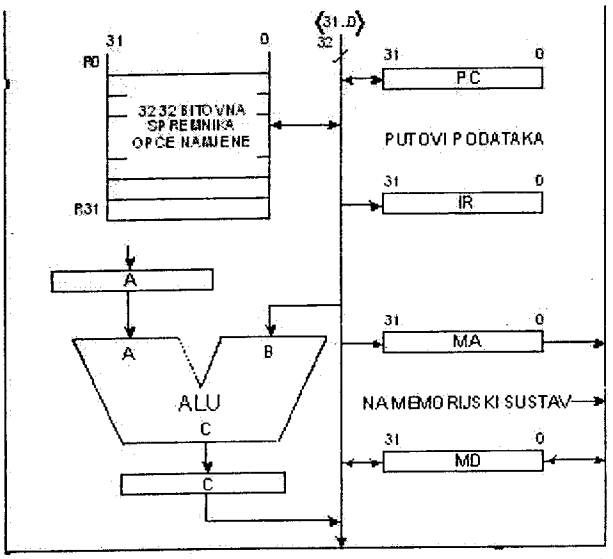
II. KOLOKVIJ

IDEALAN PRIRUČNIK za polaganje kod SVENA GOTOVCA (v1.0)

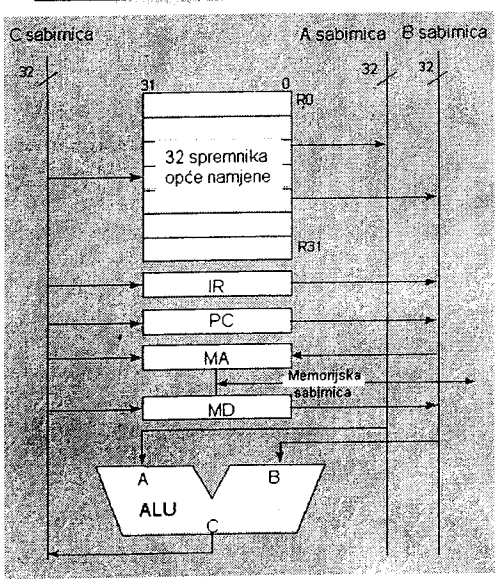
>>> by G-MEK <<<

1. PRIJENOS PODATAKA IZ REGISTRA U REGISTAR

1. Ova slika prikazuje dio jednosabirničke arhitekture SRCa. Postoje dodatni spremnici koji nisu opisani apstraktnim RTN opisom a to su A, C, MA i MD. Prva dva su potrebna za privremenu pohranu operanda i rezultata prilikom obavljanja ALU naredbi dok se MA (memory address) i MD (memory data) koriste kao međuveza s memorijom i ulazno/izlaznim uređajima. MA sadrži memorijsku adresu operanda dok je MD međuspremnik podataka koje ulaze u CPU ili izlaze iz njega. Konkretan opis naredbe **ADD** za procesor utemeljen na jednosabirničkoj mikroarhitekturi glasi : postavi sadržaj PC-a i upiši u MA. Ujedno inkrementiraj sadržaj PCa pomoću ALU i rezultat pohrani u privremeni spremnik C. 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. Postavi sadržaj MDa ma sabirnicu i upiši u IR. Sklopovi sada dekodiraju naredbu i zaključuju da se radi o zbrajanju. Postavi sadržaj spremnika rb na sabirnicu i upiši ga u privremeni spremnik A. 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. Postavi sadržaj spremnika C na sabirnicu i upiši ga u spremnik ra. Što se tiče za **LOAD i STORE** – prvo ide dohvat naredbe a zatim korak kada se u privremeni spremnik A upisuje 0 ako je rb=0, odnosno u R[rb] ako je rb različito od 0. Zatim se izdračuna efektivna adresa pribrajanjem konstante s bazom koja se nalazi u A te se upisuje memorijska adresa u spremnik MA. Kod LOAD idu zatim dva koraka – iz memorije u međuspremnik MD pa iz MD u odredišni spremnik ra dok kod STORE idu sljedeća dva koraka – podatak iz spremnika ra prebacuje se u MD pa iz MD u memoriju.



