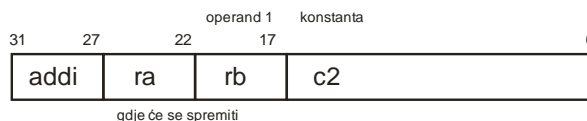


Objašnjenje upravljačke sekvence:

U koraku **T3** signalima Grb i R_{out} odabire se spremnik rb i postavlja njegov sadržaj na sabirnicu. Isti sadržaj se zatim upisuje u privremeni spremnik A aktiviranjem signala A_{in}.

U koraku **T5** se nakon postavljanja sadržaja spremnika C na sabirnicu aktiviranjem signala C_{out}, odabire spremnik ra signalom Gra i upisuje u njega sadržaj sa sabirnice aktiviranjem signala R_{in}. Sve aritmetičke i logičke naredbe koriste sličnu upravljačku sekvencu.

ADDI - naredba za neposredno zbrajanje



Korak	RTN	Upravljačka sekvenca
T0	MA ← PC; C ← PC + 4;	PC _{out} , MA _{in} , INC4, C _{in}
T1	MD ← M[MA]; PC ← C;	Čitaj, C _{out} , PC _{in} , Čekaj
T2	IR ← MD;	MD _{out} , IR _{in}
T3	A ← R[rb];	Grb, R _{out} , A _{in}
T4	C ← A + c2 {proš. predz.}	c2 _{out} , ADD, C _{in}
T5	R[ra] ← C;	C _{out} , Gra, R _{in} , Kraj

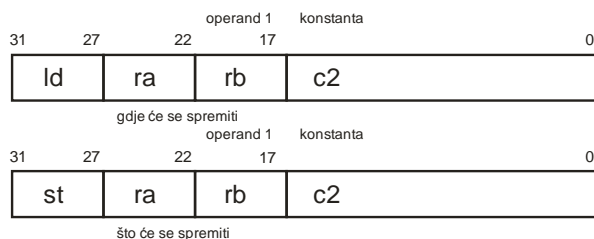
Objašnjenje RTN-a:

Velika je sličnost ove naredbe i naredbe za zbrajanje sadržaja dvaju spremnika, **add**. One se razlikuju samo u četvrtom koraku kada se drugi operand na sabirnicu postavlja direktno iz spremnika naredbe, IR, uz proširenje predznaka bitova 17..31. (**Objašnjenje:** konstanta koja je u IRu je 16bit-na i treba se pretvoriti u 32bit-nu, pa se bit predznaka prebacuje na 31. bit tako da bi se mogla zbrojiti s A).

Objašnjenje upravljačke sekvence:

U koraku **T4** drugi operand se dohvaća iz dijela spremnika naredbe aktiviranjem signala c2_{out}. Konstanti se sklopovski proširuje predznak, 16 bit, na preostale bitove 17..31.

LD (load) i ST (store) - naredbe za prebacivanje podataka



Sprema sadržaj s memorijske lokacije u registar

Sprema sadržaj registra na memorijsku lokaciju

Korak	RTN za ld	RTN za st
T0 – T2	Dohvat naredbe	Dohvat naredbe
T3	$A \leftarrow ((rb = 0) \rightarrow 0: (rb \neq 0) \rightarrow R[rb]);$	$A \leftarrow ((rb = 0) \rightarrow 0: (rb \neq 0) \rightarrow R[rb]);$
T4	$C \leftarrow A + c2 \{proš. predz.\};$	$C \leftarrow A + c2 \{proš. predz.\};$
T5	MA ← C;	MA ← C;
T6	MD ← M[MA];	MD ← R[ra];
T7	R[ra] ← MD;	M[MA] ← MD;

T3 - baza adrese određuje se kada se u privremeni spremnik A upisuje 0 ako je rb = 0, tj. sadržaj spremnika rb ako je rb ≠ 0.

T4 - izračuna se efektivna adresa pribrajanjem konstante(offset), koja je sastavni dio naredbe uz proširenje njenog predznaka, s bazom koja se nalazi upisana u privremenom spremniku A.

