arhitektura**
- **zasto se koriste određeni sklopovi (nisam nasao)**

Slika prikazuje dio jedno-sabirničke mikroarhitekture SRCa

Postoje dodatni spremnici koji nisu opisani apstraktnim RTN opisom, A C MA MD.

RTN(*Register Transfer Notation*). Ovakav opis omogućava da se procesor opiše jednostavno i nedvosmisleno, a značajan je za sklopovsku realizaciju funkcija procesora.

Prva dva su potrebna za privremenu pohranu operanda i rezultata prilikom obavljanja ALU naredbi. MA (*memory address*) i MD (*memory data*) koriste se kao međuveza s memorijom i ulazno/izlaznim uređajima. MA sadrži memorijsku adresu operanda, dok MD je međuspremnik podataka koje ulaze u CPU ili izlaze iz njega.



Konkretan RTN opis naredbe **add** za procesor temeljen na jednosabirničkoj mikroarhitekturi prikazanoj na prethodnoj slici prikazan je sljedećom tablicom:

Korak	RTN
T0	MA ← PC: C ← PC +4;
T1	MD ← MMA: PC ← C;
T2	IR ← MD;
T3	A ← R[rb];
T4	C ← A + R[rc];
T5	R[ra] ← C;

1. Postavi sadržaj PCa i upiši u MA. Ujedno inkrementiraj sadržaj PCa pomoću ALU i rezultat pohrani u privremeni spremnik C.
2. Očitaj sadržaj memorijske lokacije na adresi na koju pokazuje sadržaj MA i upiši u međuspremnik MD. Postavi sadržaj privremenog spremnika C na sabirnicu i upiši u PC.
3. Postavi sadržaj MDa ma sabirnicu i upiši u IR. Sklopovi sada dekodiraju naredbu i zaključuju da se radi o zbrajanju.
4. Postavi sadržaj spremnika rb na sabirnicu i upiši ga u privremeni spremnik A.
5. Postavi sadržaj spremnika rc na sabirnicu, naredi ALU da izvede zbrajanje sadržaja privremenog spremnika A i sadržaja na sabirnici te rezultat upiši u privremeni spremnik C.
6. Postavi sadržaj spremnika C na sabirnicu i upiši ga u spremnik ra

Konkretan RTN opis **load** i **store** naredbi prikazan je sljedećom tablicom

Korak	**RTN** za **ld**	**RTN** za **st**
T0 –T2	Dohvat naredbe	Dohvat naredbe
T3	A ← ((rb = 0) → 0: rb ≠ 0) → R[rb];	A ← ((rb = 0) → 0: rb ≠ 0) → R[rb];
T4	C ← A + (16@IR⟨16⟩#IR⟨15..0⟩);	C ← A + (16@IR⟨16⟩#IR⟨15..0⟩);
T5	MA ← C;	MA ← C;
T6	MD ← M[MA];	MD ← R[ra];
T7	R[ra] ← MD;	M[MA] ← MD;

T3. Kada se u privremeni spremnik A upisuje 0 ako je rb = 0, odnosno u R[rb] ako je rb ≠ 0.

T4. Izračuna se efektivna adresa pribrajanjem konstante s bazom koja se nalazi u A.
T5. Upisuje se memorijska adresa u spremnik MA.

Kod **ld**, T6. Iz memorije u međuspremnik MD

T7. Iz MD u odredišni spremnik ra.

Kod **st,** T6. Podatak iz spremnika ra prebacuje se u MD

T7. Iz MD u memoriju.

2. ODREĐIVANJE MAKSIMALNE FREKVENCIJE TAKTA
- ovisi o najdužoj mogućoj stazi koju signal mora proći

Procjena minimalnog takt perioda. Minimalni period takta za zadani prijenos podatka određen je vremenom propagacije signala preko putova podataka:

$$t_{min} = t_{među} + t_{sab} + t_{komb} + t_s$$

Primjer: Izračunaj maksimalnu frekvencija takta za sklop prikazan na prethodnoj slici prema podacima iz tablice.

Za FTTL sklopove minimalni period takta iznosi:

$$t_{min} = t_{među} + t_{sab} + t_{komb} + t_s = 5 + 5 + 14 + 6 = 30\ \text{ns}.$$

(vrijednosti iz tablice)

Ovoj vrijednosti praktički se obično doda još 10% radi sigurnosti što ukupno čini 33 ns, odnosno maksimalna frekvencija takta je 30MHz.

Za GaAs VITESSE logiku minimalni period takta iznosi:

$$t_{min} = t_{među} + t_{sab} + t_{komb} + t_s = 0.15 + 0.5 + 0.4 + 0.44 = 1.49\ \text{ns}.$$

(vrijednosti iz tablice)

Ukoliko se i ovoj vrijednosti doda 10% radi sigurnosti dobiva se minimalni period takta 1.6 ns, odnosno maksimalna frekvencija 625 MHZ

