6. AKCELERATORI

6.1. Koprocesori

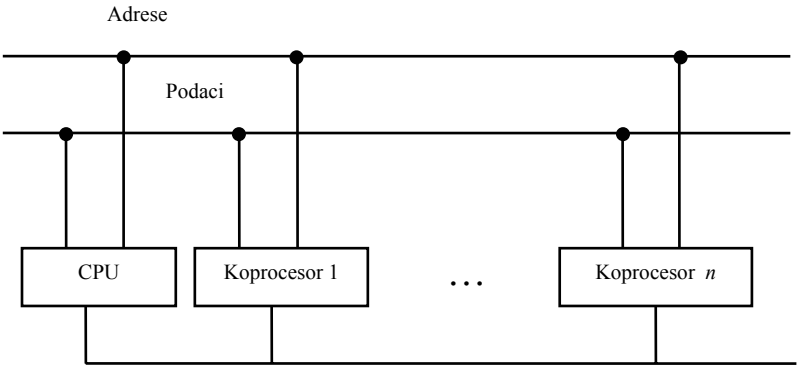
Najveći broj mikroprocesora je danas projektovan kao čip opšte namene koji se može primeniti za veliki broj različitih aplikacija. Saglasno nameni mikroprocesori imaju ugrađen skup instrukcija opšteg tipa (namene). Dizajn mikroprocesora je ograničen površinom silicijuma na kojoj je on fabrikovan, pa bi uvođenje novih instrukcija u njegov skup zadržavajući istu površinu zahtevalo preprojektovanje čipa. Preprojektovanje je skup i dugotrajan proces i ne mora da znači da će preduzeti korak biti uspešan, tj. da ćemo zadržavajući istu površinu čipa uspeti da uvedemo nove funkcije. Da bi se rešio ovaj problem na zadovoljavajući način, uvode se koprocesori. Ugradnjom koprocesora u sistem se povećava skup naredbi glavnog procesora. Obično su koprocesori zamišljeni da se koriste kao posebni čipovi, a karakterišu ih sledeće osobine:

- to je deo hardvera koji je namenjen da obavlja određenu klasu operacija pri čemu su performanse koprocesora za tu klasu operacija značajno superiornije u odnosu na one koje važe za CPU-ove opšte namene.
- koprocesor ne može izolovano da obavlja svoju operaciju; potrebno je prisustvo od najmanje još jednog procesora. Ovaj drugi procesor, obično je CPU, treba da je sposoban da izvršava svoje operacije izolovano, bez asistencije bilo kog drugog procesora.
- koprocesor poseduje neki oblik lokalnog memorisanja operanada i/ili rezultata (na primer, u obliku skupa registara). Za bilo koju koprocesorsku operaciju, u njegovoj lokalnoj memoriji pamtiće se rezultat i/ili jedan ili veći broj operanada. Zahtevi za implementaciju lokalne memorije čine da koprocesori budu različiti od jedinica koje obavljaju specifične funkcije (SFU - Special Fuction Unit). SFU može da manipuliše samo nad operandima lociranim u CPU-ovim registrima, pa se kao takva može smatrati kao ALU-ovo proširenje CPU-a.
- Svaki koprocesor implementira specifični skup instrukcija; objektni program predstavlja mešavinu CPU-ovih i koprocesorskih instrukcija. Pribavljanje CPU-ovih i koprocesorskih instrukcija iz jedinstvenog skupa instrukcija ukazuje na to da samo jedan od procesora (obično CPU) generiše adrese za pribavljanje instrukcija i "vodi brigu" o upravljanju prenosom podataka na magistrali. Zahtevi za egzistenciju jedinstvenog tipa instrukcija (single-thread) čine da koprocesori budu različiti u odnosu na *pridodate procesore* (attached processors) koji obično imaju implementiran značajan iznos instrukcija i memoriju za podatke. Vektorski procesori, koji mogu izvršavati svoje operacije skoro potpuno nezavisno od host CPU-a, su tipičan primer pridodatih procesora. U daljem tekstu ograničićemo se samo na analizu rada koprocesora zbog njihove značajne uloge u odnosu na tekuće mikroprocesorske arhitekture.

Koprocesori se ne mogu kategorisati samo prema klasi operacija koje oni izvršavaju (na primer, pokretni zarez u odnosu na decimalnu aritmetiku). Oni se takođe značajno razlikuju po načinu sprezanja sa CPU-om i po autonomnosti u radu.

6.1.1. Sprezanje koprocesora

Na implementacionom nivou, koprocesor se može integrisati u sistem kao onaj prikazan na slici 6.1.



Sl. 6.1. Primer integrisanja koprocesora u sistem.

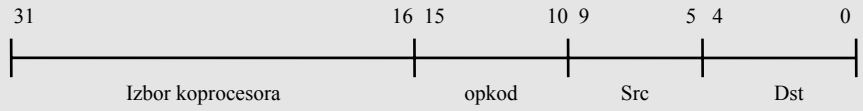
Svaki koprocesor u sistemu se može povezati na adresnu magistralu i magistralu podataka kao, i na specijalnu magistralu koja povezuje CPU sa svim koprocesorima. U ekstremnom slučaju, svi koprocesori su povezani samo na koprocesorskoj magistrali. Ovakva izvedba zahteva da se informacija prenosi preko CPU-a, a ima za posledicu da koprocesori kao čipovi nemaju veliki broj pinova. U drugom ekstremnom slučaju, koprocesori su povezani na adresnu magistralu i magistralu za podatke ali, takođe, i na koprocesorsku magistralu. Ovakvim izvođenjem ostvaruje se veoma brz prenos podataka.

Na arhitekturnom nivou, razlika se pravi u zavisnosti od vidljivosti skupa instrukcija koprocesora.

- **Arhitektura koprocesora može biti nevidljiva (transparentna) CPU-u**. To je slučaj kada se komunikacija tipa CPU↔koprocesor obavlja pomoću podskupa CPU-ovih instrukcija. Na primer, LOAD i STORE instrukcije se mogu koristiti od strane CPU-a za prenos operanada ka/iz koprocesora kao i za specificiranje operacije koja se izvršava (na primer, upis u upravljački registar koprocesora), kao i za ispitivanje statusa koprocesora (čitanjem statusnog registra). Ovakav pristup uslovljava preslikavanje registra koprocesora u memorijski prostor CPU-a.
- **Arhitektura koprocesora je delimično vidljiva CPU-u**, a to obično znači da CPU može identifikovati takvu instrukciju kao koprocesorsku; CPU ne može identifikovati pojedini tip operacije koju treba da obavi, a koja treba da se specificira u nekom drugom delu (da to ne bude opkod prostor) instrukcije. Ovakvu instrukciju možemo smatrati *osnovnom* (*generic*) *koprocesorskom instrukcijom*. Prednost ovakvog pristupa sastoji se u sledećem. I pored toga što je CPU svestan činjenice da koprocesorska instrukcija mora da se izvrši, a to može zahtevati prenos informacije preko koprocesorske magistrale, format instrukcije i operacije koprocesora su transparentne CPU-u (sa izuzetkom polja koje specificira osnovnu koprocesorsku operaciju). Ovim je ostvarena mogućnost dodavanja (ugrađivanja u sistem) koprocesora, i kasnije u toku eksploatacije sistema, a ne samo u toku fabrikacije, sa proizvoljnom funkcionalnošću (uočimo da i kod prvog pristupa postoji ovakva mogućnost).
- **Arhitektura koprocesora je kompletno vidljiva CPU-u**. U ovom slučaju, sve koprocesorske instrukcije pripadaju skupu instrukcija CPU-a, i ne mogu se kasnije dodavati nove instrukcije a da se ne izvrši modifikacija CPU-a. Kako su sve koprocesorke instrukcije poznate CPU-u, CPU može automatski da obavi izvršenje određenih delova koprocesorskih instrukcija kao što je izračunavanje adrese operanada i pribavljanje/smeštanje (memorisanje) operanada.

Primer 6.1:

FP koprocesor Weitek 1167 sadrži registarsko polje od 32×32-bitnih registara (R0 do R31). Weitek 1167 se primenjuje kao koprocesor za 32-bitni Intelov mikroprocesor iAPX 80306, a koristi memorijsko preslikavanje. Procesor je povezan na adresne linije i linije za podatke CPU-a (iAPX 80386), a preslikavanje memorije se obavlja na način prikazan na slici 6.2.



Sl. 6.2.

32-bitna adresa se interpretira od strane koprocesora na sledeći način: MS 16 adresnih bitova se koriste za izbor pojedinog koprocesora, a sledećih 6 bitova određuje tip operacije koju procesor obavlja; Src poljem specificira se jedan od 32-bitnih lokalnih registara kao izvorni operand (kada je Src=0 linije za podatke sadrže vrednost izvornog operanda, a ne sadržaj lokalnog registra (R0)); Dst poljem specificira se lokalni registar, koji se javlja kao drugi izvorni i odredišni operand (kada je Dst=0, podatak se postavlja na linije za podatke a ne u R0). FP izraz A:=B*C+D se može izvršiti na sledeći način (mnemonik posle MOV specificira operaciju koja se obavlja od strane koprocesora):

```
MOV.WRITEF    B,R1    ; upiši B u koprocesorski registar R1
MOV.MULF      C,R1    ; R1:=(R1)*C
MOV.ADDF      D,R1    ; R1:=(R1)+D
MOV.READF     R1,A    ; A:=(R1)
```

6.1.2. Autonomnost koprocesora

Koprocesor prima svoje instrukcije iz (istog) instrukcionog niza kao i CPU. CPU je odgovorna za generisanje adresa instrukcija, a to čini da koprocesor bude zavisan od CPU-a. Koprocesor može biti autonoman u zavisnosti od načina kako se:

- pribavlja instrukcija,

- izračunava adresa operanda, i
- pribavlja/memoriše podatak.

Pribavljanje instrukcija od strane koprocesora

Koprocesori mogu pribavljati svoje instrukcije putem praćenja instrukcija (instruction tracing) ili raspodelom instrukcija (instruction dispatching). Kod praćenja instrukcija, CPU generiše adrese instrukcija, a svi koprocesori ispituju linije za podatke sa ciljem da prepoznaju i pribave instrukciju koja je namenjena njima. Moguće je da više od jednog koprocesora reaguje na jedinstvenu (jednu) instrukciju - na primer kada treba da se obavi prenos podataka između dva koprocesora. Ovaj pristup zahteva da arhitektura koprocesora bude, u najmanju ruku, delimično vidljiva CPU-u (jer CPU mora da ignoriše one instrukcije koje nisu namenjene njemu) i da obezbedi visok stepen autonomnosti rada koprocesora.

Kod raspodele instrukcija, CPU je jedina jedinica koja ispituje instrukcije. Nakon što CPU pribavi instrukciju za koprocesor on (CPU) je predaje odgovarajućem koprocesoru. Kada je arhitektura koprocesora nevidljiva CPU-u, koristi se tip instrukcije "move memory", a koprocesor mora biti povezan na adresne linije i linije za podatke. Kada je arhitektura koprocesora (delimično) vidljiva CPU-u, CPU se uključuje u izvršenje određenih delova koprocesorske instrukcije i može menjati informaciju sa koprocesorom preko koprocesorske magistrale.

Određivanje adresa i pribavljanje/memorisanje podataka za koprocesor

Analogno CPU-ovim operandima, koprocesorski operandi se mogu specificirati kao neposredni operandi (deo su instrukcije) i oni se mogu nalaziti u lokalnim koprocesorskim registrima ili u nekim memorijskim lokacijama. Kada se nalaze u memorijskim lokacijama potrebno je da se na neki način odredi kako će se vršiti izračunavanje adresa (na primer, u kom slučaju se koristi indeksno adresiranje). Kod ovakve situacije, u zavisnosti od autonomije koprocesora, postoji nekoliko alternativa:

- Koprocesor obavlja izračunavanje adrese operanda i vrši pribavljanje/memorisanje podataka. Koprocesor mora biti povezan na adresne linije i linije za podatke i mora da poseduje lokalni skup adresnih (indeks) registara. Ovaj način rada obezbeđuje visok stepen autonomnosti po ceni ugradnje u koprocesor kompletne jedinice za pribavljanje operanada.
- Koprocesor je taj koji sam određuje adrese operanda. Izračunate (određene) adrese moraju se predati CPU-u, koji zatim pribavlja/memoriše podatke. Ovaj pristup zahteva da koprocesor sadrži najveći deo hardvera koji pripada jedinici za pribavljanje operanda. Znatno veliki iznos komunikacije CPU↔koprocesor postoji u ovom slučaju, pa je i to razlog što ova alternativa nije do sada komercijalno implementirana.
- CPU određuje adrese operanada. Koprocesor pribavlja/memoriše podatke, koristeći linije za podatke (u sistemu). Ovo zahteva da su formati operanda koprocesorskih instrukcija vidljivi CPU-u, ili da koprocesor mora predati informaciju CPU-u na način kao i kad bi on sam određivao adrese operanada. Kako se kod CPU-ova standardno ugrađuje jedinica za pribavljanje operanda, a takođe ugrađuje i skup adresnih registara, ova alternativa predstavlja rešenje koje se često koristi.
- CPU određuje adrese operanada i vrši pribavljanje/memorisanje podataka. Ovo rešenje je slično prethodnoj alternativi ali, pored svega, magistrala koprocesora je ta koja se koristi za prenos podataka koprocesor↔CPU.

6.1.3. Konkurentnost i sinhronizacija

Kada je procesor u stanju da izvršava instrukcije konkurentno sa instrukcijama koje izvršava CPU, za interfejs CPU-koprocesor kažemo da je *asinhroni*. Kada se koprocesorske instrukcije mogu izvršavati (jedino) nekonkurentno sa izvršenjem CPU-ovih instrukcija (tj. CPU mora da čeka na izvršenje sledeće instrukcije sve dok se koprocesorska instrukcija ne završi), za interfejs CPU-koprocesor kažemo da je *sinhroni*. Potpuna konkurentnost nije moguća jer, u sistemu gde su instalirani koprocesori, može se izvršavati samo jedan (jedinstveni) tok instrukcija pa je na taj način olakšana i obrada izuzetka. Sinhronizacija sa CPU-om se zbog toga zahteva kako u toku izvršenja instrukcija tako i u toku obrade izuzetka.

Sinhronizacija u toku izvršenja koprocesorskih instrukcija

Svaki put kada CPU želi da koristi novu koprocesorsku instrukciju, neophodno je izvršiti sinhronizaciju. Isti tip sinhronizacije se zahteva i kada treba da se obavi prenos operanada ka koprocesoru i rezultati prebace u glavnu meoriju (usvajamo da je autonomnost koprocesora takva da on ne može da izračuna (odredi) njegove operande i/ili pribavi/memoriše podatke), a mora da se obavi prenos podataka između CPU-a i koprocesora (na primer, predaja "condition code" koprocesoru). Sinhronizacija se može izvesti na dva načina:

1. Zauzetost testiranjem signala čekanja (busy waiting). Koprocesor drži CPU u stanju čekanja sve dok koprocesor ne završi svoju tekuću instrukciju.
2. Produžavanje aktivnosti CPU-a uz testiranje signala čekanja (non-busy waiting). CPU proverava da li je koprocesor zauzet, ako je, on može izvršiti drugu instrukciju (a ne da čeka) i da ponovo kasnije proveri da li je koprocesor zauzet. Ovakav način rada dozvoljava viši stepen konkurentnosti, ali je teži za implementaciju sa tačke gledišta programske realizacije.