T5 - upisuje se memorijska adresa u spremnik MA.

load

T6 iz memorije u međuspremnik MD

T7 iz MD u odredišni spremnik ra

store

T6 podatak iz spremnika ra prebacuje se u MD

T7 iz MD u memoriju

Upravljačka sekvenca za load naredbu:

Korak	RTN	Upravljačka sekvenca
T0 –T2	Dohvat naredbe	
T3	$A \leftarrow ((rb = 0) \rightarrow 0: (rb \neq 0) \rightarrow R[rb]);$	Grb, BA _{out} , A _{in}
T4	$C \leftarrow A + c2 \{ \text{proš. predz.} \}$	c2 _{out} , ADD, C _{in}
T5	MA $\leftarrow C;$	C _{out} , MA _{in}
T6	MD $\leftarrow M[MA];$	Čitaj, Čekaj
T7	R[ra] $\leftarrow MD;$	MD _{out} , Gra, R _{in} , Kraj

T3 – sadržaj spremnika rb se postavlja na sabirnicu ne pomoću signala R_{out}, nego pomoću signala BA_{out} (upravljački signal (BA – Base Address) koji se koristi u proračunu efektivne adrese iz bazne adrese). Ovim rješenjem se na sabirnicu postavlja sadržaj odabranog spremnika ukoliko nije odabran R[0] (kada je rb=0) kada se na sabirnicu postavljaju sve 0. Ova vrijednost se upisuje u privremeni spremnik A.

T4 - na sabirnicu se postavlja konstanta c2 kojoj se proširi predznak i ona se pribroji sadržaju privremenog spremnika A. Na kraju ovog koraka u privremenom spremniku C upisana je efektivna adresa operanda iz memorije.

Posljednja tri koraka slična su dohvat u naredbe.

Upravljačka sekvenca za store naredbu:

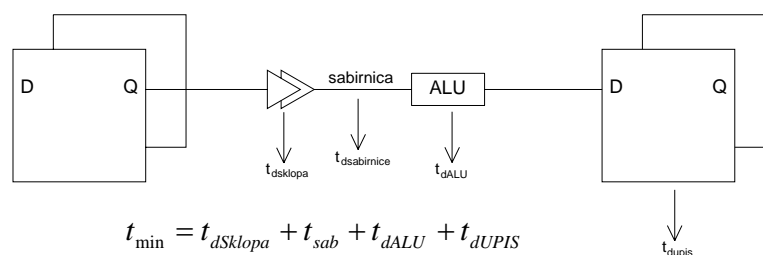
Naredba za upis sadržaja spremnika na memorijsku lokaciju, store, slična je opisanoj naredbi upisa sadržaja memorijske lokacije u spremnik. Razlika je u koracima T6 i T7.

T6 - upisuje se sadržaj odabranog spremnika u MD spremnik aktiviranjem signala MD_{bus} i Upiši. (signali MD_{bus} i MD_{rd} odabiru smjer iz kojeg se podatak upisuje u spremnik MD, prvi s procesorske sabirnice, a drugi s memorijske podatkovne sabirnice).

T7 - generira se signal Piši (Write), a signal Čekaj podrazumijeva odgovor memorije da je upisan sadržaj s podatkovne sabirnice postavljanjem signala Izvršeno.

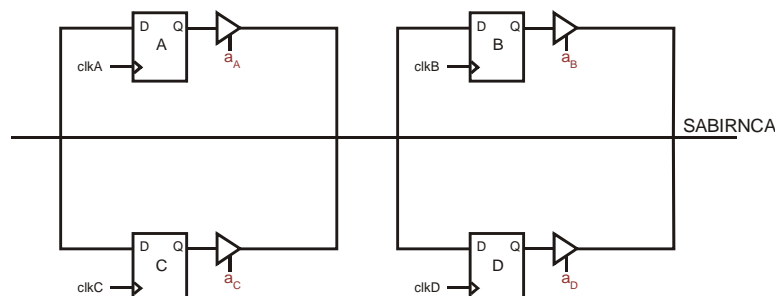
MAKSIMALNA FREKEVENCIJA TAKTA

Kašnjenje na sklopovima je ograničavajući faktor brzine rada. Kašnjenje se definira kao vrijeme koje protekne od promjene na ulazu do promjene na izlazu sklopa. Maksimalna frekvencija ovisi o ukupnom vremenu kašnjenja koje određujemo prema kašnjenju naredbe koja prolazi kroz najviše sklopova.