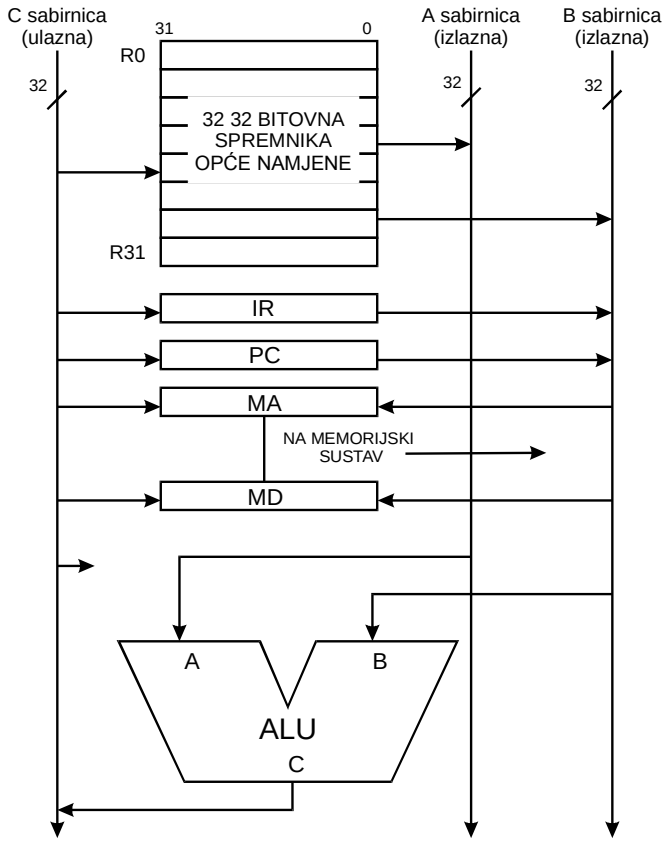
3. POBOLJŠANJE SUSTAVA

- ubrzavanje sklopova

- smanjenje takta pomoæu 3 sabirnice (nacrtat shemu)

Trosabirnički SRC

Uvođenjem treće sabirnice u arhitekturu SRCa može se istovremeno na ALU dovesti oba operanda i rezultat upisati u odredišni spremnik



Kod trosabirnicke arhitekture naredbe **ld** je smanjenja sa osam na četiri koraka. Uz povećanje perioda takta za 10% dobiva se ubrzanje od 82%. Neka je nova pretpostavka da je potrebno tri ciklusa za pristup memoriji umjesto jedan kako je prvobitno pretpostavljeno, odnosno potrebna su dva ciklusa za čekanje. Nadalje, neka je 20% naredbi koje upisuju sadržaj iz memorije u spremnik, ld, te da kod upisa u memoriju nema čekanja. Kako svaka naredba mora izvest dohvat naredbe iz

www.eISPIT.com - online ispiti i skripte

memorije, te još 20% naredbi čita iz memorije. Tada ce ubrzanje iznositi 48%,što opet predstavlja značajno povećanje performansi.

4. CJEVOVOD

a) objašnjenje rada

- nacrtana je shema i treba objasniti cikluse naredbi
(3 koraka za dohvat i, ovisno o naredbi,definirati i korake izvođenja naredbe)

b) opasnosti u cjevovodu

- ovisnost podataka (objasniti čemu služe nop i data forwarding)
- grananje (ako je ispunjen uvjet za grananje, jednu naredbu moramo eliminirati)
-može biti zadan dio koda u kojem treba objasniti moguće opasnosti

Dohvat i izvođenje ALU naredbi, prema slici 5.2, može se opisati na sljedeći način:

Korak 1. Naredba na koju pokazuje programsko brojilo dohvata se iz programske memorije, a sadržaj programskog brojila se inkrementira. Na kraju prvog ciklusa Upisuje se naredba u spremnik naredbe IR2 i nova vrijednost u programsko brojilo.

Korak 2. Naredba se čita iz spremnika naredbe te se dekodira njen sadržaj. Radi se o ALU naredbi.Signal neposredno određuje se iz polja operacijskog koda naredbe i koristi se za upravljanje multipleksorom Mp4.

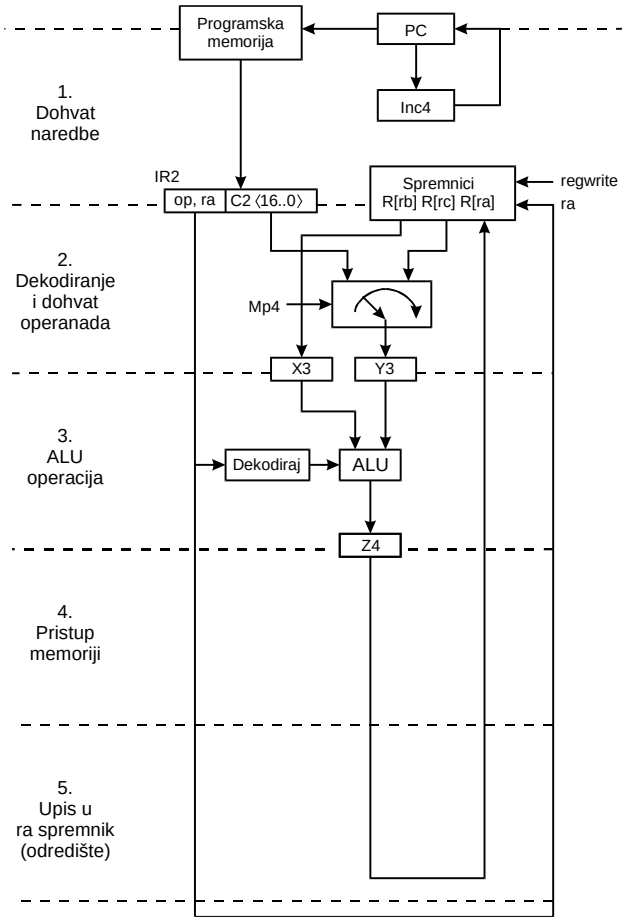
Korak 3. Naredba se dekodira kako bi se odabrala odgovarajuća ALU operacija. Rezultat se upisuje u privremeni spremnik Z4.