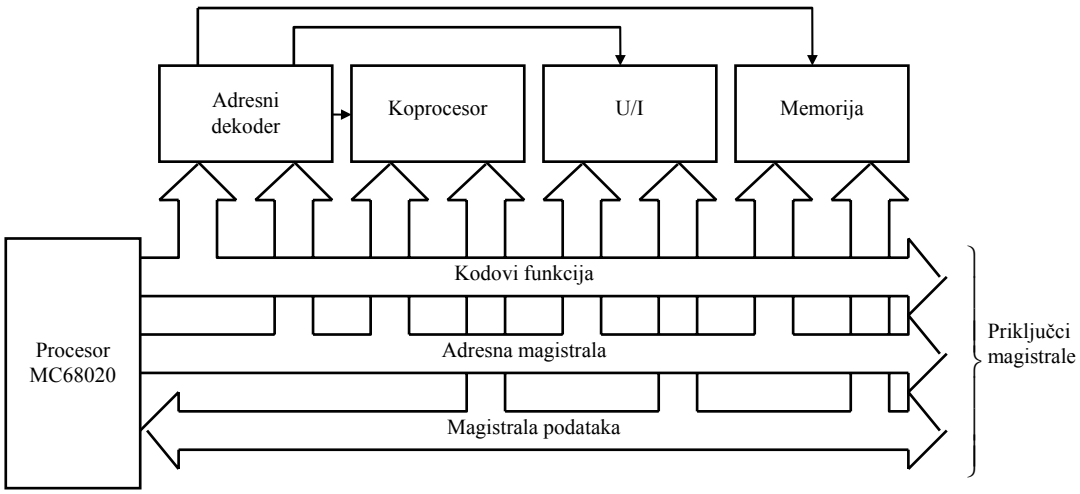
Treba naglasiti da se koprocesori mogu implementirati na takav način da se njihovim lokalnim registrima (koji ne učestvuju u izvršenju tekuće instrukcije) može vršiti upis/čitanje od strane CPU-a.

Sinhronizacija kod izuzetaka

U toku izvršenja koprocesorskih instrukcija mogu se javiti izuzeci, kao što su deljenje nulom ili greška u parnosti podataka. Ako koprocesor "ne može da izađe na kraj" sa ovim izuzecima njemu će trebati pomoć (asistencija) CPU-a. Zbog toga je neophodno uvesti čekanje. Čekanje može biti tipa: "busy" (koprocesor čeka sve dok CPU ne proveri status koprocesora) ili "non-busy" (koprocesor signalizira CPU-u izuzetak preko aktiviranja signala na liniji zahtev za prekid).

6.1.4. Sprezanje MC68020 sa koprocesorom

Na slici 6.3 prikazana je uprošćena šema povezivanja koprocesora na CPU-ovoj magistrali. Koprocesori se obično realizuju kao čipovi specifične namene među kojima se najčešće koriste FPU-ovi. Oni izvršavaju instrukcije koje predstavljaju proširenje skupa naredbi mikroprocesora. Za MC68020 komercijalno su dostupni sledeći koprocesorski čipovi MC68851 PMMU (o načinu rada ovog čipa smo već govorili) i FP koprocesori MC68881 i MC68882.

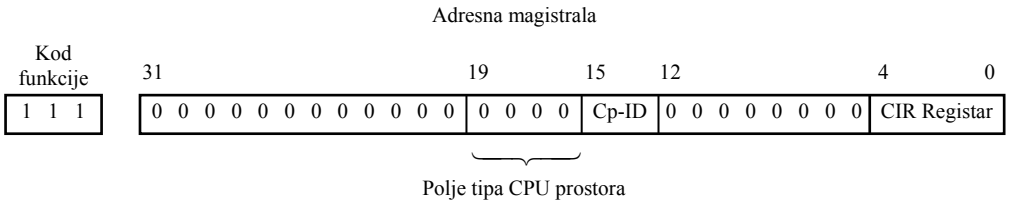


Sl. 6.3. Povezivanje koprocesora na magistralu CPU.

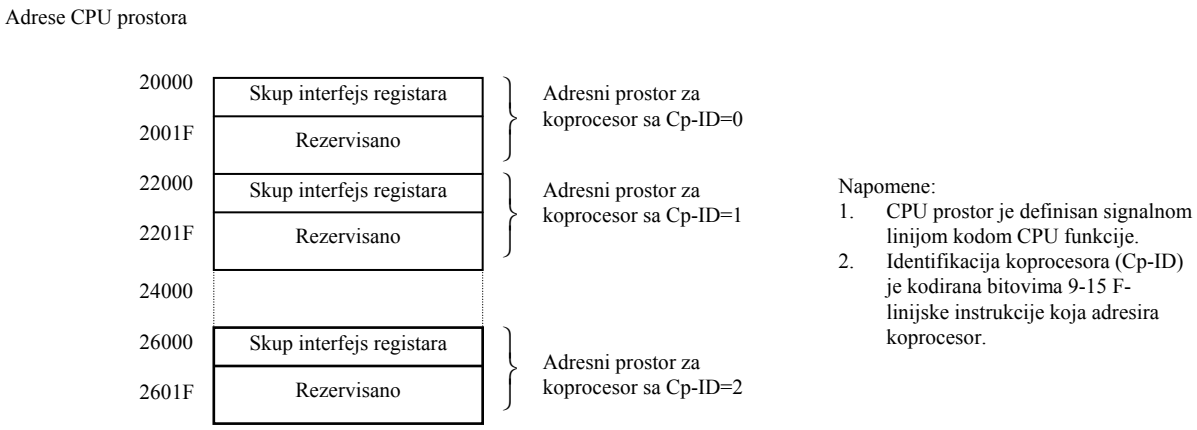
CPU i koprocesor komuniciraju preko registara koji su deo koprocesora. Da bi se inicirala koprocesorska operacija potrebno je da se izvrši koprocesorska instrukcija. CPU predaje instrukciju koprocesoru putem adresiranja koprocesorskog komandnog registra, koji se nalazi u CPU-ovom adresnom prostoru, a za pristup se koristi adresni format prikazan na slici 6.4.

Na slici 6.4b prikazane su moguće adrese za osam koprocesora koliko je moguće maksimalno povezati na sistem zasnovan na MC68020. Identifikacija koprocesora (CP-ID) čija je vrednost 000 je rezervisana za MC68851, (CP-ID) čija je vrednost 001 rezervisana za FPU. Ostale vrednosti od (002-007) se mogu koristiti za identifikaciju koprocesora specijalne namene.

Nakon što koprocesor prihvati CPU-ovu komandu, on se odaziva postavljanjem bit oblika, koji zovemo kod odziva, u svom registru namenjenom za čuvanje odgovora. CPU nakon toga čita odgovor i obavlja odgovarajuću uslugu.



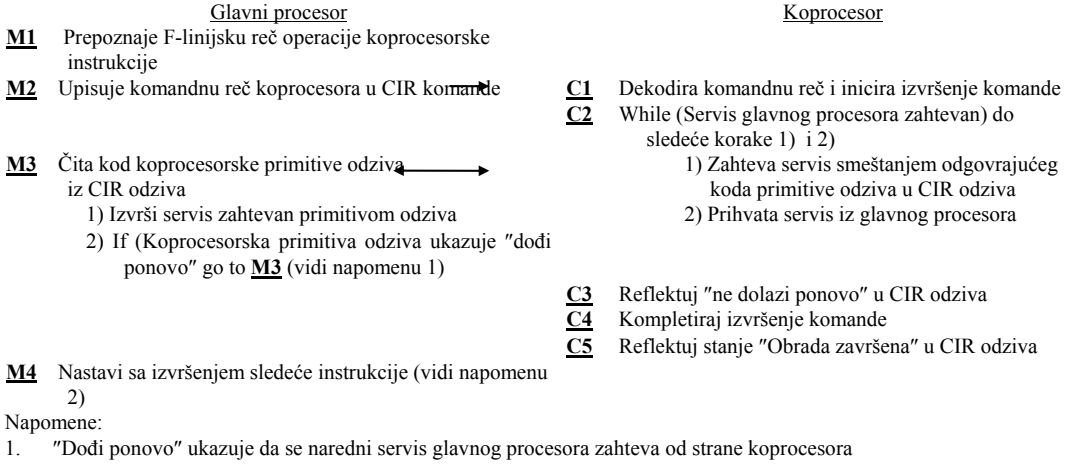
(a)



(b)

Sl. 6.4. Adresni format koprocesorske instrukcije.

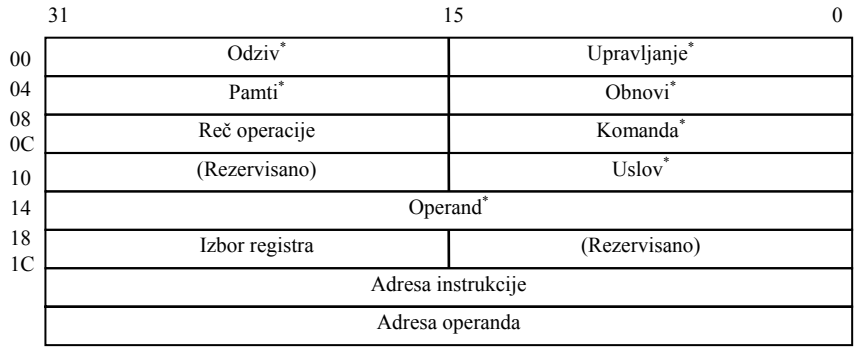
Opšta sekvenca CPU-ove i koprocesorske komunikacije prikazana je na slici 6.5.



Sl. 6.5. Opšta sekvenca komunikacije CPU-a i koprocesora.

Sekvenca počinje kada se u toku programa izvršava F-linijska instrukcija. Kada se F-linijska instrukcija prepozna od strane CPU-a instrukcija se ne dekodira nego se predaje koprocesorskom komandnom registru. Komunikacija između CPU-a i koprocesora produžava pri čemu se na dalje koristi odgovarajući koprocesorski interfejs registar (CIR). U toku izvršenja instrukcije, koprocesor može zahtevati od strane CPU-a različita opsluživanja, a on to čini postavljanjem odgovarajućeg koda u CIR namenjenom za odziv. Ovi kodovi, nazvani primitive odziva koprocesora, predstavljaju status ili zahtev izdat od strane koprocesora. Kada koprocesor završi svoju instrukciju, on ukazuje da je završio preko registra za odgovor nakon čega CPU može da produži sa izvršenjem naredne instrukcije u programu.

Adresna struktura koprocesorskog registarskog skupa prikazana je na slici 6.6.



Napomena: Registri označeni zvezdicom (*) se zahtevaju za svaki koprocesor.

Sl. 6.6. Adresna struktura skupa registara koprocersora.

Registri se adresiraju pomoću njihovog ofseta u odnosu na baznu adresu u CPU-om prostoru. Bazna adresa u CPU-ovom prostoru određena je bit poljem (bitovi 16-19) sa slike 6.4a). Pojedini CIR se bira bitovima 0-4. CPU automatski bira odgovarajući registar kada se izvršava jedna od koprocesorskih instrukcija. U koprocesorima specifične namene ne moraju da budu ugrađeni svi registri prikazani na slici 6.6 (neophodni su samo oni koji su označeni zvezdicom).

Sve instrukcije koprocesora se mogu svrstati u jednu od sledeće četiri kategorije:

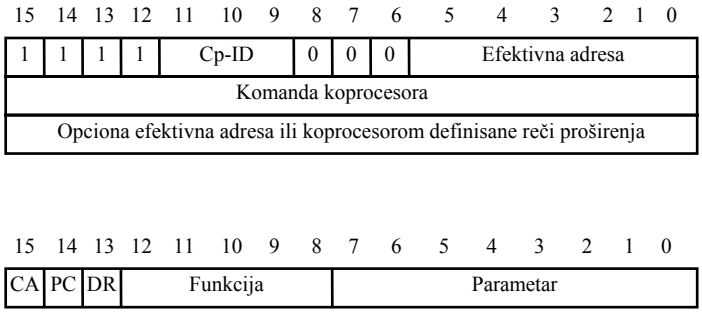
- opšte,
- uslovne,
- koriste se za memorisanje konteksta,
- koriste se za obnavljanje stanja konteksta.

Nabrojane instrukcije se razlikuju u zavisnosti od tipa operacije koju obavlja koprocesor, ali one ne određuju specifičnost rezultata jer on zavisi od tipa koprocesora. Kategorije instrukcija određuju kojim se CIR-ovima pristupa od strane CPU-a i koprocesora. U Tabeli 6.1 prikazana je lista specifičnih koprocesorskih instrukcija koje su deo skupa instrukcija mikroprocesora MC68020.

Tab. 6.1. Koprocesorske instrukcije.

Kategorija koprocesorske instrukcije	Tipična upotreba
Opšta	
cpGEN	Za iniciranje obrade podataka ili druge operacije definisane za koprocesor
Uslovne	
cpBcc cpDBcc cpScc cpTRAPcc	Za omogućavanje programskog upravljanja zasnovanog na koprocesorskim instrukcijama i koprocesorskom odzivu
Pamćenje	
cpSAVE	Za pamćenje stanja koprocesora
Obnavljanje	
cpRESTORE	Za obnavljanje stanja koprocesora

Primitive koje se odnose na odziv koprocesora kodiraju se pomoću 16-bitne reči koja se čita od strane CPU-a iz koprocesorskog registra za odziv. Kao što je prikazano na slici 6.8 primitiva odziva se deli na dva polja koja odrejuju specifičnu akciju, i tri bita (13-15) koji upravljaju drugim operacijama. Koprocesor mora kao odgovor da preda odgovarajuću primitivu za svaku koprocesorsku komandu koju CPU izda. Pored ostalog, CA (Come and Call Again) bit se postavlja na {1} od strane koprocesora, što uslovljava da CPU čita registar odziva veći broj puta u toku izvršenja jedne koprocesorske instrukcije. Ako je PC (Program Counter) bit jednak {1}, CPU predaje sadržaj PC-a koprocesoru. Bit DR (Direction) određuje smer prenosa operanda između CPU-a i koprocesora.



Sl. 6.8. Format opšte koprocesorske instrukcije i primitiva koprocesorskog odziva.

Deo mogućih odziva koprocesora (ne svi odzivi) prikazani su u Tabeli 6.2.