PRIJENOS PODATAKA IZ REGISTRA U REGISTAR

Moramo biti u mogućnosti prebaciti podatak iz bilo kojeg registra u bilo koji registar, a ne imati točno određen skup ulaznih i skup izlaznih registara (npr. registar A može u jednom trenutku biti ulaz, a u drugom izlaz).



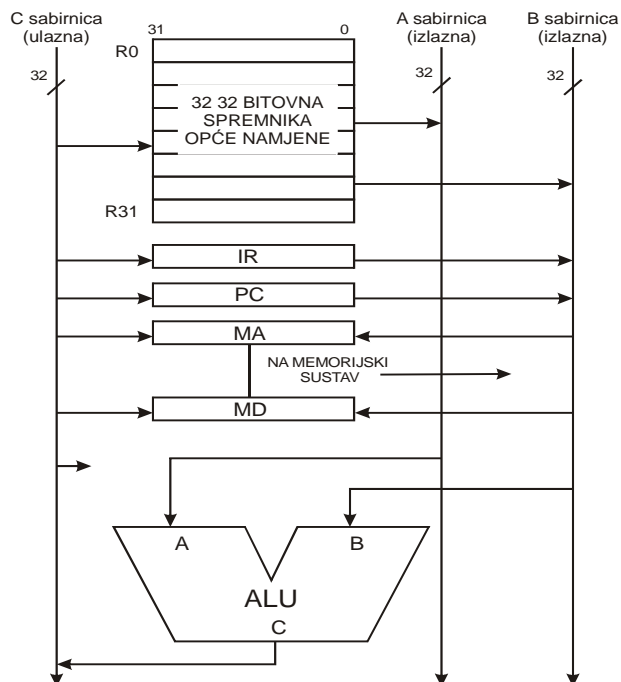
Ovo je moguća pod uvjetom da su svi a-ovi osim jednog u 0. Nakon što su samo jedna vrata(a) aktivna clock signalom se zapisuju podatak u odgovajući registar.

POBOLJŠANJE SUSTAVA:

Dvo- i tro-sabirnička SRC arhitektura: ukoliko se poveća broj sabirnica, odnosno broj međuveza, moguće je istovremeno prenositi više podataka. Tako je broj i konfiguracija sabirnica jedan od odlučujućih faktora u povećanju performansi procesora. S druge strane povećanje broja sabirnice, neovisno o razini implementacije, rezultira u povećanju cijene procesora. Ponovo potrebno je pronaći kompromis između dva oprečna zahtjeva, performanse – cijena.

Dvo-sabirnička SRC arhitektura: Detaljnom analizom se pokaže kako je stvarno poboljšanje performansi znatno ispod očekivanih. Uzimajući u obzir dodatnu složenost sklopova proizlazi zaključak da se praktički ne isplati uvoditi drugu sabirnicu.

Tro-sabirnička arhitektura:



Uvođenjem treće sabirnice u arhitekturu SRCa može se istovremeno na ALU dovesti oba operanda i rezultat upisati u odredišni spremnik.
Ovom arhitekturom smanjen je ukupan broj koraka na svega tri te je eliminiran i privremeni spremnik A.

Detaljnijom analizom se pokaže kako je stvarno poboljšanje performansi značajno. primjer. analiza naredbe ld pokazala je smanjenje broja koraka s osam na četiri

PROJEKTIRANJE PROCESORA SA CJEVOVODOM:

Kod procesora s cjevovodom praktički je uvijek jedna ili više naredbi u fazi obrade, odnosno ne postoji trenutak kad su sve naredbe izvedene. Jednostavno rečeno cjevovod podrazumijeva dohvat i početak izvođenja sljedeće naredbe prije nego što je završila prethodna naredba. Cilj cjevovoda je zaposliti što više funkcionalnih jedinica, tj. skratiti vrijeme u kojem nisu zaposlene. Cjevovod povećava propusnost procesora ali povećava i vrijeme izvođenja naredbi.