Korak 4. ALU naredba ne pristupa memoriji pa nema aktivnosti u ovom koraku.

Korak 5. U ovom koraku rezultat iz spremnika Z4 upisuje se u odredišni spremnik R[ra]. U ovom koraku potrebno je imati i vrijednost upisanu u Z4 kao i lokaciju odredišta, odnosno polje ra spremnika naredbe. U ovom koraku potrebno je aktivirati signal upisa u spremnike.

Privremeni spremnici X3, Y3 i Z4 omogućavaju propagaciju informacija iz jednog u drugi korak cjevovoda, a posebno su značajni kada se više različitih naredbi propagira kroz cjevovod.

www.eISPIT.com - online ispiti i skripte



Slika 5.2. Prikaz izvođenja ALU naredbi kod procesora sa cjevovodom.

Naredbe čitaj (*load*) i piši (*store*). Slika 5.3 prikazuje putove podataka kojima se realiziraju naredbe koje prebacuju sadržaj spremnika u memoriju i obratno. To su naredbe ld, ldr, st i str. Naredbe za čitanje i pisanje razlikuju se samo u četvrtom koraku, odnosno fazi pristupa memoriji. Također naredba za upis je neaktivna u petom koraku jer kod ove naredbe ne postoji upis u spremnik

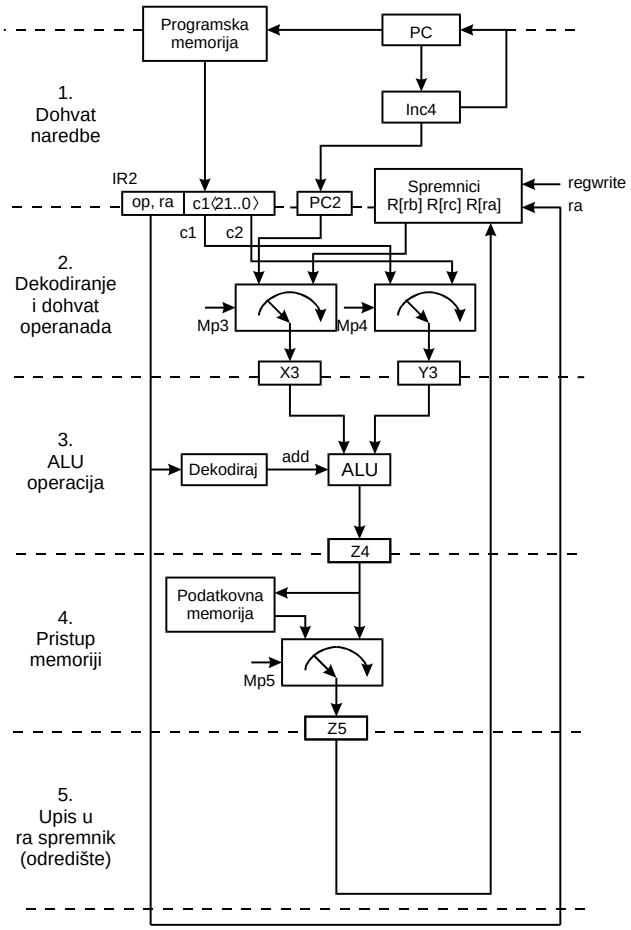
Korak 1. Dohvat naredbe i inkrementiranje sadržaja programskog brojila koji se upisuje i u spremnike PC i PC2.

Korak 2. Dohvat operanda. Ako je relativni adresni mod, tada se sadržaj PC2 i c1 prosljeđuje u spremnike X3 i Y3, a ako je apsolutni adresni mod tada se u iste spremnike prosljeđuje sadržaj spremnika rb i konstanta c2.

Korak 3. Relativna ili apsolutna adresa izračuna se zbrajanjem sadržaja spremnika X3 i Y3. Rezultat se upisuje u spremnik Z4.