Tab. 6.2. Primeri koprocesorskih odziva.

Odziv	Upotreba ili uslov
STATUS	
Busy	CPU treba da nastavi sa izdavanjem koprocesorske komande dok se koprocesor ne oslobodi
Null	Stanje koprocesora se predaje CPU
REQUEST	
Izračunava i prenosi efektivnu adresu ili operand koprocesoru	Koprocesor od CPU-a zathteva adresu ili operand
Prenosi koprocesorski operand u memoriju	Koprocesor ima vrednost operanda koju CPU treba da smesti u memoriju.
Prenosti sadržaj registra između CPU i koprocesora	Razmena operanada ili adresa između skupa registara CPU-a i koprocesora.
Prihvata izuzetak	Zahtev za obradu CPU izuzetka

Koprocesor ima tri primitive koje se odnose na odziv a koje uzrokuju obradu izuzetaka od strane CPU-a. Ove primitive se koriste kada koprocesor detektuje neki uslov kojeg ne može sam rešiti. Pored ostalog koprocesor može detektovati nekorektni protokol, ilegalnu komandu ili specifičnu grešku kod obrade podataka. Uzrok greške i obrada izuzetaka koja se preuzima od strane CPU-a zavisi od tipa koprocesora. Za MC68851 i MC68881 specifični vektori izuzetaka su definisani u vektorskoj tabeli.

FPU MC68881 i MC68882

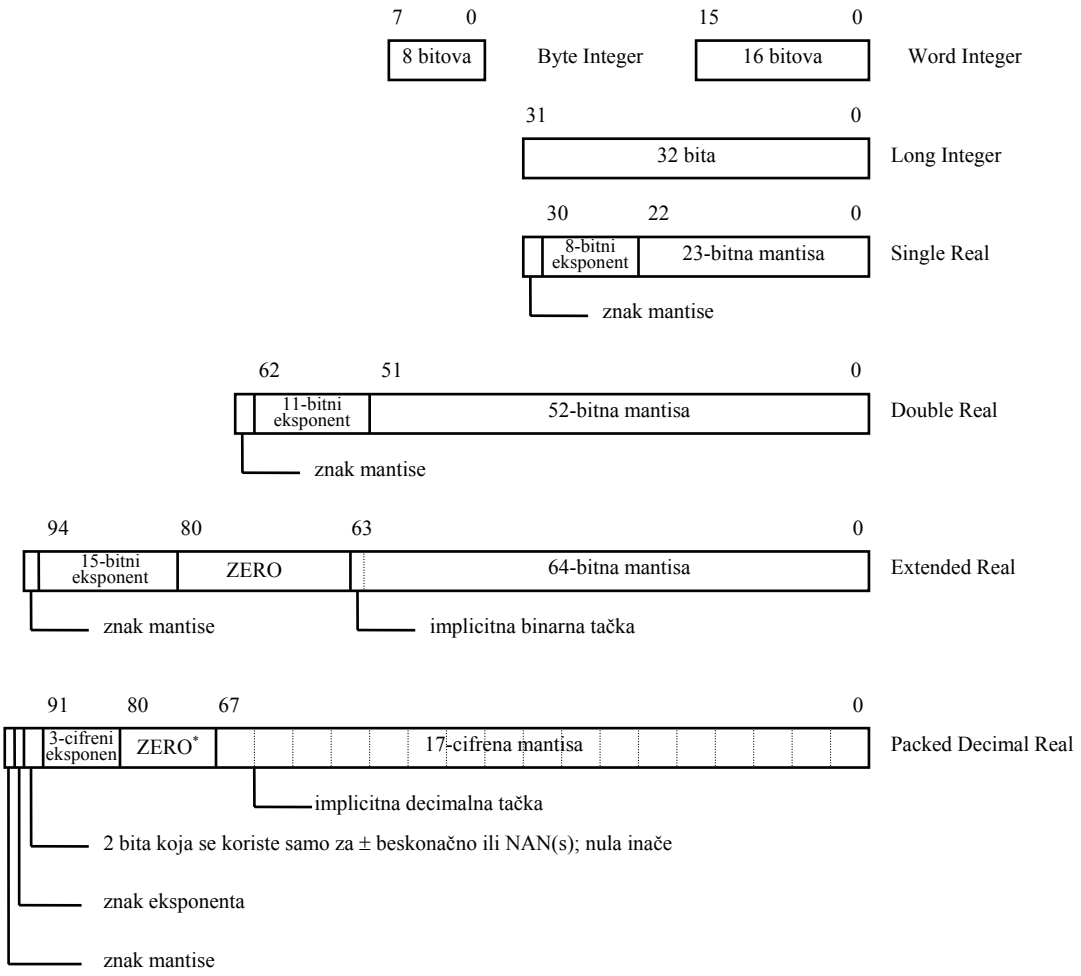
MC68881 i MC68882 su koprocesori koji obavljaju matematičke operacije. U daljem tekstu obradićemo detaljnije programski model koprocesora MC68881 (MC68882 ima identičan programski model). Oba koprocesora koriste IEEE format za rad sa FP brojevima.

Kao što je prikazano na slici 6.9, MC68881 može manipulisati sledećim tipovima podataka:

a) (B) - bajt celobrojna vrednost
b) (W) - reč celobrojna vrenost
c) (L) - duga reč celobrojna vrednost
d) (S) - realna u običnoj preciznosti
e) (D) - realna u duploj preciznosti
f) (X) - realna u proširenoj preciznosti
g) (P) - realna kao pakovani decimalni niz

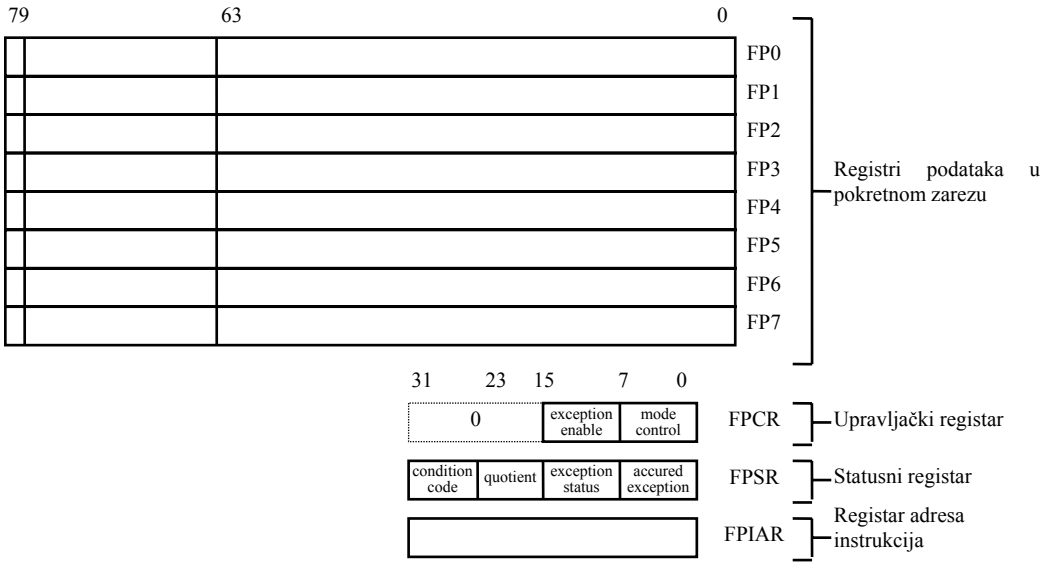
Slovo u zagradi koje prati tip podatka predstavlja sufiks koji se dodaje instrukciji u asemblerskom obliku.

Sa programske tačke gledišta registarski skup MC68020/MC68881 čine kombinacija registara D0-D7 i A0-A7 (pripadaju MC68020) i registri prikazani na slici 6.10 koji pripadaju FPU-u MC68881 .



*Ukoliko se javi prekoračenje kod konverzije binarnih u decimalne

Sl. 6.9. Formati tipova podataka kod MC68881.



Sl. 6.10. Registri MC68881.

U registrima FP0-FP7 se čuvaju vrednosti u proširenoj preciznosti koje se mogu konvertovati u druge formate kada se brojevi prenose iz MC68881. Ako se promena formata vrši u FPU, vrednost se prvo konvertuje u proširenu preciznost, a zatim se smešta u jedan od FPU-ovih registara.

Upravljački, statusni i instrukcioni adresni registri se koriste za upravljanje ili nadgledanje rada FPU-a. Jedan bajt u upravljačkom registru se koristi da dozvoli ili zabrani obradu izuzetka kada se javi određeni tip greške. "Mode control" bajt omogućava programeru da odabere metod numeričkog zaokruživanja rezultata. Četvorobajtni statusni registar sadrži bitove koji ukazuju na rezultat prenosa podataka ili matematičke operacije. Različiti bitovi ukazuju na greške kao što su premašaj, podbačaj, ili deljenje nulom. Instrukciono adresni registar čuva adresu instrukcije koja se tekuće izvršava. Koristi se od strane FP rutine za obradu izuzetka kada je potrebno da odredi adresu instrukcije koja je uzrok izuzetka.

Skup instrukcija FPU-a MC68881 se može podeliti u pet kategorija:

a) premeštanje podataka
b) diadik operacije
c) monadik operacije
d) programsko upravljanje
e) sistemsko upravljanje

Tab. 6.3. Notacija za skup instrukcija (MC68881/MC68882).

B, W, L	Ista veličina kao kod čitave MC68000 familije procesora; specificira označeni celobrojni tip podataka (dvoični komplement) dužine bajt (8 bitova), reč (16 bitova) ili duga reč (32 bita)
S	Format podatka jednostruke preciznosti u pokretnom zarezu (32 bita)
D	Format podatka dvostruke preciznosti u pokretnom zarezu (64 bita)
X	Format podatka proširene preciznosti u pokretnom zarezu (96 bitova, 16 bitova se ne koristi)
P	Format pakovanih BCD podataka u pokretnom zarezu (96 bitova, 12 bajtova)
FPm,FPn	Jedan od osam registara za podatke u pokretnom zarezu
FPcr	Jedan od tri registra u pokretnom zarezu za upravljanje sistemom (FPCR, FPSR ili FPIAR)
<EA>	Bilo koji validni adresni način rada za MC68020
K	Označeni ceo broj u dvoičnom komplementu (-64 do +17) koji specificira format broja koji se smešta u pakovanom decimalnom formatu
ccc	Indeks u MC68881 ROM-u za konstante
<lista>	Lista registara za podatke u pokretnom zarezu ili upravljačkih registara
<labela>	Relativna labela koju asembler koristi za izračunavanje razmeštaja

U Tabeli 6.3 definisane su notacije koje se koriste u tabelama instrukcija. Sufiksi B, W ,L, S, D, X i P se odnose na obim podataka. Registri FP0, ... , FP7 se koriste kao izvorišni i odredišni registri za operande; FPm se koristi kao izvorni registar a FPu kao odredišni. Oznaka "ccc" predstavlja konstantu koja se čuva u ROM memoriji FPU-a (konstante mogu biti π, e, i dr.).

Instrukcije za kopiranje podataka FPU-a date su u Tabeli 6.4.

Instrukcije FMOVE i FMOVEM (move multiple) slične su instrukcijama MOVE i MOVEM (mikroprocesora MC68020).

Tab. 6.4. Instrukcije za prenos podataka (MC68881/MC68882).

Instrukcija	Sintaksa operanda	Format operanda	Efekat
FMOVE	FPm,FPn	X	Izvor → odredište
	<EA>,FPn	B, W, L, S, D, X, P	
	FPm,<EA>	B, W, L, S, D, X, P	
	FPM,<EA>{#K}	P	
	FPM,<EA>{Dn}	P	
	<EA>,FPcr	L	
	FPcr,<EA>	L	
FMOVECR	#ccc,FPn	X	ROM konstanta → FPn
FMOVEM	<EA>,<lista>[1]	L, X	Pobrojani registri → odredište
	<EA>,Dn	X	
	<lista>[1],<EA>	L, X	Izvor → pobrojani registri
	Dn,<EA>	X	

Napomena: Lista registara može da obuhvata bilo koju kombinaciju od osam FP registara ili može da sadrži bilo koju kombinaciju tri upravljačka registra FPVR, FPSR i FPIAR. Ako se maska liste registara nalazi u registru za podatke glavnog procesora, jedino se FP registri za podatke mogu specificirati.

Primer 6.2:

Instrukcijom

FMOVE.S (A0)+,FP0

vrši se prenos podataka u jednostrukoj preciznosti iz memorijske lokacije adresirane sa A0 u FP0.

Instrukcijom FMOVECR (Move Constant ROM) punimo konstantu u FP registar. Ova konstanta se definiše pomerajem u ROM-u od $00 do $3F. Na primer, instrukcijom

FMOVECR.X #0,FP0

puni se FP0 vrednošću π, kao konstantom u proširenoj preciznosti. Ostale ofset vrednosti definisane su u katalogu firme Motorola, koji se odnosi na FPU MC68881/MC68882.

U Tabeli 6.5 prikazana je sintaksa opšte binarne instrukcije, a nakon toga je dat listing instrukcija koje pripadaju ovoj grupi. U ovu grupu spadaju instrukcije +, -, *, /, poređenje kao i neke specijalne instrukcije (sve operišu nad FP brojevima). Kod ovih operacija izvorni operand može biti lociran u FP registru, Dn registru (MC68020), ili u memoriji.

Unarne operacije koje izvršava FPU MC68881 prikazane su u Tabeli 6.6. Kod ovih matematičkih operacija prihvata se vrednost izvoprnog operanda i izračunava vrednost selektovane funkcije.

Tab. 6.5.Diadičke instrukcije (MC68881/MC68882).

(a) Format operacije.

Instrukcija	Sintaksa operanada	Format operanada	Efekat
F<dop>	<EA>,FPn	B, W, L, S, D, X, P	Fpn<funkcija>izvor → FPn
	FPm,FPn	X	

Napomena: <mop> je bilo koji specifikator diadičke instrukcije.

(b) Diadičke operacije.

Instrukcija	Funkcija
FADD	Sabiranje
FCMP	Poređenje
FDIV	Deljenje
FMOD	Ostatak po modulu
FMUL	Množenje
FREM	IEEE ostatak
FSCALE	Umnožavanje eksponenta
FSGLDIV	Deljenje u jednostrukoj preciznosti
FSGLMUL	Množenje u jednostrukoj preciznosti
FSUB	Oduzimanje

Tab. 6.6. Monadičke instrukcije (MC68881/MC68882).
(a) Format operacije.

Instrukcija	Sintaksa operanda	Format operanda	Efekat
F<mop>	<EA>,FPn	B, W, L, S, D, X, P	Izvor →funkcija→FPn
	FPm,FPn	X	
	FPn	X	FPn→funkcija→FPn

Napomena: <mop> je bilo koji specifikator monadičke instrukcije.

(b) Monadičke operacije.