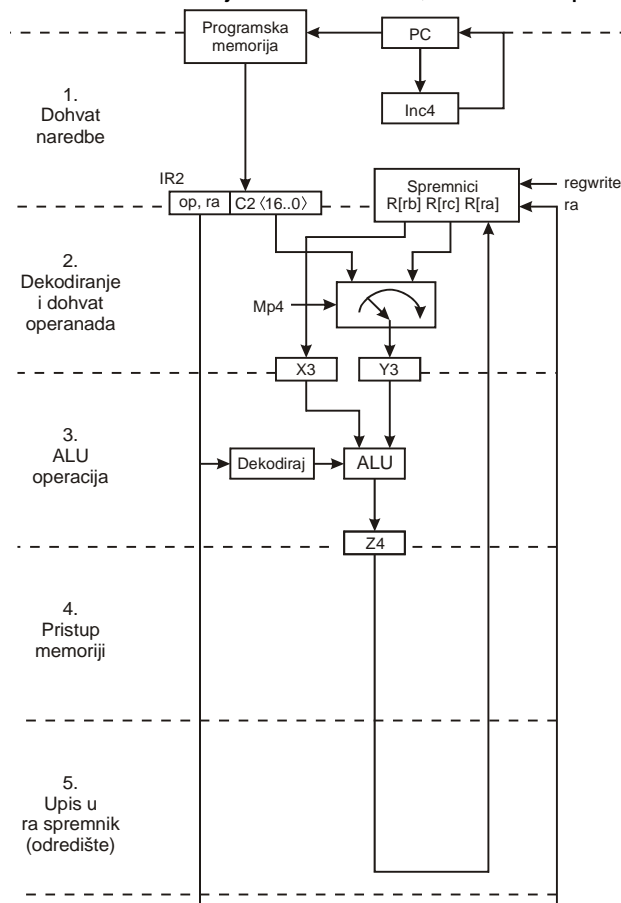
Općenito sve naredbe podijeliti će se na pet koraka:

1. dohvat naredbe,
2. dekodiranje i dohvat operandi,
3. ALU operacija,
4. pristup memoriji,
5. upis u spremnik.

Prikaz izvođenja ALU naredbi kod procesora sa cjevovodom:

Koristi se Harvardska arhitektura, odnosno nezavisna memorija za program i nezavisna za podatke. Također uveden je zaseban sklop za inkrementiranje programskog brojala kao nekoliko zasebnih spremnika za privremenu pohranu podataka.

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



Korak 1. Naredba na koju pokazuje programsko brojilo dohvaća 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 IR2 te se dekodira njen sadržaj → radi se o ALU naredbi. Ove naredbe imaju sljedeći format:

$R[ra] \leftarrow R[rb] \text{ op } R[rc]$ obrada sadržaja spremnika
 $R[ra] \leftarrow R[rb] \text{ op } c2(16..0)$ obrada sadržaja spremnika i konstante
U drugom koraku dohvaća se ili sadržaji (R[rb] i R[rc]) ili (R[rb] i c2). R[rb] se upisuje u privremeni spremnik X3 dok se u privremeni spremnik Y3 upisuje ili R[rc] ili konstanta c2. Pomoću multipleksa Mp4 odabire se koji će se podatak upisati u privremeni spremnik Y3, R[rc] ili c2.