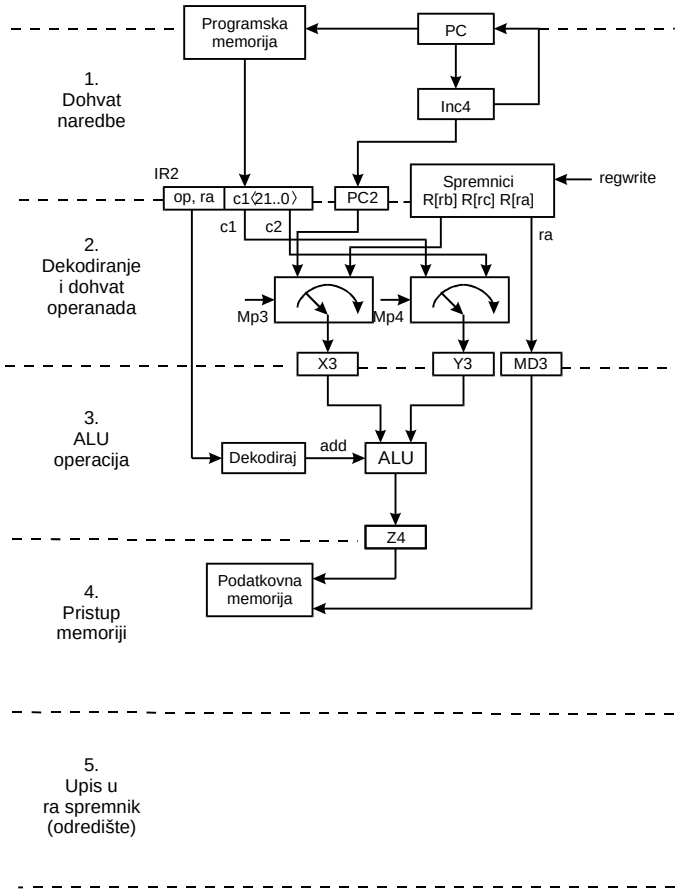
Korak 4. Ukoliko se radi o čitanju iz memorije, ld ili ldr naredbe, tada se sadržaj iz podatkovne memorije, određen adresom upisanom u Z4 upisuje u spremnik Z5. Ukoliko je upis adrese u spremnik, odnosno naredbe la ili lar, tada se sadržaj spremnika Z4 direktno prebacuje u spremnik Z5. Kod naredbi za upis u memoriju, vrijednost iz spremnika MD3 upisuje se na memorijsku adresu podatkovne memorije određene sadržajem spremnika Z4.

Korak 5. Kod svih naredbi za upis u spremnik, sadržaj spremnika Z5 upisuje se u odredišni spremnik ra., određen ra poljem spremnika naredbe IR2. Naredba za upis u memoriju je neaktivna u ovom koraku.



Slika 5.3. Naredba za upis u spremnik adrese ili sadržaja memorijske adrese

www.eISPIT.com - online ispiti i skripte



Slika 5.4. Naredba za upis iz spremnika u memoriju.

Opasnosti vezani uz primjenu cjevovoda

Posljedica su kada izvodimo naredbu koja koristi podatak iz jedne od prethodnih naredbi čija obrada još nije završila. Pogreške koje nastaju zbog primjene cjevovoda rezultat su ili loše napisanog programa prevodioca ili programske pogreške u simboličkom programu, a posljedica su ili nepoznavanja principa rada cjevovoda ili nemarnosti programera. Program prevodilac mora unaprijed predvidjeti, prije izvođenja programa, moguće izvore pogrešaka te ih otkloniti. Pri tome se rukovodi strategijom najgoreg slučaja (*worst case*). Opasnosti vezane uz primjenu cjevovoda mogu se podijeliti prema tipu naredbe uz koji su vezani. Iako postoji više mogućih podjela, osnovna je na opasnosti vezane uz podatke i one vezane uz grananja.

Opasnosti u primjene cjevovoda vezane uz podatke.

Kad naredba pristupa spremniku ili memorijskoj lokaciji prije nego je u nju neka od prethodnih naredbi upisala rezultat.

Može se analizirati sljedeći programski odsječak:

```
100:    add   r0, r2, r4
104:    sub   r3, r0, r1
```

Naredba za zbrajanje **add** zbraja sadržaje spremnika r2 i r4 i rezultat upisuje u spremnik r0. U sljedećoj naredbi rezultatu prethodnog zbrajanja oduzima se sadržaj spremnika r1. Rezultat operacije se upisuje u odredišni spremnik tek u petom koraku, dok se operandi dohvaćaju veću drugom koraku. Tako naredba za oduzimanje

pristupa spremniku r0 prije nego je prethodna naredba za zbrajanje u njega upisala rezultat. Ovo se naziva opasnost primjene cjevovoda vezan uz pristup podacima.

Sklopovsko rješenje ovog problema:

Konstrukcija sklopovlja koje može pravovremeno proslijediti potreban podatak sljedećoj naredbi, tzv. **sklopovlje za prosljeđivanje** (***forwarding hardware).***

Prvi korak vezan je uz naredbu,

- korak kada je rezultat normalno dostupan.
- korak kada je rezultat najprije dostupan.

(rezultat normalno dostupan)/(rezultat najranije dostupan) označava se kao 6/4.

drugi korak uz operande,

- korak kada je operand normalno potreban
- korak kada je operand najkasnije potreban.

(operand normalno potreban)/(operand najkasnije potreban) iznosi 2/3.

Temeljen ovih podataka moguće je izračunati minimalni potrebni razmak između dviju ovisnih ALU operacije kao (6-2)/(4-3) = 4/1, odnosno između dvije zavisne ALU naredbe potrebno je ubaciti četiri slobodna mjesta, a korištenje posebnog sklopovlja za prosljeđivanje podatka broj potrebnih slobodnih mjesta reducira se na jedno.

Ovo je samo za pristup spremnicima, dok je zanemaren pristup podacima u memoriji. Ovaj slučaj je znatno jednostavniji jer problem može samo postojati ukoliko iza naredbe za upis slijedi naredba za čitanje iz iste lokacije. Ali kako se svaki pristup memoriji odvija u četvrtom koraku ne postoji opasnost pristupa podacima.

Opasnost u cjevovodu kod naredbi za grananje. Kod grananja postoji problem da se obavezno izvodi naredba koja slijedi naredbu za grananje neovisno o rezultatu grananja. To znači da je potrebno ubaciti jedno slobodno mjesto iza naredbe za grananje. Postoje neke tehnike predviđanja rezultata grananja ali one prelaze okvire ovog kolegija.