Instrukcija	Funkcija
FABS	Apsolutna vrednost
FACOS	Arkus kosinus
FASIN	Arkus sinus
FATAN	Arkus tangens
FATANH	Arkus tangens hiperbolički
FCOS	Kosinus
FCOSH	Kosinus hiperbolički
FETOX	e^x
FETOXM1	e^x - 1
FGETEXP	Izdvajanje eksponenta
FGETMAN	Izdvajanje mantise
FINT	Izdvajanje celobrojnog dela
FINTRZ	Izdvajanje celobrojnog dela, zaokruženo na nulu
FLOGN	ln(x)
FLOGNP1	ln(x+1)
FLOG10	$\log_{10}(x)$
FLOG2	$\log_2(x)$
FNEG	Promena znaka
FSIN	Sinus
FSINH	Sinus hiperbolički
FSQRT	Kvadratni koren
FTAN	Tangens
FTANH	Tangens hiperbolički
FTENTOX	10^x
FTWOTOX	2^x

(c) Format dualne monadičke operacije.

Instrukcija	Sintaksa operanda	Format operanda	Efekat
FSINCOS	<EA>,FPc:FPs	B, W, L, S, D, X, P	SIN(izvor) → FPs
	FPm,FPc:FPs	X	COS(izvor) → FPc

Ove funkcije se obično koriste kao deo matematičkih bibliotečnih potprograma HLL-ova ili kao deo funkcijskih poziva u jezicima kao što je FORTRAN.

Instrukcije MC68020 i MC688881

Spisak svih instrukcija, uređen po alfabetskom redosledu, mikroprocesora MC68020 i koprocesora MC68881 prikazan je u Tabeli 6.7. i Tabeli 6.8.

Tab. 6.7. Pregled skupa instrukcija za MC68000/MC68010/MC68020.

Mnenomik	Podrazumevana veličina operanda	Operacija	Asemblerska sintaksa		Kod uslova				
					X	N	Z	V	C
ABCD	B	Saberi decimalne brojeve sa proširenjem	ABCD ABCD	Dy,Dx -(Ay),-(Ax)	*	U	*	U	*
ADD	W	Saberi binarno (nap. 1)	ADD ADD	<ea>,Dn Dn,<ea>	*	*	*	*	*
ADDA	W	Saberi adresu	ADDA	<ea>,An	-	-	-	-	-
ADDI	W	Saberi neposredno	ADDI	#<data>,<ea>	*	*	*	*	*
ADDQ	W	Saberi brzo	ADDQ	#<data>,<ea>	*	*	*	*	*
ADDX	W	Saberi sa proširenjem	ADDX ADDX	Dy,Dx -(Ay),-(Ax)	*	*	*	*	*

Mnenomik	Podrazumevana veličina operanda	Operacija	Asemblerska sintaksa	Kod uslova				
				X	N	Z	V	C
AND	W	Logičko I	AND <ea>,Dn AND Dn,<ea>	-	*	*	0	0
ANDI	W	Neposredno I	ANDI #<data>,<ea>	-	*	*	0	0
ASL, ASR	W	Aritmetičko pomeranje	ASd Dx,Dy ASd #<data>,Dy ASd <ea>	*	*	*	*	*
Bcc	W	Uslovno grananje	Bcc <labela>	-	-	-	-	-
BCHG	W	Testiraj bit i komplementiraj ga	BCHG Dn,<ea> BCHG #<data>,<ea>	-	-	*	-	-
BCLR	W	Testiraj bit i obriši ga	BCLR Dn,<ea> BCLR #<data>,<ea>	-	-	*	-	-
BFCHG	U	Komplementiraj bit polje (MC68020)	BFCHG <ea>,{<offset>:<width>}	-	*	*	0	0
BFCLR	U	Obriši bit polje (MC68020)	BFCLR <ea>,{<offset>:<width>}	-	*	*	0	0
BFEXTS	U	Izdvoj bit polje znakovno prošireno (MC68020)	BFEXTS <ea>,{<offset>:<width>},Dn	-	*	*	0	0
BFEXTU	U	Izdvoj bit polje bez znakovnog proširenja (MC68020)	BFEXTU <ea>,{<offset>:<width>},Dn	-	*	*	0	0
BFFFO	U	Nađi prvu jedinicu u bit poljuo (MC68020)	BFFFO <ea>,{<offset>:<width>},Dn	-	*	*	0	0
BFINS	U	Ubaci bit polje (MC68020)	BFINS Dn,<ea>,{<offset>:<width>}	-	*	*	0	0
BFSET	U	Postavi bit polje (MC68020)	BFSET <ea>,{<offset>:<width>}	-	*	*	0	0
BFTST	U	Testiraj bit polje (MC68020)	BFTST <ea>,{<offset>:<width>}	-	*	*	0	0
BKPT	U	Prekidna tačka (MC68020)	BKPT #<data>	-	-	-	-	-
BRA	W	Granaj se uvek	BRA <labela>	-	-	-	-	-
BSET	L	Testiraj bit i postavi ga	BSET Dn,<ea> BSET #<data>,<ea>	-	-	*	-	-
BSR	W	Granaj se na potprogram	BSR <labela>	-	-	-	-	-
BTST	L	Testiraj bit	BTST Dn,<ea> BTST #<data>,<ea>	-	-	*	-	-
CALLM	U	Poziv modula (MC68020)	CALLM #<data>,<ea>	-	-	-	-	-
CAS	W	Poredi i razmeni sa opperandom	CAS Dw,Do,<ea>	-	*	*	*	*
CAS2	W	Poredi i razmeni sa opperandom	CAS2 Dw1:Dw2,Do1:Do2,(Rz1):(Rz2)	-	*	*	*	*
CHK	W	Provera da li je sadržaj registra u okviru granica	CHK <ea>,Dn	-	*	U	U	U
CHK2	W	Provera da li je sadržaj registra u okviru granica (MC68020)	CHK2 <ea>,Rn	-	U	*	U	*
CLR	W	Obriši operand	CLR <ea>	-	0	1	0	0
CMP	W	Aritmetičko poređenje	CMP <ea>,Dn	-	*	*	*	*
CMPA	W	Aritmetičko poređenje adresa	CMPA <ea>,An	-	*	*	*	*
CMPI	W	Aritmetičko poređenje sa neposrednim operandom	CMPI #<data>,<ea>	-	*	*	*	*
CMPM	W	Poređenje memorijskih lokacija	CMPM (Ay)+,(Ax)+	-	*	*	*	*
CMP2	W	Poređenje sadržaja registra sa granicama (MC68020)	CMP2 <ea>,Rn	-	U	*	U	*
DBcc	W	Testiraj uslov, dekrementiraj i granaj se (nap. 2)	DBcc Dn,<labela>	-	-	-	-	-

Mnenomik	Podrazumevana veličina operanda	Operacija	Asemblerska sintaksa		Kod uslova				
					X	N	Z	V	C
DIVS	W	Označeno deljenje	DIVS	<ea>,Dn	-	*	*	*	0
DIVSL	L	Označeno deljenje sa odsecanjem	DIVSL	<ea>,Dr:Dq	-	*	*	*	0
DIVSU	W	Neoznačeno deljenje	DIVU	<ea>,Dn	-	*	*	*	0
DIVUL	L	Neoznačeno deljenje sa odsecanjem	DIVUL	<ea>,Dr:Dq	-	*	*	*	0
EOR	W	Isključivo logičko ILI	EOR	Dn,<ea>	-	*	*	0	0
EORI	B|W	Isključivo ILI neposredno	EORI	#<data>,<ea>	-	*	*	0	0
EXG	L	Izmena vrednosti registara	EXG	Rx,Ry	-	-	-	-	-
EXT	W	Znakovno proširenje	EXT	Dn	-	*	*	0	0
EXTB	W	Znakovno proširenje bajta (MC68020)	EXTB	Dn	-	*	*	0	0
EXTW	W	Znakovno proširenje reči (MC68020) (deo EXT instrukcije)	EXTW	Dn	-	*	*	0	0
JMP	U*	Skok	JMP	<ea>	-	-	-	-	-
JSR	U*	Skok na potprogram	JSR	<ea>	-	-	-	-	-
LEA	L	Napuni efektivnu adresu	LEA	<ea>,An	-	-	-	-	-
LINK	W	Poveži i dodeli (nap. 5)	LINK	An,#<disp>	-	-	-	-	-
LSL, LSR	W	Logičko pomeranje	LSD LSd LSd	Dx,Dy #<data>,Dy <ea>	*	*	*	0	*
MOVE	W	Kopiranje podatka sa izvora na odredište	MOVE	<ea>,<ea>	-	*	*	0	0
MOVE to SR	W	Kopiraj u statusni registar	MOVE	<ea>,SR	*	*	*	*	*
MOVE from SR	W	Kopiraj iz statusnog registra	MOVE	SR,<ea>	-	-	-	-	-
MOVE to CCR	L	Kopiraj u registar kodova uslova	MOVE	<ea>,CCR	*	*	*	*	*
MOVE from CCR	L	Kopiraj iz registra kodova uslova	MOVE	CCR,<ea>	-	-	-	-	-
MOVE USP	L	Kopiraj ukazatelj korisničkog magacina	MOVE MOVE	USP,An An,USP	-	-	-	-	-
MOVEA	W	Kopiraj adresu	MOVEA	<ea>,An	-	-	-	-	-
MOVEC	L	Kopiraj u/iz upravljačkog registra (MC68010 ili noviji) (nap. 3)	MOVEC MOVEC	Rc,Rn Rn,Rc	-	-	-	-	-
MOVEM	W	Kopiraj više registara (nap. 4)	MOVEM MOVEM	<lista registara>,<ea> <ea>,<lista registara>	-	-	-	-	-
MOVEP	W	Kopiraj podatke periferala	MOVEP MOVEP	Dx,d(Ay) d(Ay),Dx	-	-	-	-	-
MOVEQ	L	Brzo kopiranje	MOVEQ	#<data>,Dn	-	*	*	0	0
MOVES	W	Kopiraj u/iz adrese (MC68010 ili noviji)	MOVES MOVES	<ea>,Rn Rn,<ea>	-	-	-	-	-
MULS	W	Označeno množenje	MULS	<ea>,Dn	-	*	*	0	0
MULU	W	Neoznačeno množenje	MULU	<ea>,Dn	-	*	*	0	0
NBCD	B	Promena znaka decimalnog broja sa znakovnim proširenjem	NBCD	<ea>	*	U	*	U	*
NEG	W	dvoični komplement	NEG	<ea>	*	*	*	*	*
NEGX	W	Promena znaka sa proširenjem	NEGX	<ea>	*	*	*	*	*
NOP	U	Bezefektna naredba	NOP		-	-	-	-	-
NOT	W	Logičko komplementiranje	NOT	<ea>	-	*	*	0	0
OR	W	Logičko ILI	OR OR	<ea>,Dn Dn,<ea>	-	*	*	0	0

Mnenomik	Podrazumevana veličina operanda	Operacija	Asemblerska sintaksa		Kod uslova				
					X	N	Z	V	C
ORI	B|W	Logičko ILI neposredno	ORI	#<data>,<ea>	-	*	*	0	0
PACK	U	Pakovani BCD (MC68020)	PACK PACK	-(Ax),-(Ay),#<adjust> Dx,Dy,#<adjust>	-	-	-	-	-
PEA	L	Smesti efektivnu adresu u magacin	PEA	<ea>	-	-	-	-	-
RESET	U	Resetovanje spoljnog uređaja	RESET		-	-	-	-	-
ROL, ROR	W	Rotiranje bez proširenja	ROd ROd ROd	Dx,Dy #<data>,Dy <ea>	-	*	*	0	*
ROXL, ROXR	W	Rotiranje sa proširenjem	ROXd ROXd ROXd	Dx,Dy #<data>,Dy <ea>	*	*	*	0	*
RDT	U	Povratak iz potprograma sa razmeštajem (MC68010 ili noviji) (nap. 5)	RDT	#<disp>	-	-	-	-	-
RTE	U	Povratak iz izuzetka	RTE		*	*	*	*	*
RTM	U	Povratak iz modula (MC68020)	RTM	Rn	-	-	-	-	-
RTR	U	Povratak i obnavljanje kodova uslova	RTR		*	*	*	*	*
RTS	U	Povratak iz potprograma	RTS		-	-	-	-	-
SBCD	B	Oduzimanje decimalnih brojeva sa proširenje	SBCD SBCD	Dy,Dx -(Ay),-(Ax)	*	U	*	U	*
Scc	B	Postavi na osnovu slova	Scc	<ea>	-	-	-	-	-
STOP	U	Zaustavljanje izvršenja programa	STOP	#<data>	-	-	-	-	-
SUB	W	Binarno oduzimanje	SUB SUB	<ea>,Dn Dn,<ea>	*	*	*	*	*
SUBA	W	Oduzimanje adresa	SUBA	<ea>,An	-	-	-	-	-
SUBI	W	Neposredno oduzimanje	SUBI	#<data>,<ea>	*	*	*	*	*
SUBQ	W	Brzo oduzimanje	SUBQ	#<data>,<ea>	*	*	*	*	*
SUBX	W	Oduzimanje sa proširenjem	SUBX SUBX	Dy,Dx -(Ay),-(Ax)	*	*	*	*	*
SWAP	W	Razmena polovina registara	SWAP	Dn	-	*	*	0	0
TAS	B	Testiraj i postavi operand	TAS	<ea>	-	*	*	0	0
TRAPcc	U	Trap na osnovu koda uslova (MC68020)	TRAPcc TRAPcc.W,#<data> TRAPcc.L,#<data>		-	-	-	-	-
Tcc	U	Trap na osnovu koda uslova (MC68020)	Tcc		-	-	-	-	-
TDIVS	W|L	Označeno deljenje sa odsecanjem (MC68020)	TDIVS	<ea>,(Di:)Dj	-	*	*	*	0
TDIVU	W|L	Neoznačeno deljenje sa odsecanjem (MC68020)	TDIVU	<ea>,(Di:)Dj	-	*	*	*	0
TMULS	L	Označeno množenje sa odsecanjem (MC68020)	TMULS		-	*	*	*	0
TMULU	L	Neoznačeno množenje sa odsecanjem (MC68020)	TMULU		-	*	*	*	0
TPcc	W	Trap na kod uslova (MC68020)	Tpcc	#xxx	-	-	-	-	-
TRAP	U	Trap	TRAP	#<vector>	-	-	-	-	-

Mnenomik	Podrazumevana veličina operanda	Operacija	Asemblerska sintaksa		Kod uslova				
					X	N	Z	V	C
TRAPV	U	Trap na prekoračenje	TRAPV		-	-	-	-	-
TST	W	Testiranje operanda	TST	<ea>	-	*	*	0	0
UNLK	U	Odvezivanje	UNLK	An	-	-	-	-	-
UNPK	U	Otpakivanje BCD (MC68020)	UNPK UNPK	-(Ax),-(Ay),#<adjust> Dx,Dy,#<adjust>	-	-	-	-	-

Napomene:
1. <ea> označava efektivnu adresu.
2. Asembler prihvata DBRA kao F (nikad tačno) uslov.
3. Rc osnačava upravljaki registar.
4. <lista registara> označava registre izabrane za transfeer u ili iz memorije. <lista registara> može biti
   Rn - jedan registar.
   Rn - Rm - opseg uzastopnih registara gde je m veće od n.
   Bilo koja kombinacija gornjeg razdvojena kosom crtom.
5. <disp> je ceo broj u dvoičnom komplementu, veličine 16 bitova, koji se znakovno proširuje do 32 bita pre dodavanja ukazatelju magacina.
6. BFxxx instrukcije i TDIVx instrukcije koriste vitičaste zagrade { } u svojoj sintaksi. U svim ostalim slučajevima zagrade predstavljaju opcioni element,
7. Znak dve tačke : se zahteva u nekim novim instrukcijama mikroprocesora MC68020.
8. Podrazumevani kodovi veličina:
   B - Bajt (.B)
   W - Reč (.W)
   L - Duga reč (.L)
   U - Neoznačeno (nije dozvoljen kod veličine)
   U* - Normalno neoznačen
9. Asemblerska sintaksa za svaku instrukciju podrazumeva opcionu labelu i kod veličine tamo gde se može primeniti.