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

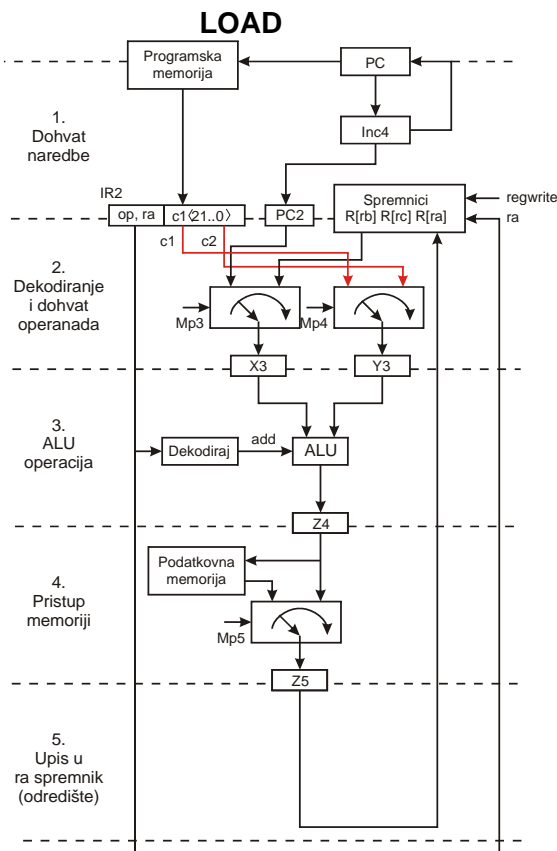
Korak 4. ALU naredba ne pristupa memoriji pa nema aktivnosti u ovom koraku.

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.

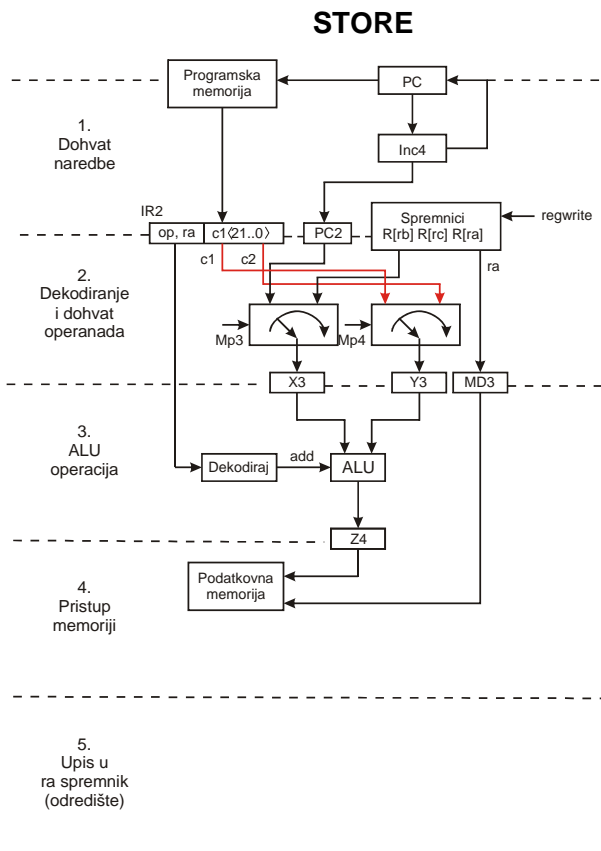
Napomena: spremnik IR se koristi u 2. koraku pa se zato zove IR2. Dakle, ovisno o koraku u kojem se koristi određeni spremnik ovisit će njegovo ime.

Naredbe: čitaj (load) i piši (store)

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.



Naredba za upis u spremnik adrese ili sadržaja mem.adrese



Naredba za upis iz spremnika u mem.

Korak 1. Dohvat naredbe i inkrementiranje sadržaja programskog brojala koji se upisuje i u spremnike PC i PC2 (jer se koristi u 2. koraku).

Korak 2. Dohvat operandata. Ako je relativni adresni mod (konstanti, tj. bazi se dodaje sadržaje PCa), tada se sadržaj PC2 i c1 proslijeđuje u spremnike X3 i Y3, a ako je apsolutni adresni mod tada se u iste spremnike proslijeđuje sadržaj spremnika rb i konstanta c2. Multiplekserima Mp3 i Mp4 određuje se koji će se sadržaji proslijediti.

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

Korak 4.