Nop- umetanje praznih mjesta da bi se zaustavila naredba.

5. MEMORIJA
- osnovne izvedbe (ROM, PROM, EPROM, DRAM, SRAM)

RAM i ROM. RAM je skraćenica od *Random Access Memory*. memorijskim elementima RAM pristupa se proizvoljnim rasporedom i uvijek s istim vremenom pristupa. Nasuprot RAMu, ROM je preprogramirana poluvodička memorija iz koje se može samo čitati što opisuje i njen naziv: *Read Only Memory*. Programira se tvornički tako da je izrada ovakvih sklopova za manje serije ekonomski neisplativa, za svaku seriju ROM-ova potrebna je izrada maski ili filmova koja se može isplatiti tek kroz veću seriju proizvoda.

Programabilna ispisna memorija (*Programable ROM*) (PROM)

Problemi ekonomicnosti ROM-a riješeni su sklopovima nazvanim PROM (***P****rogramable* ***ROM***) koji imaju iste karakteristike kao i ROM, ali ih korisnik sam može programirati. Koristeći poseban uređaj za programiranje (PROM programator) korisnik proizvoljno pregara

(*burn, blow*) pojedine spojeve kako bi ostvario željeno funkcionalno djelovanje ovog sklopa. Jednom programirani PROM nije moguće više preprogramirati.

Izbrisivi PROM (*Erasable PROM*) EPROM

Programabilne memorije nije bilo moguće preprogramirati zato što su fizički uništene veze emitera tranzistora s izlazima. Izbrisivi PROM zasnovan je MOSFET strukturi. Ovaj MOSFET s dvostrukim vratima naziva se FAMOS (***F****loating-gate* ***A****valanche-injection* ***M****etal* ***O****xide* ***S****emiconductor*). Potpuno je okružen izolacijskim oksidnim materijalom SiO_2 i nema omskog kontakta prema vani pa se zato i naziva plivajuća vrata. Kada se narine relativno veliki napon, npr. 25V, ne postoji mogućnost pražnjenja nagomilanog naboja tako da nestankom vanjskog napona, potencijal vrata G_1 je negativan u odnosu na uvod S. Ovaj negativni potencijal onemogućava stvaranje kanala između uvoda i odvoda čak i kada se na vrata G_2 narine pozitivan napon od 5V. Rezultat opisanog postupka je zapis logičke 1 u ovoj memorijskoj ćeliji. SiO_2 je jako dobar izolacijski materijal tako da akumulirani naboj na vratima G_1 ostaje dugo uskladišten. Ispitivanja su pokazala da 70% uskladištenog naboja ostaje i nakon 10 godina pri temperaturi skladištena od 125 ^{0}C.

Ukoliko se memorijska ćelija izloži ultra-ljubičastom zračenju, SiO_2 postaje djelomično vodljiv i fotoelektrična struja odvede nagomilani naboj s plivajućih vrata G_1. Ovim postupkom briše se sadržaj memorije. Postupak brisanja traje približno 10 minuta.

Statički RAM, SRAM

Za razliku od ROMa ćelija RAMa realizirana je sa šest tranzistora spojenih u bistabil. Dva tranzistora tvore aktivni teret, dva tranzistora su u spoju bistabila te dva tranzistora služe kao sklopke koje spajaju ulaz/izlaz memorijske ćelije na zajedničku liniju.

Princip rada RAM memorijske ćelije može se opisati na sljedeći način:

- Memorijska ćelija realizirana je kao direktno vezani bistabil.
- Tranzistori za pristup ćeliji omogućavaju operacije pisanja i čitanja. Ćelija se odabire preko linije za odabir riječi (redak matrice) i u ćeliju je moguće pisati ili iz nje čitati pomoću dva tranzistora.
- Stupac memorijske matrice povezan je dvjema linijama (*dual rail*) na koje se postavlja stvarna b_i i invertirana $\overline{b}_i$ vrijednost stupca.
- Podatak se upisuje u ćeliju na način da se vrijednost b_i i $\overline{b}_i$ narine na linije preko logike za pisanje u ćeliju i odabere se riječ u koju se upisuje.

Podatak se čita iz ćelije na način da se linije predpolariziraju na napon između logičke nule i jedinice, a zatim se odabere redak i kojeg se čita. Ovaj podatak preko logike za detekciju i pojačanje vodi se na izlaz sklopa

7.2.3. Dinamički RAM

Umjesto da se informacija pohranjuje u bistabil moguće je kao memorijski element koristiti kondenzator.

Jedan tranzistor i kondenzator zamjenjuju šest tranzistora. Ukoliko se kondenzator realizira pomoću tranzistora slijedi zaključak da je potrebno tri puta manje komponenata za realizaciju ovakve memorije. Ovakva memorija radi na sljedeći način:

www.eISPIT.com - online ispiti i skripte

- Za upis podatka u memorijsku ćeliju, podatak se postavlja na b_i liniju i odabire se riječ u koju se podatak upisuje. Kondenzator se nabija ili izbija u ovisnosti o vrijednosti bita koji se upisuje.
- Čitanje podataka ostvaruje se na sličan način kao i čitanje kod statičkog RAMa. Linija za podatak se predpolariziraja na napon između logičke nule i jedinice, a zatim se odabere redak i kojeg se čita. Podatak iz kondenzatora postavlja se na liniju podatka te se preko logike za detekciju i pojačanje vodi na izlaz sklopa. U ovom procesu ukoliko je nabijen kondenzator se isprazni. Zato je potrebno osvježiti njegovo stanje.
- Sklop za detekciju i pojačanje postavlja ponovo podatak na liniju podatka i upisuje podatak u ćeliju, odnosno osvježava naboj kondenzatora. Ovaj proces naziva se ciklus osvježavanja (*refresh cycles*).