Tabela 6.8. Pregled skupa instrukcija MC68881.

Mnenomik	Podrazumevana veličina operanda	Operacija	Asemblerska sintaksa		Kodovi uslova				
					X	N	Z	V	C
FABS	W|X	Apsolutna vrednost funkcije	FABS FABS FABS	<ea>,FPn FPm,FPn FPn	*	*	*		*
FACOS	W|X	Arkus kosinus	FACOS FACOS FACOS	<ea>,FPn FPm,FPn FPn	*	*	*		*
FADD	X	Sabiranje u pokretnom zarezu	FADD FADD	<ea>,FPn FPm,FPn	*	*	*		*
FASIN	W|X	Arkus sinus	FASIN FASIN FASIN	<ea>,FPn FPm,FPn FPn	*	*	*		*
FATAN	W|X	Arkus tangens	FATAN FATAN FATAN	<ea>,FPn FPm,FPn FPn	*	*	*		*
FATANH	W|X	Hiperbolički arkus tangens	FATANH FATANH FATANH	<ea>,FPn FPm,FPn FPn	*	*	*		*
FBcc	W	Koprocesorsko uslovno grananje (MC68881)	FBcc	<labela>	-	-	-		-
FCMP	X	Poređenje u pokretnom zarezu	FCMP	<ea>,FPn	*	*	*		*
FCOS	W|X	Kosinus	FCOS FCOS FCOS	<ea>,FPn FPm,FPn FPn	*	*	*		*
FCOSH	W|X	Hiperbolički kosinus	FCOSH FCOSH FCOSH	<ea>,FPn FPm,FPn FPn	*	*	*		*
FDBcc	W	Dekrementiraj i granaj se na osnovu uslova (MC68881)	FDBcc	Dn,<labela>	-	-	-		-
FDIV	X	Deljenje u pokretnom zarezu	FDIV FDIV	<ea>, Fpn FPm,FPn	*	*	*		*
FETOX	W|X	Funkcija e^x	FETOX FETOX FETOX	<ea>,FPn FPm,FPn FPn	*	*	*		*

Mnenomik	Podrazumevana veličina operanda	Operacija	Asemblerska sintaksa	Kodovi uslova X	N	Z	V	C
FETOXM1	W|X	Funkcija $e^{x(x-1)}$	FETOXM1 <ea>,FPn FETOXM1 FPm,FPn FETOXM1 FPn	*	*	*		*
FGETEXP	W|X	Eksponent	FGETEXP <ea>,FPn FGETEXP FPm,FPn FGETEXP FPn	*	*	*		*
FGETMAN	W|X	Mantisa	FGETMAN <ea>,FPn FGETMAN FPm,FPn FGETMAN FPn	*	*	*		*
FINT	W|X	Celobrojni deo	FINT <ea>,FPn FINT FPm,FPn FINT FPn	*	*	*		*
FINTRZ	W|X	Celobrojni deo zaokružen na nulu	FINTRZ <ea>,FPn FINTRZ FPm,FPn FINTRZ FPn	*	*	*		*
FLOG10	W|X	Dekadni logaritam	FLOG10 <ea>,FPn FLOG10 FPm,FPn FLOG10 FPn	*	*	*		*
FLOG2	W|X	Binarni logaritam	FLOG2 <ea>,FPn FLOG2 FPm,FPn FLOG2 FPn	*	*	*		*
FLOGN	W|X	Prirodni logaritam	FLOGN <ea>,FPn FLOGN FPm,FPn FLOGN FPn	*	*	*		*
FLOGNP1	W|X	Prirodni logaritam od (x+1)	FLOGNP1 <ea>,FPn FLOGNP1 FPm,FPn FLOGNP1 FPn	*	*	*		*
FMOD	X	Moduo u pokretnom zarezu	FMOD <ea>,FPn FMOD FPm,FPn	*	*	*		*
FMOVE	W	Kopiranje u FP registar iz memorije ili drugog FP registra	FMOVE <ea>,FPn FMOVE FPm,FPn	*	*	*		*
	W	Kopiranje iz FP registra u memoriju	FMOVE FPn,<ea> FMOVE.P Fpn,<ea>{#data} FMOVE.P FPn,<ea>{Dn}	*	*	*		*
	L	Kopiranje u/iz memorije iz/u specijalni registar	FMOVE <ea>,CONTROL|STATUS|IADDR FMOVE.P CONTROL|STATUS|IADDR,<ea>					
FMOVECR	X	Kopiranje iz ROM-a u FP registar	FMOVECR #data,FPn	*	*	*		*
FMOVEM	X	Kopiranje u više FP registara iz memorije	FMOVEM <ea>,<fp reg lista>	*	*	*		*
	X	Kopiranje u registar za podatke	FMOVEM <ea>,Dn FMOVEM <ea>,CONTROL|STATUS|IADDR					
	L	Kopiranje u specijalni registar	FMOVEM <ea>,FPCR|FPSR|FPIAR					
	W	Kopiranje iz više FP registara u memoriju	FMOVEM <fp reg lista>,<ea>					
	X	Kopiranje iz registra za podatke u memoriju	FMOVEM Dn,<ea> FMOVEM CONTROL|STATUS|IADDR,<ea>					
	L	Kopiranje iz specijalnog registra u memoriju	FMOVEM FPCR|FPSR|FPIAR,<ea>					
FMUL	X	Množenje u pokretnom zarezu	FMUL <ea>,FPn FMUL Fpm,FPn	*	*	*		*
FNEG	W|X	Promena znaka	FNEG <ea>,FPn FNEG Fpm,FPn FNEG FPn	*	*	*		*
FNOP	U	Bezefektna naredba	FNOP	*	*	*		*
FREM	X	Ostatak u pokretnom zarezu	FREM <ea>,FPn FREM FPm,FPn	*	*	*		*
FRESTORE	U	Obnavljanje internog stanja koprocesora (MC68881) (P)	FRESTORE <ea>	-	-	-		-
FSAVE	U	Koprocesorsko pamćenje (MC68881) (P)	FSAVE <ea>	-	-	-		-

Mnenomik	Podrazumevana veličina operanda	Operacija	Asemblerska sintaksa		Kodovi uslova X	N	Z	V	C
FSCALE	X	Umnožavanje eksponenta u pokretnom zarezu	FSCALE FSCALE	<ea>,FPn FPm,FPn	*	*	*		*
FScc	B	Postavi zavisno od uslova (MC68881)	Fscc	<ea>	-	-	-		-
FSGLDIV	X	Deljenje u pokretnom zarezu u jednostrukoj preciznosti	FSGLDIV FSGLDIV	<ea>,FPn FPm,FPn	*	*	*		*
FSGLMUL	X	Množenje u pokretnom zarezu u jednostrukoj preciznosti	FSGLMUL FSGLMUL	<ea>,FPn FPm,FPn	*	*	*		*
FSIN	W|X	Sinus	FSIN FSIN FSIN	<ea>,FPn FPm,FPn FPn	*	*	*		*
FSINCOS	W|X	Sinus/kosinus	FSINCOS	<ea>,FPm:FPn	*	*	*		*
FSINH	W|X	Sinus hiperbolički	FSINH FSINH FSINH	<ea>,FPn FPm,FPn FPn	*	*	*		*
FSQRT	W|X	Kvadratni koren	FSQRT FSQRT FSQRT	<ea>,FPn FPm,FPn FPn	*	*	*		*
FSUB	X	Oduzimanje u pokretnom zarezu	FSUB FSUB	<ea>,FPn FPm,FPn	*	*	*		*
FTAN	W|X	Tangens	FTAN FTAN FTAN	<ea>,FPn FPm,FPn FPn	*	*	*		*
FTANH	W|X	Tangens hiperbolički	FTANH FTANH FTANH	<ea>,FPn FPm,FPn FPn	*	*	*		*
FTENTOX	W|X	10^x	FTENTOX FTENTOX FTENTOX	<ea>,FPn FPm,FPn FPn	*	*	*		*
FTcc FTRAPcc	U	Trap na uslov bez parametara (MC68881)	FTcc FTRAPcc		-	-	-		-
FTPcc FTRAPcc	W	Trap na uslov sa parametarom (MC68881)	FTPcc FTRAPcc	#<data> #<data>	-	-	-		-
FTST	W|X	Testiranje operanda u pokretnom zarezu	FTST FTST	<ea> FPn	*	*	*		*
FTWOTOX	W|X	2^x	FTWOTOX FTWOTOX FTWOTOX	<ea>,FPn FPm,FPn FPn	*	*	*		*
FYTOX	X	y^x u pokretnom zarezu	FYTOX FYTOX	<ea>,FPn FPm,FPn	*	*	*		*

Napomene:
1. Asemblerska sintaksa za svaku instrukciju podrazumeva opciono polje labela i kod veličine operanda tamo gde se može primeniti.
2. Instrukcije kod kojih je podrazumevana veličina operanda W|X uzimaju X kao podrazumevanu veličinu jedino kada je operand jedan Fpn registar.
3. cc kod upotrebe u MC68881 instrukcijama označava kod uslova u pokretnom zarezu.
4. (P) označava privilegovanu instrukciju

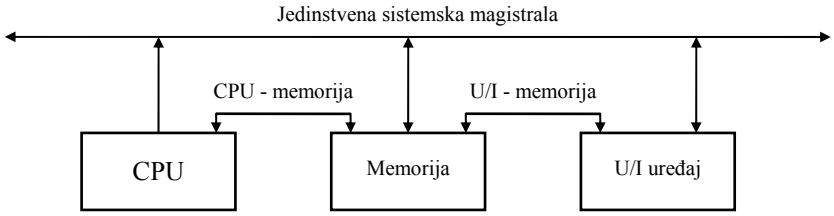
6.2. Keš (skrivena) memorija

Među brojnim faktorima koji ograničavaju performanse računarskog sistema dva su najkarakterističnija:
1. vreme oristupa memoriji (memory access time),
2. propusnost magistrale (bus bandwidth - BW).

Prenos podataka ka/iz memorije je, kao aktivnost, oduvek bio i biće usko grlo skoro kod svih računarskih sistema, nezavisno od toga da li su računari projektovani za opštu ili specifičnu namenu. Neki od faktora koji imaju presudni uticaj na stvaranje tog uskog grla su:

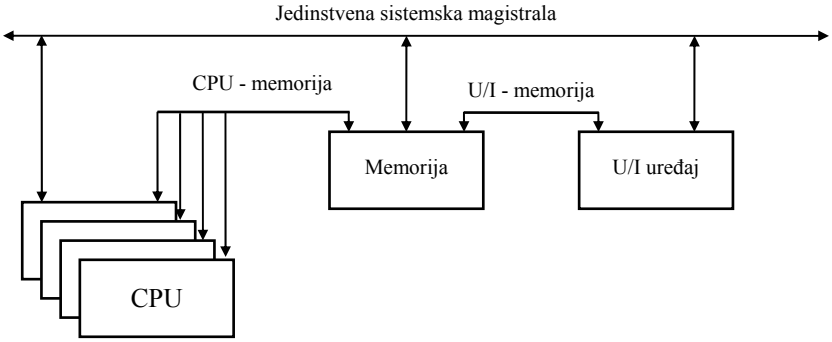
1. Program i podaci kod standardnih Fon Nojmanovih (Von Neuman) mašina dele zajedničku memoriju. Sa druge strane, U/I pristupima se (slika 6.11) takođe povećavaju konflikti, jer u suštini ova dva podsistema (CPU i U/I) pristupaju istom resursu (memoriji). Time se indirektno još više ističe problem uskog grla.
2. Napredak tehnologije (submikronski CMOS) uslovio je rad pri velikim brzinama. Sada se od projektanata zahteva ugradnja memorijskih podsistema koji imaju veoma brz odziv. 32-bitni mikroprocesor Motorola

MC68030 danas radi sa taktnom frekvencom od 20MHz, a ona će uskoro biti 30mhz. Projektanti sistema veruju da će se početkom devedesetih godina dostići taktna frekvencija od 200Mhz što predstavlja teoretsku granicu CMOS kola. Rad pri brzinama čak i od 20mhz u velikoj meri zavisi od vremenskog kašnjenja signala kroz kablove čija dužina ne premašuje ni desetinu centimetra, a takođe i od kašnjenja (vreme propagacije) kroz drajverska primopredajna kola. Ovakva kašnjenja usporavaju rad sistema i nalažu preduzimanje nekih upravljačkih akcija (uvođenje jednog, dva, tri ili više stanja čekanja), a time i direktno uslovljavaju da sistem radi sa brzinom koja je ispod CPU-ove.



Sl. 6.11.

3. Kod savremenih računarskih arhitektura kao štu su multiprocesorski sistemi koji rade u čvrstoj sprezi i računarske arhitekture zasnovane na RISC procesorima, problem propusnosti memorije (memory bandwidth) je još izrazitiji. Kod multiprocesorskih sistema, kao što je onaj prikazan na slici 6.12, pozitivni efekat koji se postiže paralelnim izvršavanjem većeg broja zadataka, uslovio je, sa druge strane, da se poveća broj zahteva za pristupom memoriji (negativni efekat). U principu, K procesora u poređenju sa jednoprocesorskim sistemom zahteva K puta veću memorijsku propusnost. Kod RISC-ova (Reduces Instruction Set Computer) na mašinskom nivou postoji veoma prosto kodiranje instrukcije, jer je svesno žrtvovana kodna gustina, tj. složenost kodiranja instrukcija koja ja postignuta kod CISC-ova (Complex Instruction Set Computer). Ovakav pristup uslovljava da RISC CPU u poređenju sa CISC CPU-om uvek zahteva veću propusnost memorije.



Sl. 6.12. Blok šema višeprocesorskog sistema.