2. POBOLJŠANJE SUSTAVA CPU

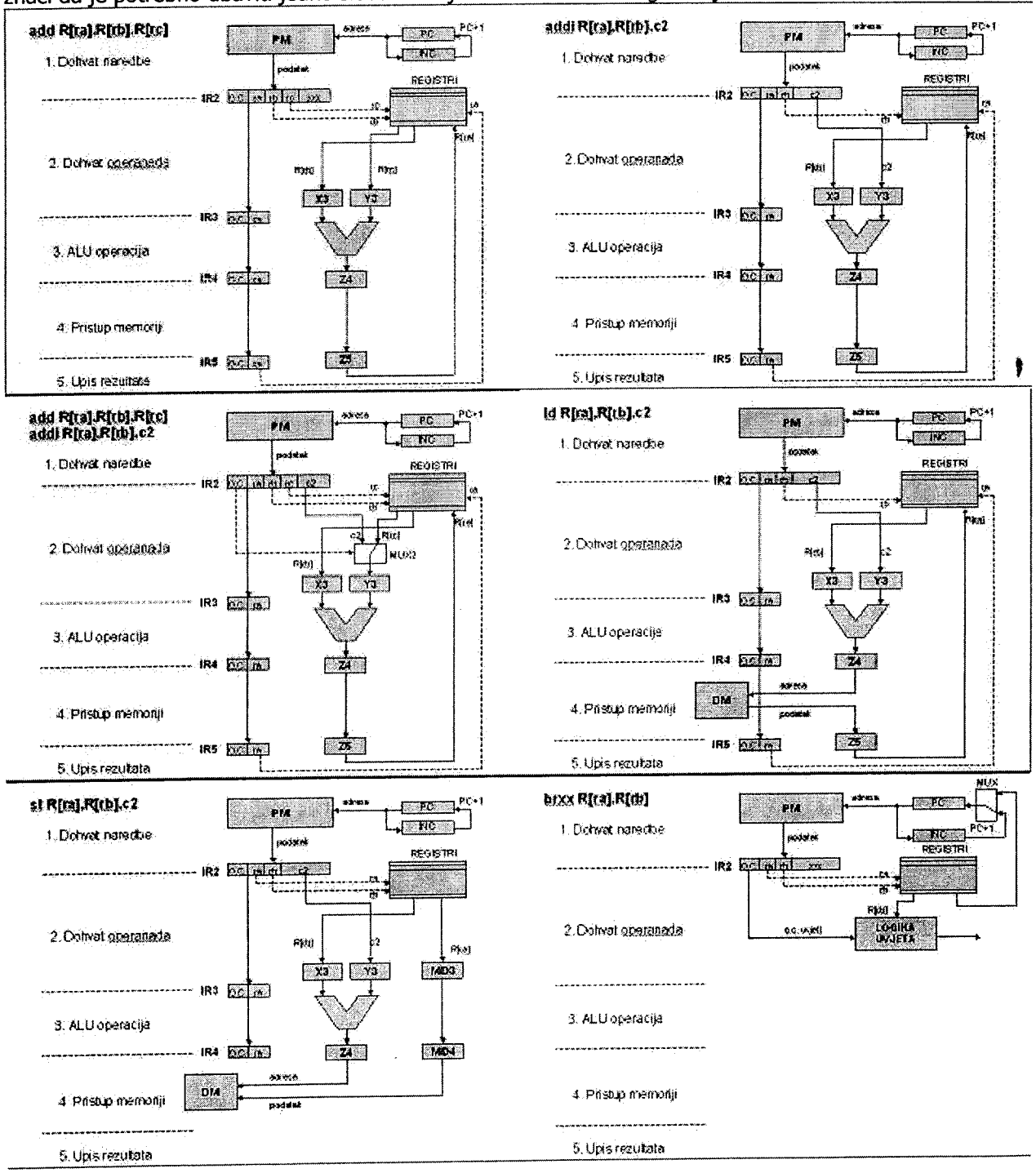


Dvo- i tro-sabirnička SRC arhitektura: ukoliko se poveća broj sabirnica, odnosno broj međuveza, moguće je istovremeno prenositi više podataka. Tako je broj i konfiguracija sabirnica jedan od odlučujućih faktora u povećanju performansi procesora. S druge strane povećanje broja sabirnice, neovisno o razini 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. Sustav se poboljšava uvođenjem **treće sabirnice** u arhitekturu SRC-a jer se tako mogu na ALU istovremeno dovesti oba operanda ali i rezultat upisati u spremnik. Naredba LOAD je smanjena sa 8 na 4 koraka. Ukoliko povećamo period takta za 10%, dobijemo ubrzanje od 82%! Prvo smo pretpostavili da je potreban jedan ciklus za pristup memoriji a neka je nova pretpostavka da su potrebna 3 ciklusa (2 su onda potrebna za čekanje). Zatim neka je 20% naredbi koje upisuju sadržaj iz memorije u spremnik LOAD, te da kod upisa u memoriju nema čekanja. Budući da svaka nareda mora izvesti dohvat naredbe iz memorije, te ako računamo još onih 20%, to znači ubrzanje koje iznosi 48% što je svejedno značajno povećanje perfomansi.

3. CJEVOVOD

Budućući da izvođenje jedne naredbe može ovisiti o završetku druge, rješenje je zaustavljanje cjevovoda a ta se ovisnost među naredbama može detektirati tijekom pripreme programa za izvođenje. Glavnoj memoriji mora se pristupati u jednom taktu a to je moguće realizirati skupom memorijom. Uobičajno rješenje je brza međumemorija "Cache". Naredbe i podaci moraju se pohraniti u odvojenim memorijama. Sve naredbe moraju se uklopiti u zajedničku cjevovodnu strukturu a koristi se 5-razinski cjevovod - (1) Dohvat naredbe, (2) Dekodiranje i dohvat operanada, (3) ALU operacija, (4) Pristup podatkovnoj memoriji,