Zbog potrebe osvježavanja sadržaja memorijskih ćelija ovakva memorija naziva se dinamički RAM ili DRAM.

6. CACHE
- prevođenje adresa
- asocijativni registri i direktno mapiranje

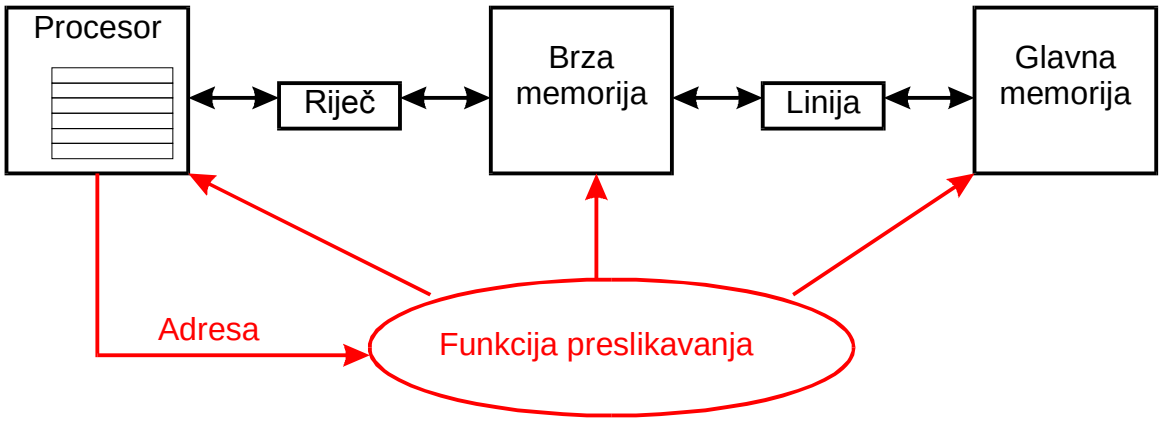
Danasnje memorije se sastoje od brze memorija (***cache***) kojoj procesor pristupa u jednom taktu, te sporije memorije koja je znatno većeg kapaciteta. Procesor pristupa informacijama, naredbe i podaci, koje su samo u brzoj memoriji. Ukoliko informacija nije u brzoj memoriji, potrebno ju je prebaciti iz sporije u brzu memoriju. Ova operacija zahtijeva dodatno vrijeme i smanjuje brzinu izvođenja obrade.

Brza memorija (*Cache*)

Rad brze memorije transparentan je programeru. Program generira efektivnu adresu (prije nazvana sistemska adresa) te definira operaciju (čitanje ili pisanje). Memorijski sustav mora realizirati ovu operaciju neovisno da li je informacija u primarnoj ili samo u sekundarnoj memoriji. Način realizacije memorijske transakcije nije vidljiv programu i programeru. Između procesora i primarne memorije prenose se riječi dok se između primarne i sekundarne memorije prenose linije (blokovi) riječi.

Funkcije preslikavanja (*Mapping function*)

Funkcije preslikavanja između različitih memorijskih razina prikazane su na slici 7.26.

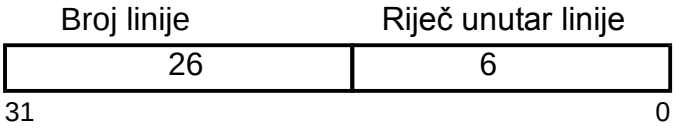


Funkcije preslikavanja odgovorne su za funkcioniranje više-razinske memorije. Zbog brzine rada ove funkcije su sklopovski realizirane i određuju sljedeće:

- Strategiju unosa linije - gdje u brzu memoriju pohraniti liniju iz glavne memorije,
- Strategiju zamjene – koju liniju iz brze memorije zamijeniti ako adresirana linija nije u brzoj memoriji (*cache miss*),
- Strategiju čitanja i pisanja – kako izvoditi operacije čitanja i pisanja ukoliko je linija u brzoj memoriji (*cache hit*) ili nije u njoj (*cache miss*).

Danas se susrećemo tri različite funkcije preslikavanja: **asocijativno** (*associative*), **direktno** (*direct*), **asocijativno po skupinama blokova** (*block-set-associative*).