LOAD

Ukoliko se radi o čitanju iz memorije tada se sadržaj iz podatkovne memorije, određen adresom upisanom u Z4 upisuje u spremnik Z5. Ukoliko je upis adrese u spremnik tada se sadržaj spremnika Z4 direktno prebacuje u spremnik Z5.

STORE

Kod naredbi za upis u memoriju, vrijednost iz spremnika MD3 upisuje se na memorijsku adresu podatkovne memorije određene sadržajem spremnika Z4.

Korak 5. (LOAD) Kod svih naredbi za upis u spremnik, sadržaj spremnika Z5 upisuje se u odredišni spremnik ra. Naredba za upis u memoriju je neaktivna u ovom koraku.

OPASNOSTI VEZANE UZ PRIMJENU CJEVOVODA:

- opasnosti vezane uz podatke:

Opasnost primjene cjevovoda vezana uz podatke odnosi se na slučaj kad naredba pristupa spremniku ili memorijskoj lokaciji prije nego je u nju neka od prethodnih naredbi upisala rezultat.

Postoje dva rješenja ovakvog problema. Prva mogućnost je detektirati međuovisnost te zabraniti ovisnoj naredbi ulazak u cjevovod dok prva naredba ne završi s obradom. Drugo je rješenje da se rezultat obrade prve ovisne naredbe proslijedi ne čekajući fazu upisa rezultata naredbi koja koristi taj rezultat (*forwarding*).

primjer.

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`. Opis obrade naredbi sa cjevovodom pokazao je da se rezultat operacije upisuje u određeni 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.

- opasnosti kod naredbi za grananje:

Adresa naredbe koja slijedi iza naredbe za grananje poznata je tek u trenutku kada se izračuna da li je zadovoljen uvjet grananja. Kod cjevovoda ovo je poznato tek nakon drugog koraka, kada je već sljedeća naredba uzeta u obradu.

Procesor bi tada morao ovu naredbu ili odbaciti, ili umetnuti jedno slobodno mjesto iza naredbe za grananje. Program može ovdje umetnuti i naredbu koja se mora svakako izvesti bez obzira na naredbu za grananje, odnosno neovisna je o naredbi za grananje. Ova naredba poznata je pod nazivom umetak za kašnjenje grananja (*branch delay slot*).

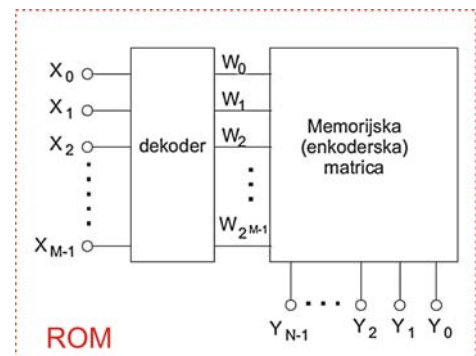
MEMORIJA:

RAM (Random Access Memory): memorijskim elementima RAM pristupa proizvoljnim rasporedom i uvijek s istim vremenom pristupa. Prema konvenciji ovaj termin rezerviran je za poluvodičku memoriju u koju se može pisati te iz koje se može čitati.

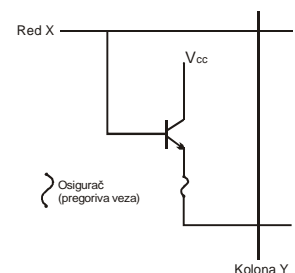
ROM je predprogramirana poluvodička memorija iz koje se može samo čitati što opisuje i njen naziv: *Read Only Memory*. Čelijama ROMa (i RAMa) može se pristupati proizvoljnim rasporedom i uvijek s istim vremenom pristupa.

Izvedba:

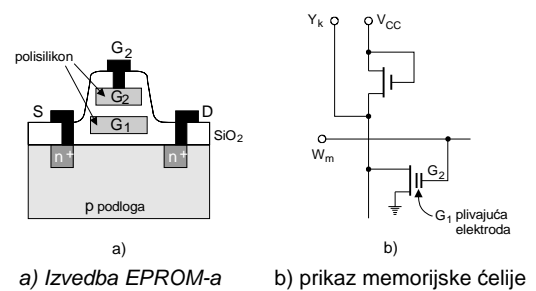
ROM se sastoji od dva dijela, dekodera i memorijske (enkoderske) matrice u kojoj su upisane informacije. Memorijska matrica može biti realizirana pomoću poluvodičkih dioda, tranzistora ili MOSFETa.