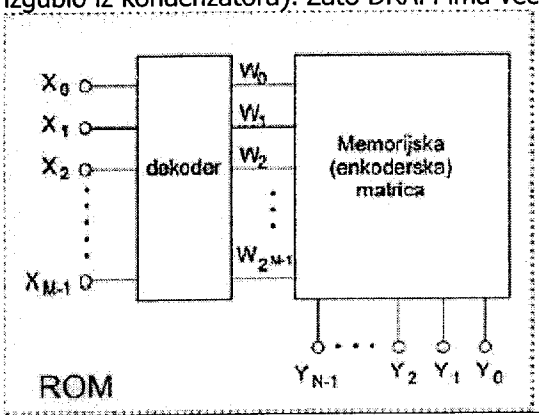
(5) Upis u spremnik. Potrebno je naredbe load/store, ALU, i branch uklopiti u ovu shemu. Što se tiče grananja, nova vrijednost programskog brojila poznata je u 2 koraku a ne u prvom dok se samo grananje u potprogram upisuje u spremnik u 5. koraku Nema ALU operacija i pristupa memoriji. Naredbe koje se nalaze u cjevovodu izvode se paralelno. Međuovisnost naredbi može uzrokovati probleme, dobro poznate u paralelnom procesiranju Temeljni problem je da naredba ovisi o rezultatu predhodne naredbe a koja nije okončana. Dvije su kategorije problema: problemi s podacima: nepravilna uporaba zastarjelih ili novih podataka te problemi grananja: dohvat krive naredbe prije promjene sadržaja Pca nakon uspješnog grananja. Analizirajmo sljedeći programski odsječak : 100: add r0, r2, r4 i 104: sub r3,r0,r1. Naredba za zbrajanje ADD zbraja sadržaje spremnika r2 i r4 i rezultat upisuje u spremnik r0 dok se u sljedećoj naredbi rezultatu prethodnog zbrajanja oduzima sadržaj spremnika r1. Rezultat operacije se upisuje u odredišni spremnik tek u petom koraku dok se operandi dohvaćaju već u drugom. Tako naredba za SUB pristupa spremniku r0 prije nego je prethodna naredba za zbrajanje u njega upisala rezultat i tu govorimo o opasnosti primjene cjevovoda vezanog uz pristup podacima. Da bi se to riješilo, tj. da bi se pravovremeno proslijedilo potreban podatak sljedećoj naredbi, koristimo FORWARDING HARDWARE konstrukciju sklopovlja. prvi je korak vezan uz naredbu (korak kada je rezultat normalno dostupan i korak kada je rezultat najprije dostupan) a drugi uz operande (korak kada je operand normalno potreban i korak kada je operand najkasnije potreban). 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 ubaviti jedno slobodno mjesto iza naredbe za grananje.