Kako bi se realizirala funkcija preslikavanja memorijska adresa dijeli se na dva dijela: broj linije i pomak riječi unutar linije. U sljedećem primjeru 32 bitovna memorijska adresa podijeljena je na 26 bita za broj linije i 6 bita za pomak riječi unutar linije veličine:

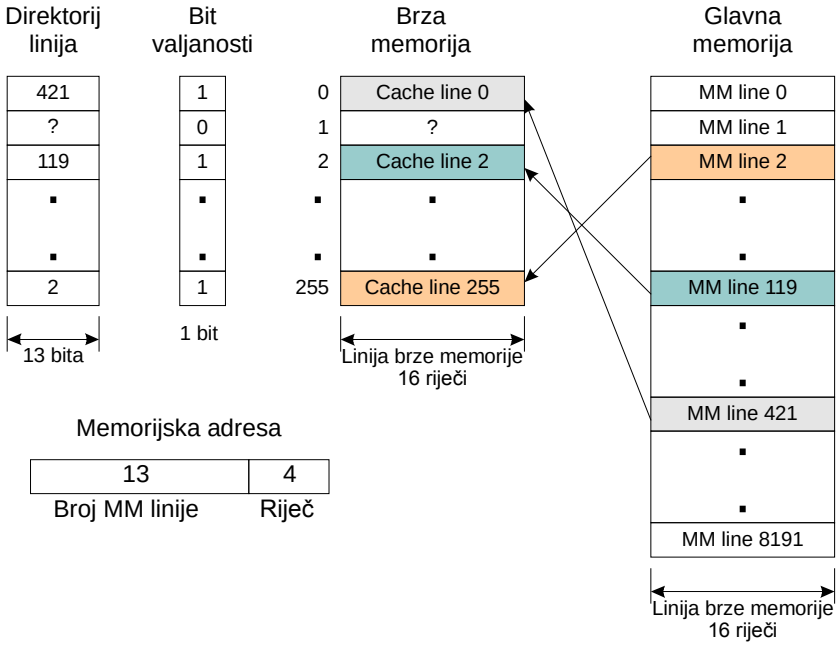


Na ovaj način moguće je adresiranje 2^{28} = 256 M linija veličine 2^4 = 16 riječi.

Asocijativno preslikavanje

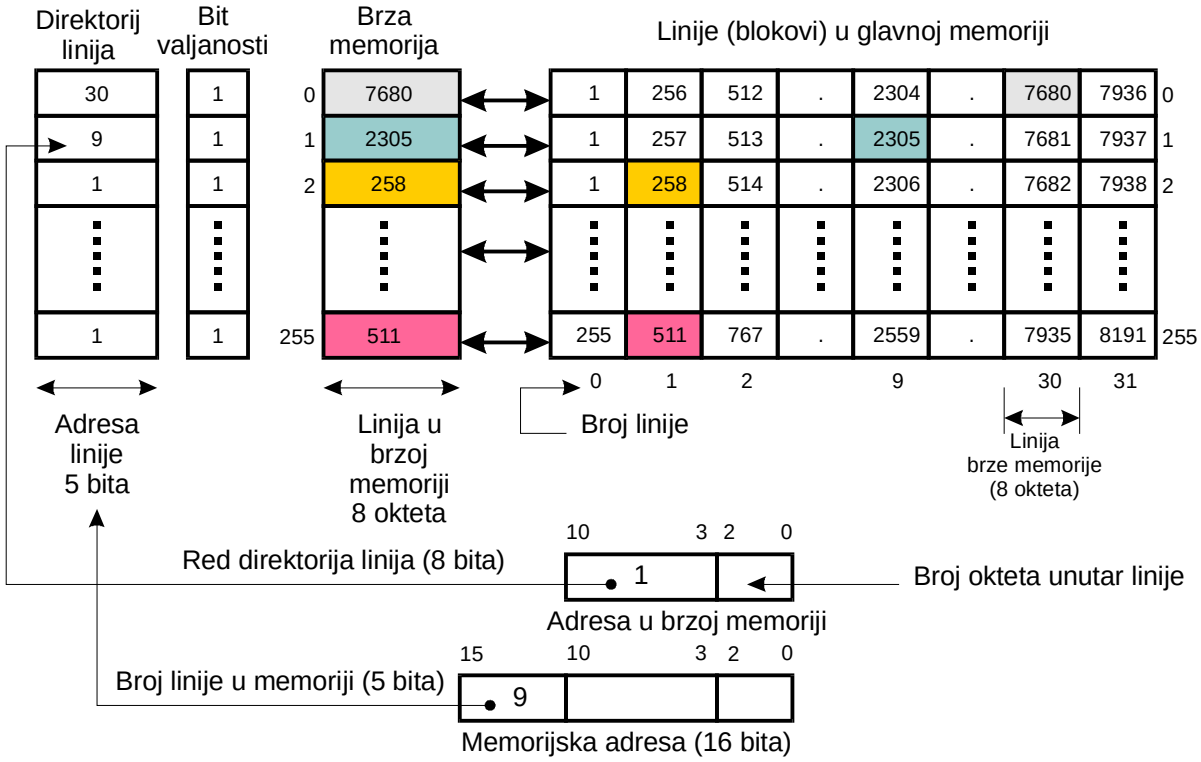
Kod asocijativnog preslikavanja svaka linija iz glavne memorije može se smjestiti bilo gdje u brzoj memoriji. Nakon što se unese u brzu memoriju linija je jedinstveno identificirana brojem linije ili znakom (*tag*) koji se upisuje u posebni dio brze memorije direktorij blokova (*tag memory*). Također bez obzira na vrstu brze memorije linija upisana u brzoj memoriji može ali i ne mora sadržavati valjane podatke. Npr. prilikom ukapčanja sustava svi podaci u brzoj memoriji su nevažeći. To je razlog da se uz direktorij linija uvodi i bitovi valjanosti (*valid bit*) koji označavaju da li je dana linija važeća ili ne.

Na slici 7.27. prikazan je primjer realizacije asocijativne memorije s 256 linija veličine 16 riječi.