4. Težnja da se mikroračunarski sistemi učine univerzalnim i otvorenim za dalju nadgradnju prvenstveno zahteva da se oni zasnivaju na specifičnoj magistrali kao što su VME, MULTIBUS i dr. Ovakav pristup dovodi do neoptimalnog prenosa podataka između CPU-a i memorije čime se direktno utiče na smanjenje efikasnosti sistema. Drugim rečima, tehnološka poboljšanja u izradi CPU-a nisu smanjila cenu komunikacije CPU↔memorija onako brzo kao što je to slučaj sa cenom izračunavanja.

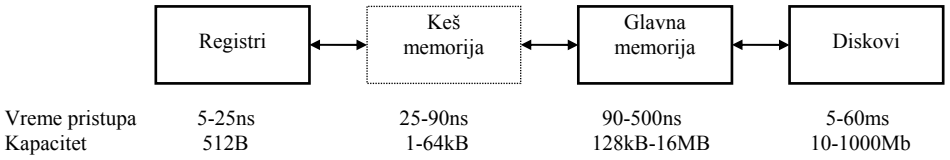
Na osnovu do sada izloženog je jasno da se problem povećanja memorijske propusnosti može rešiti na jedan od sledeća tri načina:

1. Korišćenjem skupih DRAM-ova, ili još skupljih SRAM-ova velikog kapaciteta, pomoću kojih se može realizovati glavna memorija imajući u vidu da se danas i najstandardnije računarske mreže, kao što su personalni računari tipa IBM PC, isporučuju skoro uvek sa najmanje 4MB instaliranog RAM-a, cena memorijskih sistema realizovanih sa brzima DRAM-ovima ili SRAM-ovima bi bila izuzetno visoka čak i za najklasičnije računarske mašine. Sa druge strane, fizički je, zbog prostora, nemoguće realizovati memorijski podsistem relativno velikog kapaciteta sa memorijskim čipovima malog kapaciteta koji će biti instalirani na istoj štampanoj ploči kao i CPU.

2. Korišćenjem jeftinijih DRAM-ova, što se izvodi ubacivanjem stanja čekanja (wait states) kod svakog ciklusa pristupa memoriji, ili korišćenjem poboljšanih šema za memorijsko preslikavanje što se izvodi MMU-om.
3. Instaliranjem memorije malog kapaciteta sa veoma brzim pristupom, koja se nalazi između CPU-a i glavne memorije i predstavlja mali izabrani podskup glavne memorije računara. Ovaj memorijski podskup koji glavnu meoriju računara preslikava u RAM sa veoma brzim pristupom je fizički lociran blizu CPU-a i zove se *keš* (cashe) *memorija*.

Prvo rešenje je sigurno najjednostavnije, ali nesumnjivo i najskuplje. Ono se danas koristi kod onih aplikacija gde cena nije važna, kao što su namenski kontroleri za vojne i industrijske primene. Drugo rešenje se takođe jednostavno može realizovati i koristi se tamo gde brzina rada nije kritična tako da se svesno gubi najveći broj prednosti koje nude savremeni 32-bitni mikroprocesori. Treća solucija je do skoro bila isuviše složeno i skupo rešenje za sisteme srednje složenosti, a primenjuje se prvenstveno zbog nove generacije SRAM-ova velikog kapaciteta, posebno Cache-Tag-RAM-ova, jer predstavlja najefikasnije rešenje. Da bi jasnije ukazali na ovaj problem analizirajmo najpre princip i način rada keš memorije.

Po definiciji keš memorija je mala memorija sa relativno brzim pristupom koja se nalazi između CPU-a i glavne memorije sistema (slika 6.13).

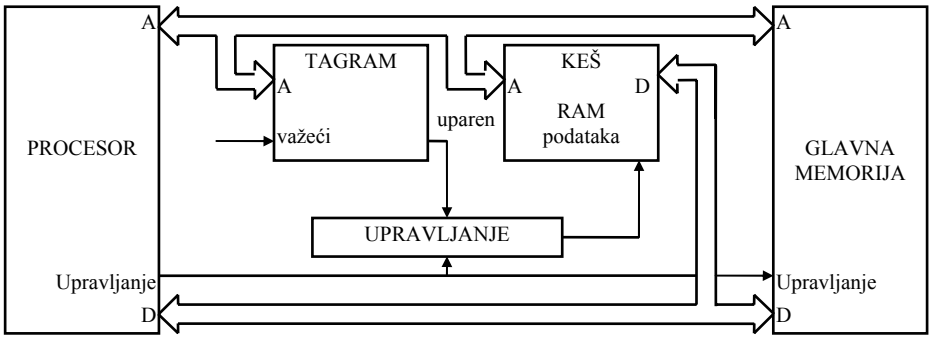


Sl. 6.13. Hijerarhija memorije kod standardnih računara.

Strukturno, keš sadrži kombinaciju podataka kojima se pridružuju adrese glavne memorije. Keš memorija se sastoji iz tri dela:

1. Standardni SRAM sa veoma brzim pristupom u koji se smeštaju kopije podataka koji se upisuju ili čitaju iz glavne memorije DRAM tipa.
2. Veoma brzo SRAM polje u kome se čuvaju adrese kojima ti podaci pripadaju.
3. Upravljačka logika.

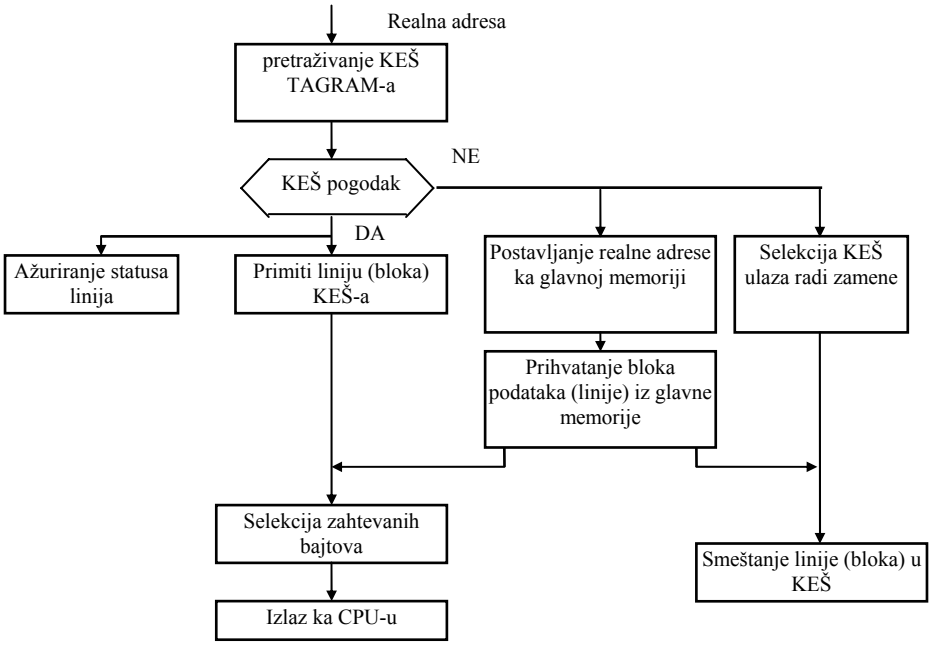
Sistem zasnovan na korišćenju keš memorije (slika 6.14) radi na sledeći način:



Sl. 6.14. Generalizovani keš blok dijagram.

1. CPU generiše zahtev za pristup memoriji (na primer, ciklus čitanja).
2. Ako se adresa CPU-ovog pristupa upari sa adresom smeštenom u kešu, tada se odgovarajući podatak predaje CPU-u iz keš memorije. Ovo se zove *pogodak* (hit), a CPU-ov pristup memoriji obavlja se brzinom koja je svojstvena brzini pristupa keš memorji, tj. bez stanja čekanja.
3. Ako se adresa CPU-ovog pristupa ne upari sa bilo kojom adrsom u keš memoriji, tada se obavlja pristup glavnoj memoriji. Ovo se zove *promašaj* (miss), a brzina pristupa je svojstvena brzini pristupa glavne memorije, sa mogućim ubacivanjem stanja čekanja.
4. Kada se javi promašaj, podaci se takođe smeštaju u keš, tako da će sledeći pristup ovoj adresi imati za efekat pogodak.

Dijagram toka rada keš memorije prikazan je na slici 6.15.



Slika 6.15. Dijagram toka rada keša.

Uspeh keš memorije kod izvršenja najvećeg broja programa se može objasniti analizom zaključaka koji važe za princip lokalnosti. Princip lokalnosti ukazuje na to da u kratkom vremenskom intervalu program teži da favorizuje ograničeni deo adresnog prostora (mali deo memorije). Ovu lokalnost čini *prostorna* i *vremenska* lokalnost. Ako je moguće da se programski skup sa kojim se operiše (deo programa oko naredbe koja se trenutno izvršava) smesti u keš, tada će se najveći broj obraćanja memoriji od strane procesora vršiti posredstvom keša, a to će u značajnoj meri povećati performanse sistema.

Princip *vremenske lokalnosti* znači da je informacija koja će se koristiti u bliskoj budućnosti već spremna za korišćenje ili je bila skoro korišćena. Tipičan primer je programska petlja:

```
FOR J=2 TO 200
        iskaz1
        iskaz2
        iskaz3
        iskaz4
END FOR
```

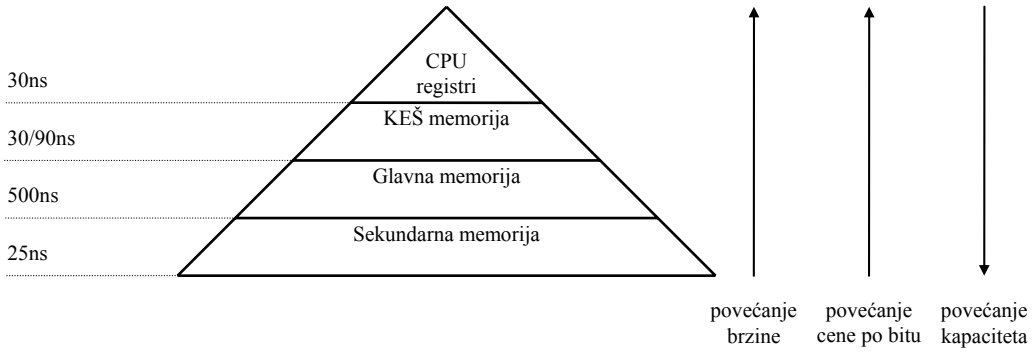
Brojač petlje J kao i iskazi 1-4 veoma se često koriste u toku kratkog vremenskog perioda.

Princip *prostorne lokalnosti* označava da je informacija koja će se u bliskoj budućnosti koristiti veoma bliska informaciji koja se već koristi. Ovo je slučaj kada program izvršava instrukcije koje slede jedna za drugom. Čak i u toku izvršenja petlji, ili kada se izvršavaju nesekvencijalna grananja ili skokovi (SHORT JUMP ili NEAR BRANCH), instrukcije koje se kao naredne izvršavaju karakterišu adrese koje su bliske tekućoj memorijskoj adresi.

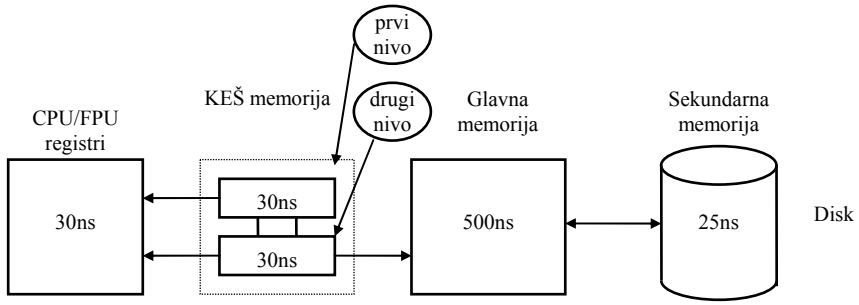
Na sličan način, podaci koji se koriste od strane programa uređeni su obično kao polja ili magacin (stack) i karakterišu ih slične osobine kao i programske instrukcije i pored toga što nisu na neki način takoskladno uređeni (misli se uređeni u odnosu na program).

6.3. Hijerarhija memorjije i stopa pogodaka

Sam naziv "cache" je francuskog porekla i znači "skriveno", tj. nevidljivo. Keš memorija kao mehanizam u memorijskoj hijerarhiji se umeće između glavne memorije i procesora (slika 6.16a), sa ciljem da se poboljša efektivna brzina memorijskog prenosa, a time i direktno poveća brzina rada procesora. Samo ime (skrivena memorija) ukazuje na činjenicu da je ovaj mehanizam transparentan korisniku. Keš memorija se obično realizuje pomoću poluprovodničkih sklopova čije su brzine kompatibilnije sa CPU-ovom, dok se glavna memorija realizuje jeftinijim i sporijim čipovima (slika 6.6b).



(a)



(b)

Sl. 6.16. Hijerarhija memorije kod računara.

Napomene: 1) Vrednosti za vreme pristupa od 30ns, 90ns, 500ns i 25ms uzete su kao ilustrativne (FPU označava procesor za rad u pokretnom zarezu (Floating Point Unit)

Ukupan broj pristupa koji predstavljaju pogodak (nezavisno od toga da li se pristupilo podacima ili programu) izražava se u procentima i zove se *stopa keš pogodaka* (cache hit rate). Kada se procesor obrati podatku koji se nalazi u kešu tada kažemo da se javlja *pogodak* (hit). Sa druge strane, ako se vrši obraćanje podatku koji nije prisutan u kešu, kažemo da se radi o *promašaju* (miss). Stopa pogodaka *h* za datu sekvencu memorijskih obraćanja definiše se kao broj pogodaka podeljen ukupnim brojem obraćanja memoriji. Na sličan način, stopa promašaja *m* je broj promašaja podeljen ukupnim brojem obraćanja memoriji. Kako svako obraćanje memoriji može biti bili pogodak bili promašaj, tada je $m=1-h$.

Primer 6.3:
Neka je vreme pristupa glavnoj memoriji T_{AGM}=500ns, a vreme pristupa keš memoriji T_{ACM}=50ns. Ako je stopa pogodaka 99%, izračunati srednje vreme pristupa ako je u sistem instalirana keš memorija.

Rešenje:
Srednje vreme pristupa memoriji, kada ne postoji keš, $T_{AM}=T_{AGM}$=500ns. Srednje vreme pristupa memoriji kada je u sistem instalirana keš meorija iznosi:

$$T_{AM}=T_{ACM}\times 0.99+T_{ACM}(1-0.99)=50\times 0.99+500\times 0.01=54.5\text{ns}$$

Postiže se poboljšanje koje je skoro red veličine.