PROM: Problemi ekonomičnosti (isplativosti) 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.

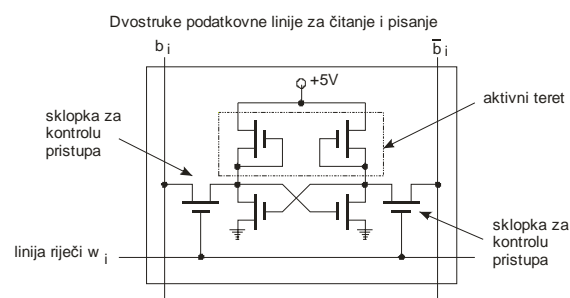


EPROM (Erasible PROM): Programibilne memorije nije bilo moguće preprogramirati zato što su fizički uništene veze emitera tranzistora s izlazima. EPROM se može više puta čitati i pisati. Izbrisivi PROM zasnovan je MOSFET strukturi. Postupak brisanja zahtijeva relativno dugo izlaganje komponente UV zračenju.

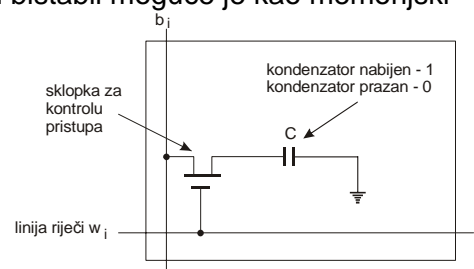


E²PROM: (Electrically Erasable PROM) Kada je EPROM sastavni dio digitalnog sklopa, neprikladan je postupak vađenja komponente, njeno stavljanje u uređaj za brisanje, te ponovo programiranje pomoću EPROM programatora. Tako je primjena EPROMa ograničena u aplikacijama koje zahtijevaju brzo i učestalo preprogramiranje memorije. Ovaj nedostatak riješen je električki izbrisivim PROM. Struktura ove memorije slična je strukturi EPROMa s time da je debljina izolacijskog sloja između vrata G₁ i kanala znatno smanjena. Dakle, čip se ne mora vaditi iz sklopa već se briše električnim putem.

SRAM (Static RAM): Proizvodi se s bistabilima. Podaci se ne uništavaju pri čitanju, pa se ne mora osvježavati. Brži su od DRAMa. Jedna memorijska ćelija se može realizirati sa samo 6 tranzistora.



DRAM (Dinamic 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. Zove se dinamički jer čitanjem prazimo kondenzator pa ga je potrebno osvježiti (upisati ponovo podatak jer smo ga čitanjem izbrisali, naboj se izgubio iz kondenzatora). DRAM ima veći kapacitet, ali manju brzinu od SRAMa.



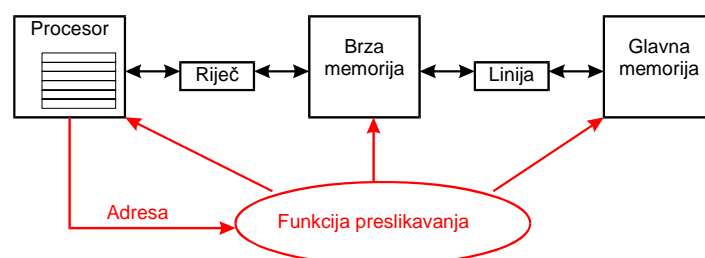
CACHE:

Današnje 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 brznoj memoriji. Ukoliko informacija nije u brznoj memoriji, potrebno ju je prebaciti iz sporije u brzu memoriju. Ova operacija zahtijeva dodatno vrijeme i smanjuje brzinu izvođenja obrade.