Direktno mapirana brza memorija (*Direct-Mapped Cache*)

Direktno mapirana memorija je sasvim različit pristup pohrane memorijske linije. Za razliku od asocijativne memorije gdje se memorijska linija mogla upisati na proizvoljno mjesto u brzoj memoriji, kod direktno mapirane brze memorije linija se može nalaziti samo na određenom mjestu u brzoj memoriji. Na slici 7.30. je prikazan primjer direktno mapirane memorije.



Slika 7.30. Direktno mapirana brza memorija.

Glavna memorija je podijeljena na 8191 linija s 8 okteta. Blokovi su postavljeni u 256 redova s 32 linija. Brza memorija ima 256 linija (koliko ima redova glavne memorije) te se u brzu memoriju prebacuje samo po jedna linija iz svakog pojedinog reda iz glavne memorije. U direktorij linija upisuje se broj stupca u kojem je linija pohranjena u glavnoj memoriji a koji je upisan u brzoj memoriji. Tako bitovi 3 – 10 memorijske adrese (8 bita) određuje koji se red adresira, bitovi 11 – 15 stupac u kojem je memorijska linija (5 bita), a bitovi 0 – 2 (3 bita) oktet unutar linije. Na primjeru na slici u prvom redu se nalaze linije 0, 256, 512, ..., n·256, n $\in$ 0, 31, u drugom 1, 257, ..., n·256+1, itd. U brzoj memoriji je upisana 30. linija iz prvog reda (linija broj 30·256 = 7680) u drugom 9. linija iz drugog reda (linija broj 9·256 + 1 = 2305), itd.

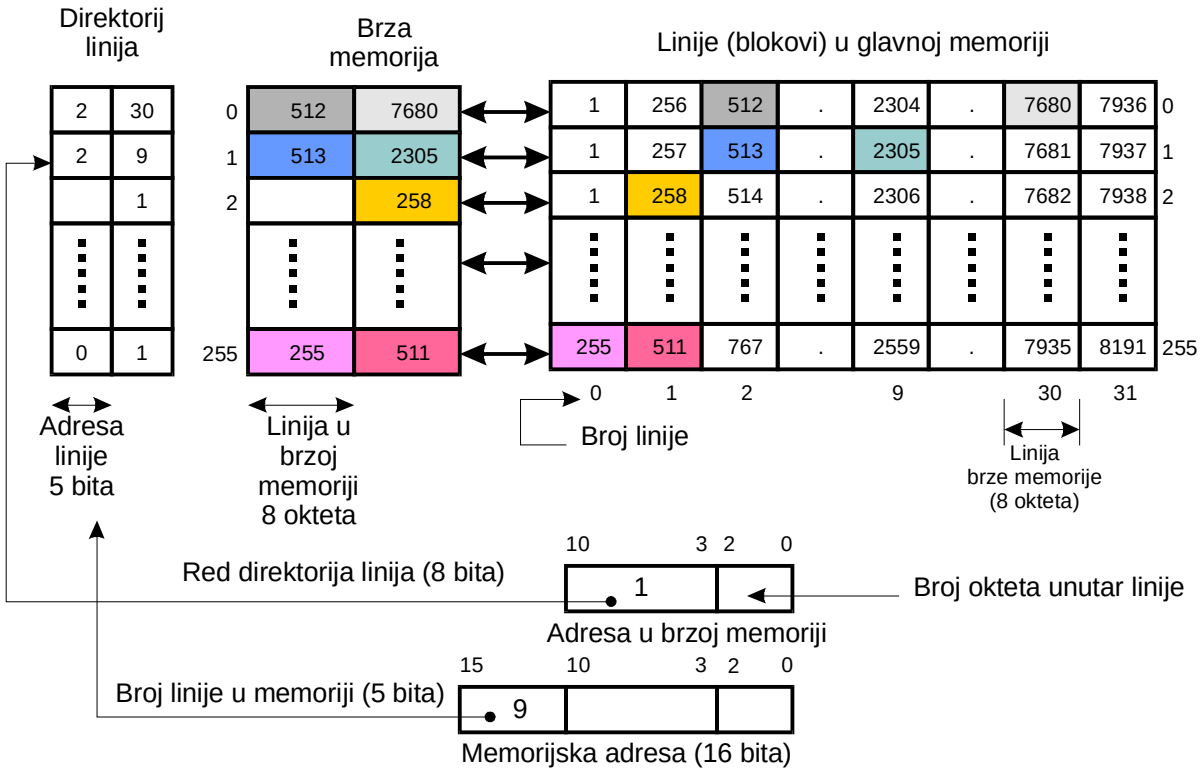
Memorijske linije asocijativno preslikane po skupinama

(*Block-Set-Associative Caches*)

Asocijativno preslikavanje unutar skupina linija je kombinacija dviju prethodno opisanih metoda. Direktorij linija kod direktno preslikane memorije se proširuje dodatnim stupcima. Ukoliko se proširi na dva stupca naziva se dvostruka, s četiri stupca četverostruka itd. brza memorija s linijama asocijativno preslikanim po skupinama (*Two-Way Block-Set-Associative*

www.eISPIT.com - online ispiti i skripte

Caches, Four-Way ..., itd.). Primjer brze memorije s dvije skupine linija asocijativno preslikanih po skupinama prikazan je slikom 7.32.



Slika 7.32: Brza memorija s dvije skupine linija koji su asocijativno preslikani po skupinama.