Veličina stope keš pogodaka zavisi od kapaciteta keša, fizičke organizacije, programa koji se izvršava i keš algoritma zamene. Ako program pristupa lokacijama u memoriji potpuno slučajno, predvideti kojoj će se od narednih lokacija pristupiti u sledećem memorijskom ciklusu je skoro nemoguće. No, na sreću, u realnim situacijama programi pristupaju lokacijama koje su veoma blizu u odnosu na tekuću ili prethodnu.

Projektovanje keša ukazuje da je u najvećem broju slučajeva potrebno napraviti kompromis između sledećih, ponekad kontradiktornih, zahteva:

a) veća stopa pogodaka,
b) kraća vremena pristupa kešu kod pogodaka,
c) skraćenje vremena pristupa glavnoj memoriji kd promašaja,

d) niža cena.

Da bi se povećala stopa pogodaka, procesor koristi tehniku pribavljanja blokova (block fetch). Keš kontroler deli glavnu memoriju na blokove (poznate kao "line") čija je veličina obično 2, 4, 8 ili 16 bajtova. Veličina bloka je jedan od najvažnijih parametara projektantima sistema. Kada su blokovi mali, efikasnost pribavljanja blokova iz glavne memorije i stopa keš pogodaka opada.

Sa druge strane, ako su blokovi veliki, smanjuje se broj blokova koji se mogu smestiti u keš memoriju. Kod 32-bitnih procesora jedan blok obično čine dve do četiri reči. Kada se javi keš promašaj, keš kontroler premešta (kopira) čitav blok koji sadrži reč iz glavne memorije u keš memoriju.

Sumirajući sve što je do sada kazano o keš mehanizmu jedan opšti zaključak bi glasio: Kod projektovanja računarskih sistema ne postoji druga tehnika koja će za tako "male pare" (minorna investicija) dati tako velike rezultate (misli se na povećanje brzine rada koja se postiže zahvaljujući ugradnji keš meorije).

Kao ilustrativni primer opravdanosti korišćenja keš memorije analizirajmo mašinu Micro Vax 3500/3600, koja poseduje dvonivovski keš memorijski sistem prikazan na slici 6.17. Prvi nivo ima 1kB memorije i realizovan je direktnim ugrađivanjem u mikroporocesor. Drugi nivo ima 64kB memorije i instaliran je na ploči procesora a u toj memoriji se čuva informacija i o prvom nivou keš memorije, kao i neke druge informacije koje su potrebne procesoru.

a) Ako je stopa pogodaka za prvi nivo keš memorije 70%, a 30% za glavnu memoriju, tada trajanje efektivnog memorijskog ciklusa iznosi:

$$(70\% \times 100\text{ns}) + (30\% \times 400\text{ns}) = 70 + 120 = 190\text{ns}$$

Prvi nivo keša smanjuje efektivno vreme memorijskog ciklusa više od dva puta, tj. sa 400ns na 190ns.

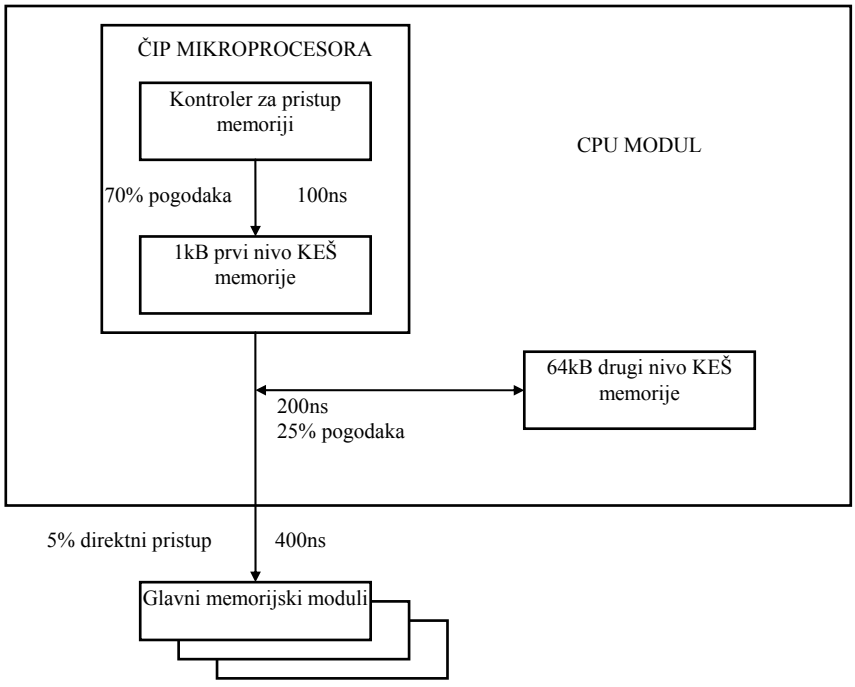
b) Ako je stopa pogodaka drugog nivoa keš memorije 95%, a 5% za glavnu memoriju, tada trajanje efektivnog memorijskog ciklusa iznosi:

$$(95\% \times 200\text{ns}) + (5\% \times 400\text{ns}) = 190 + 20 = 210\text{ns}$$

c) Ako je stopa pogodaka 70% za prvi nivo keš memorije, 25% za drugi nivo keša, a 5% za glavnu memoriju, tada efektivno vreme memorijskog ciklusa iznosi

$$(70\% \times 100\text{ns}) + (25\% \times 200\text{ns}) + (5\% \times 400\text{ns}) = 70 + 50 + 20 = 140\text{ns}.$$

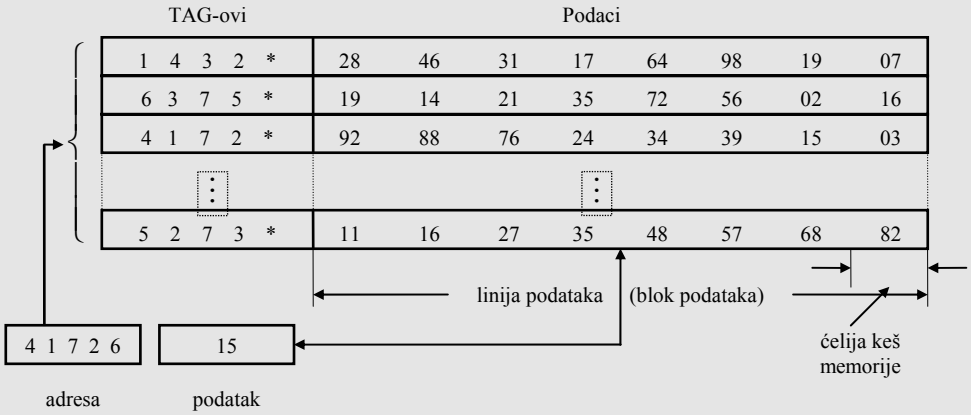
Na osnovu izloženog je evidentno da se efektivno vreme memorijskog ciklusa celog sistema smanjuje skoro tri puta, tj. sa 400ns na 140ns.



Sl. 6.17. Dvonivovska keš memorija kod Micro Vax 3500/3600.

Primer 6.4:

Struktura jedne tipične keš memorije ima sledeći oblik:



Princip rada keš memorije može se objasniti kroz sledeće aktivnosti:

1. Svako obraćanje ćeliji keš memorije predstavlja se kešu.
2. Keš pretražuje adresar adresnih TAG-ova da bi proverio da li je podatak (stavka) u kešu.
3. Ako podatak nije u kešu, javlja se promašaj.
4. U konkretnom slučaju, obraćanje adresi 41726 poklapa se sa adresnim TAG-om 4172*, gde * označava oktalni broj od 0 do 7. Podaci pridruženi adresnom TAG-u imaju adrese od 41720 do 41727, pa zbog toga mora da se pristupi sedmom podatku čija je adresa 41726.
5. Podatak čija je vrednost 15 kopira se u registar za podatke koji je deo keš sistema.
6. Obraćanje adresi 12345 generisaće promašaj pošto za prikazane adresne TAG-ove ne postoji uparenost TAG-ova sa ovom adresom.

6.4. Aspekti projektovanja keš sistema

U toku projektovanja keš memorijskog sistema neophodno je učiniti veliki broj kompromisa i doneti zadovoljavajuće odluke. Prva odluka se odnosi na liniju i obim keša. Drugi važni aspekti se odnose na određivanje politike pribavljanja u keš memoriju, politike smeštaja, politike zamene i politike ažuriranja keš memorije.

Politikom pribavljanja odlučuje se kada treba pribaviti podatak u keš. Algoritmi pomoću kojih se vrši *pribavljanje unapred* (prefech) obezbeđuju egzistenciju informacije u kešu pre nego što se ona eksplicitno zahteva.

Politika smeštaja ustanovljava relativnu poziciju linije keša sa jedne strane i linije glavne memorije sa druge strane. U tom smislu, u daljem tekstu će se analizirati politike smeštaja tipa: potpuno asocijativne, direktno preslikane i skupno asocijativne.

Politikom zamene u slučaju kada se javi keš promašaj, a keš memorija je puna, određuje se koju liniju treba zameniti. Tipično se politika zamene realizuje jednom od sledećih tehnika: proizvoljna, FIFO i LRU.

Politika ažuriranja glavne memorije odnosi se na ažuriranje glavne memorije kada se podatak u keš memoriji modifikuje. Ažuriranje se može izvesti kod svakog upisa (write-trought), ili kada se modifikovana linija zamenjuje (copy-back).

Takođe, veoma važan aspekt koji treba imati u vidu odnosi se na efekat multiprogramskog režima rada, kada je u sistem instalirana keš memorija. Multiprogramski režim rada ima uticaja na stopu pogodaka, jer kod svake komutacije konteksta odgovarajući proces zahteva svoje sopstvene podatke koji verovatno nisu u kešu pre nego što se proces aktivira. Stopa pogodaka inicijalno praznog keša (u odnosu na novi proces) naziva se *stopa pogodaka hladnog starta* (cold start). Stopa pogodaka hladnog starta se može poboljšati ako se omogući da se nekoliko konteksta simultano smesti u keš, koristeći se pri tome TAG-om identifikacije procesa pomoću koga se identifikuje odgovarajući proces.

Nasuprot kešu kod koga se smešta veći broj konteksta, moguće je povećati vremenski period (time slice) koji se dodeljuje svakom procesu, pa se na taj način procesu pruža šansa da napuni keš odgovarajućim (sopstvenim) podacima. Kada se proces aktivira, a podaci su još u kešu, stopa pogodaka se zove *stopa pogodaka vrućeg starta* (warm start).

Drugi aspekti projektovanja keš sistema na koje ćemo u daljem toku izlaganja posvetiti posebnu pažnju odnose se na:

- deobe keša na instrukcioni keš i keš za podatke,
- dupliranje propusnosti keša,
- korišćenje keš memorija sa većim brojem ulaza,

- probleme realizacije U/I u odnosu na keš.

6.5. Organizacija keš memorije i politika smeštaja

Postoji nekoliko načina realizije keš memorija. Oni se realizuju u odnosu na to kakva veza postoji između ulaza za adresni dodatak (cache tag) i adresnog prostora glavne memorije zbog koga su u suštini i ti adresni dodaci uvedeni. Keš se obično organizuje u linijama gde linija predstavlja kontinualnu sekvencu bajtova odgovarajućeg obima. Na primer, ako je keš kapaciteta 4kB, on se može realizovati u 256 linija gde svaka linija sadrži 16-bajtni memorijski deo (memory chunk). Izbor veličine memorijske linije (parčića, tj. blokova) je veoma važan memorijski parametar. Ako je ona mala, broj obraćanja memoriji se povećava, a ako su linije velike postoji velika verovatnoća da se one neće naći u memoriji. Sa druge strane, pretraživanje keš memorije mora biti veoma efikasno, jer se proces pretraživanja obavlja u toku svakog pristupa memoriji. Ako se usvojeni keš kapaciteta 4kB podeli na 256 linija obima 16 bajtova, osnovni zadatak kod pretraživanja sastoji se u brzom određivanju da li koja od 256 linija sadrži podatak koga tražimo.

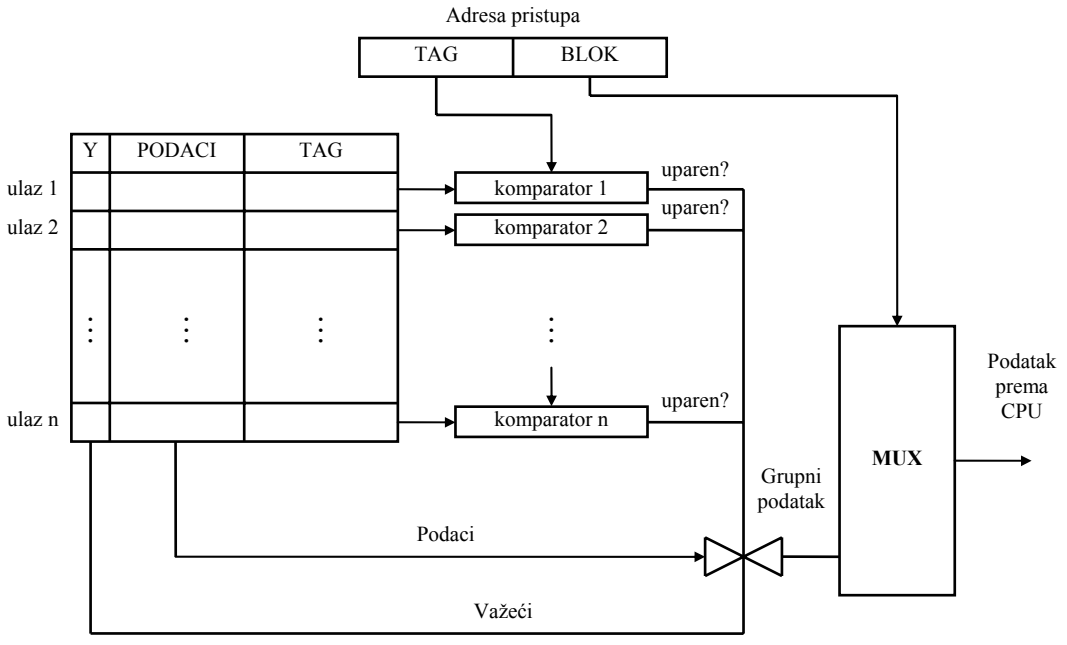
Jedan od najvažnijih projektantskih aspekata keša je smeštanje linija glavne memorije u keš memoriju; naime, gde postaviti liniju iz glavne memorije u kešu tako da se ona može jednostavno izbaviti. Ovo predstavlja problem, jer je keš po kapacitetu manji u odnosu na glavnu memoriju, tako da se nekoliko memorijskih linija preslikava u istu keš liniju. Tri poznate tehnike koje se primenjuju kod implementacije pretraživanja podataka kod keš memorije, a indirektno i načini preslikavanja podataka iz glavne memorije, su:

- **potpuno asocijativno preslikavanje** (FA - Fully Associative mapping),
- **direktno preslikavanje** (D - Direct mapping),
- **skupno asocijativno preslikavanje** (SA - Set Associative mapping)

U daljem tekstu ćemo detaljnije analizirati ove tehnike.