Cache (brza memorija): sadrži dio podataka iz glavne memorije; procesor mora znati koji mu se podaci nalaze u cache-u.

Funkcije preslikavanja (Mapping function):

Funkcije preslikavanja između različitih memorijskih razina prikazane su na slici:



Funkcija preslikavanja kod brze memorije.

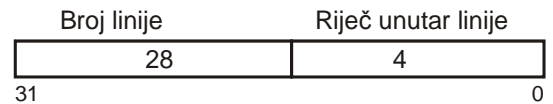
Funkcije preslikavanja odgovorne su za funkcioniranje više-razinske memorije. Zbog brzine rada ove funkcije su sklopovski realizirane i određuju sljedeće:

- 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 susreću 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 bitna memorijska adresa podijeljena je na 28 bita za broj linije i 4 bita za pomak riječi unutar linije veličine:

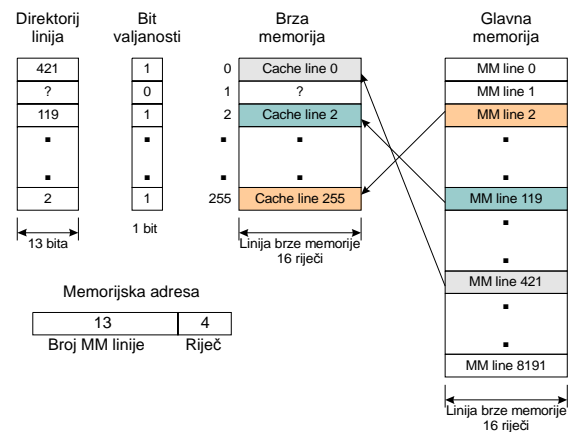
Na ovaj način moguće je adresiranje $2^{28} = 256$ M linija veličine $2^4 = 16$ riječi.



1. Asocijativno preslikavanje

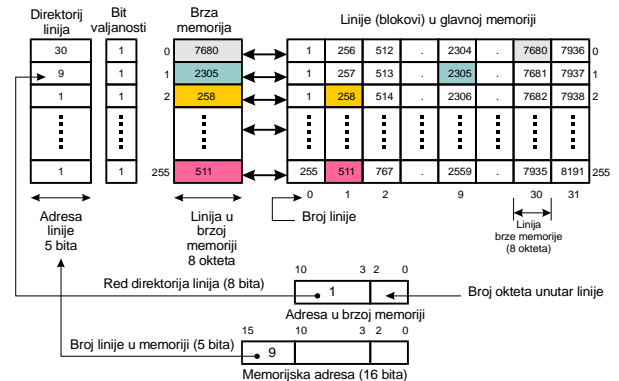
Kod asocijativnog preslikavanja svaka linija iz glavne memorije može se smjestiti bilo gdje u brzoj memoriji. Nakon što se unese u brzu memoriju linija je jedinstveno identificirana brojem linije ili znakom (*tag*) koji se upisuje u posebni dio brze memorije direktorij blokova (*tag memory*). Također 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.

-pretraživanje se vrši po sadržaju, a ne po adresi



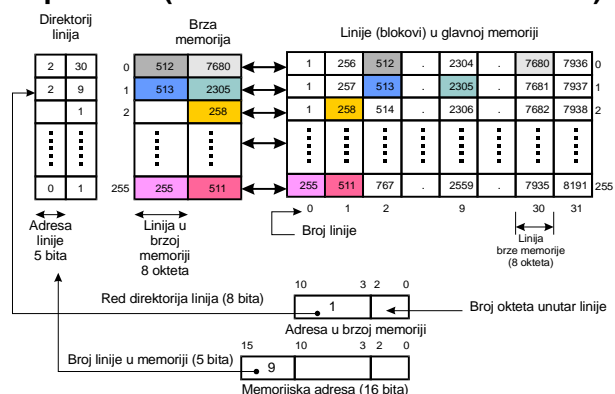
2. Direktno mapirana brza memorija (Direct-Mapped Cache)

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 je prikazan primjer direktno mapirane memorije. -određene linije mogu biti samo u određenim linijama cache-a



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



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