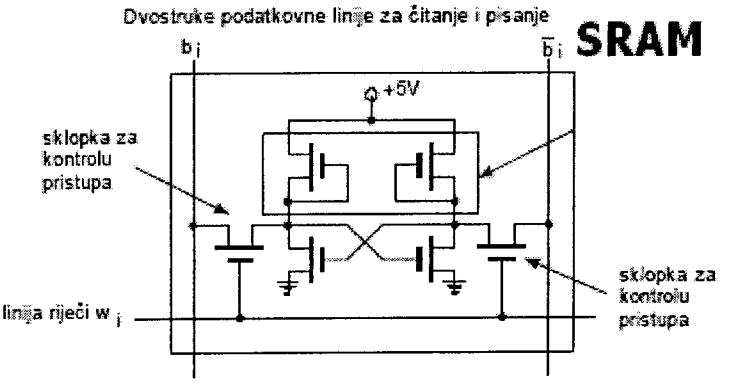


4. MEMORIJA

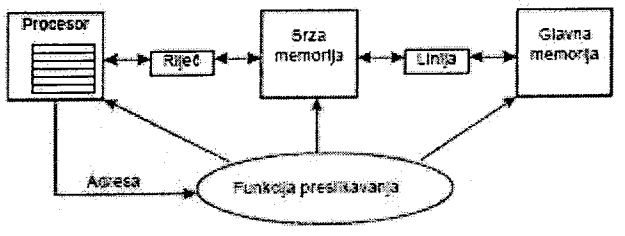
RAM (Random Access Memory): memorijskim elementima RAM pristupa proizvoljnim rasporedom i uvijek s istim vremenom pristupa. Prema konvenciji ovaj termin rezerviran je za poluvodičku memoriju u koju se može pisati te iz koje se može čitati. **ROM** je predprogramirana poluvodička memorija iz koje se može *samo čitati* što opisuje i njen naziv: *Read Only Memory*. Ćelijama ROMa (i RAMa) može se pristupati proizvoljnim rasporedom i uvijek s istim vremenom pristupa. 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 ekonomicnosti (isplativosti) ROM-a riješeni su sklopovima nazvanim PROM (***Programable 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 (E**rasible **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 zahtjeva relativno dugo izlaganje komponente UV zračenju. **E^{2}PROM**: (**E**lectrically **E**rasable **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 G_1 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 osvježavat. 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. Tako jedan tranzistor i kondenzator zamjenjuju šest tranzistora. Ukoliko se kondenzator realizira pomoću tranzistora slijedi zaključak da je potrebno tri manje komponenata za realizaciju ovakve memorije. Zovedinamički jer čitanjem praznimo kondenzator pa ga je poosvježiti (upisati ponovo podatak jer smo ga čitanjem iznaboj se izgubio iz kondenzatora). Zato DRAM ima veći kapacitet ali manju brzinu od SRAM-a.





5. CACHE



Funkcija preslikavanja kod brze memorije.

Današnje se memorije sastoje od brze memorije (cache) kojoj procesor pristupa u samo jednom taktu, te sporije memorije koja je znatno većeg kapaciteta. Procesor pristupa informacijama, naredbe i podackoje su samo u brzoj memoriji. Ukoliko informacija nije u brzoj memoriji, potrebno ju je prebaciti iz sporu 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 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*), i 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 imamo 3 različite funkcije preslikavanja: asocijativno (svaka linija se iz glavne memorije može smjestiti bilo gdje u brzoj memoriji), direktno (linija se može nalaziti samo na određenom mjestu u brzoj memoriji) i asocijativno po skupinama blokova (asocijativno preslikavanje unutar skupina linija je kombinacija prethodnih dviju metoda, direktorij linija se proširuje dodatnim stupcima pa tako imamo two-way block set, four-way...).