6.5.1. Potpuno asocijativno preslikavanje

Najmoćniji oblik keš memorije je potpuno asocijativna memorija, što znači da se bilo koji podatak iz memorijske lokacije može upisati u bilo koju keš lokaciju (slika 6.18).

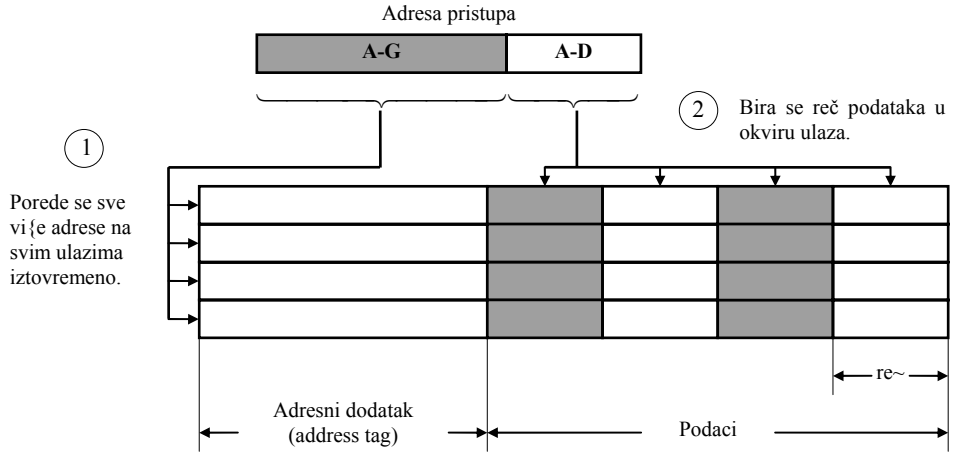


Sl. 6.18. Potpuno asocijativni keš.

Kod ovog tipa keš memorije logika za upravljanje i upoređivanje poredi adresu pristupa sa memorijskom adresom koja je smeštena u svaki keš ulaz (porede se odgovarajući TAG ulazi). Ako postoji uparivanje, odgovarajući podatak se prenosi iz keš memorije u CPU. Ako ne postoji uparivanje, blok podataka se čita iz glavne memorije i smešta u jedan od keš ulaza zajedno sa njegovom potpunom adresom.

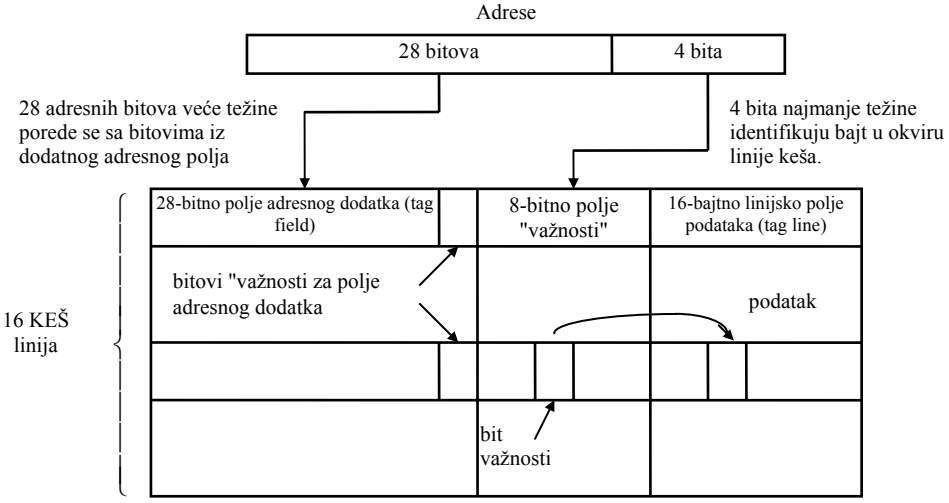
Na slici 6.19 je detaljnije prikazan princip selekcije ulaza kod asocijativne keš memorije. U ovom slučaju adresa pristupa se deli na dva polja A-G (TAG) i A-D (BLOK). Polje A-G se istovremeno poredi sa adresnim

dodatkom (address tag) na svim ulazima keš memorije. Ako postoji uparivanje, A-D poljem bira se reč podataka u okviru svakog ulaza. Asocijativna memorija se može realizovati kao sklop koji pripada mikroprocesoru, ili kao posebna jedinica (on-chip ili off-chip cache).



Sl. 6.19. Selekcija keš ulaza kod asocijativne memorije.

Na slici 6.20 je prikazana organizacija ″on-chip″ asocijativne keš memorije kod 32-bitnog mikroprocesora Z80000.

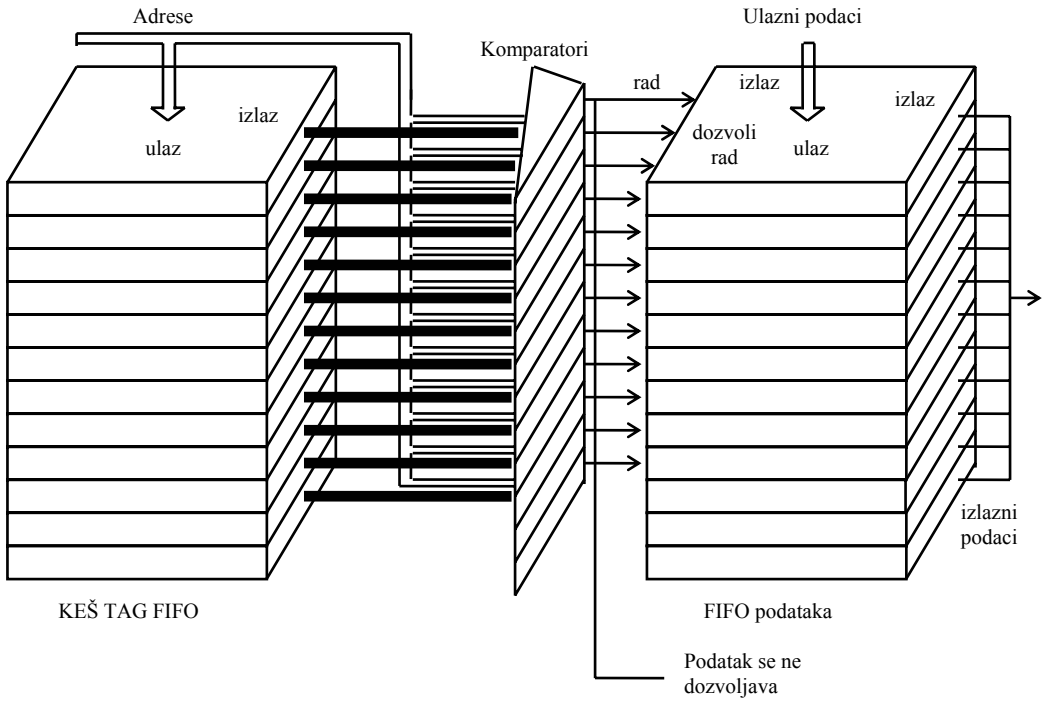


Sl. 6.20. Potpuno asocijativna keš memorija kod mikroporcesora Z80000 je organizovana sa 16 linija, gde svaka sadrži po 16 bajtova. 28 MS bitova fizičke adrese se porede sa "tag" linijama. Ako se javi pogodak i "tag" linija je važeća, bitovima 0-3 se identifikuje tražena (ciljna) reč u okviru linije. Ako je reč važeća obavlja se pristup keš memoriji.

Na slici 6.21 prikazana je jedna organizacija "off-chip" asocijativne memorije kod koje se preslikavanje zasniva na principu da se svi zahtevi smeštaju po redosledu pristizanja u specijalizovanu memoriju tipa FIFO (First-In-First-Out). Ovaj metod obezbeđuje veoma efikasno preslikavanje, jer je ostvareno kompletno čuvanje ulaza svih memorijskih lokacija kojima se najskorije pristupilo.

Osnovne karakteristike tehnike potpunog asocijativnog preslikavanja su seldeće:

- Potpuno asocijativno preslikavanje je veoma fleksibilno u pogledu zamene. Naime, bilo koja lokacija glavne memorije se može smestiti u bilo koji keš blok (liniju).
- Problem sa asocijativnom šemom je cena; naime, svaki keš ulaz sadrži odgovarajući "tag" čiji obim (broj bitova) odgovara potpunoj memorijskoj adresi s obzirom da se svaka reč može naći u bilo kom keš bloku. Da se performanse ne bi degradirale, svako "tag" polje paralelno se poredi u toku operacije pristupa memoriji. Ovakav način realizacije zahteva urađivanje velikog broja komparatora i dodatne upravljačke logike. Naime, potreban je po jedan adresni komparator za svaki asocijativni ulaz, a to čini ovo kašnjenje veoma različitim u odnosu na ono koje koristi standardna RAM ćelija. Zbog svoje složenosti, potpuno asocijativni pristup se koristi kod rešenja kod kojih je potrebno ugraditi keš memoriju malog kapaciteta. Asocijativne keš memorije većeg kapaciteta se obično izrađuju kao VLSI IC po porudžbini. Ali i integracija čipa je skupa investicija, jer su adresna ćelija i komparator nekoliko puta veći (zauzimaju veću površinu čipa) u odnosu na ćeliju podataka pa se postavlja pitanje: kog kapaciteta asocijativnu keš memoriju ima smisla integrisati, ako se uzme u obzir cena izrade?

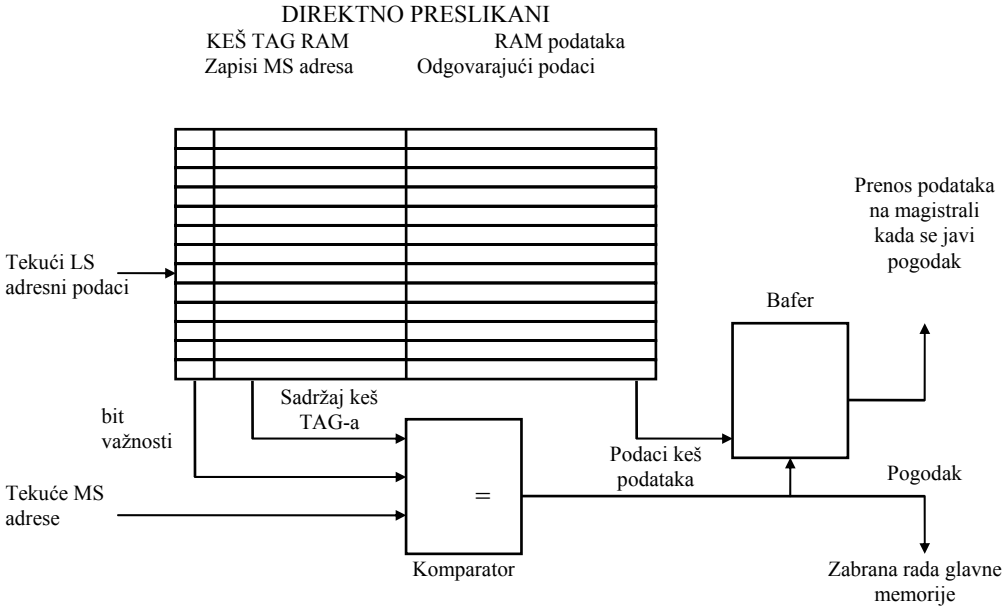


Sl. 6.21. Organizacija "off-chip" asocijativne memorije, organizovane na FIFO principu.

6.5.2. Direktno preslikavanje

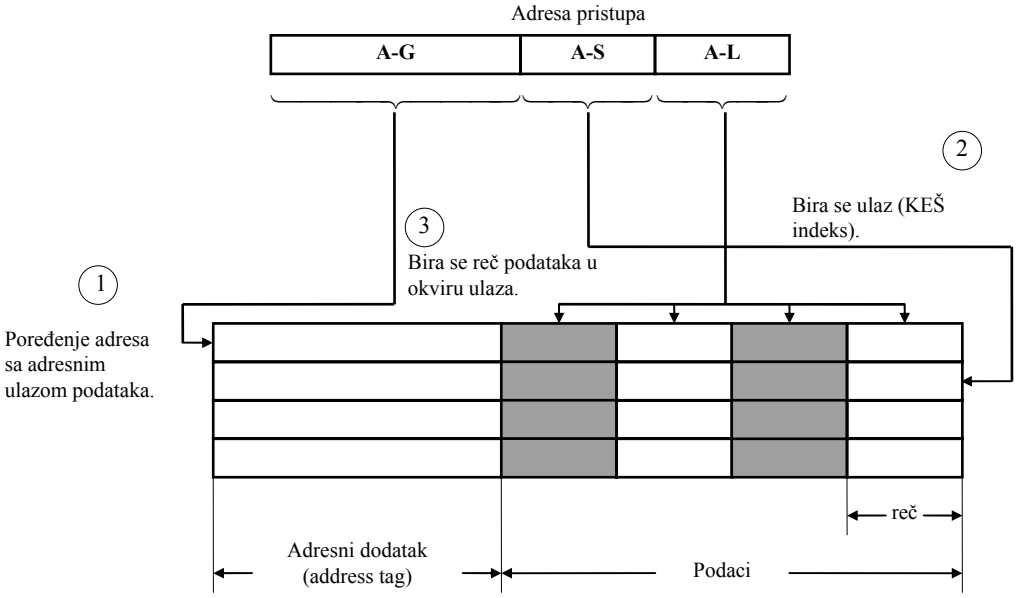
Nasuprot asocijativnom preslikavanju, direktno preslikavanje omogućava korišćenje statičkih RAM čipova u kojima se čuvaju zapisi adresa (address tag) kojima se najčešće pristupa. Kod ovog rešenja, slika 6.22, koristi se samo jedan adrresni komparator i standardne RAM ćelije za realizaciju ćelije podataka i adresa.

Keš memorija čiji je rad zasnovan na direktnom preslikavanju koristi "hash" kodiranje, softverski metod za simulacijku asocijativne memorije. Zapisi se ne smeštaju po redosledu javljanja već u lokacijama koje su u direktnom odnosu sa adresama u glavnoj memoriji. Na slici 6.23 je prikazana organizacija Odirektno preslikane keš memorije. Kao što se vidi sa slike 6.23, adresa pristupa se deli na tri polja: A-G, A-S i A-L. Poljem A-S bira se ulaz u okviru keša, a A-G polje se poredi sa adresnim dodatkom (adresnim TAG-om). Ako postoji uparenost (keš pogodak), A-L poljem se bir reč podataka u okviru ulaza. Kada ne postoji uparivanje vrši se obraćanje glavnoj meoriji. Kod direktnog preslikavanja zamena keš ulaza se vrši samo za one adrese pristupa koje su umnožak LS adresnog opsega. Sam mehanizam nije tako efikasan kao kod asocijativnog preslikavanja gde se zamena može vršiti recimo po principu LRU ili FIFO.



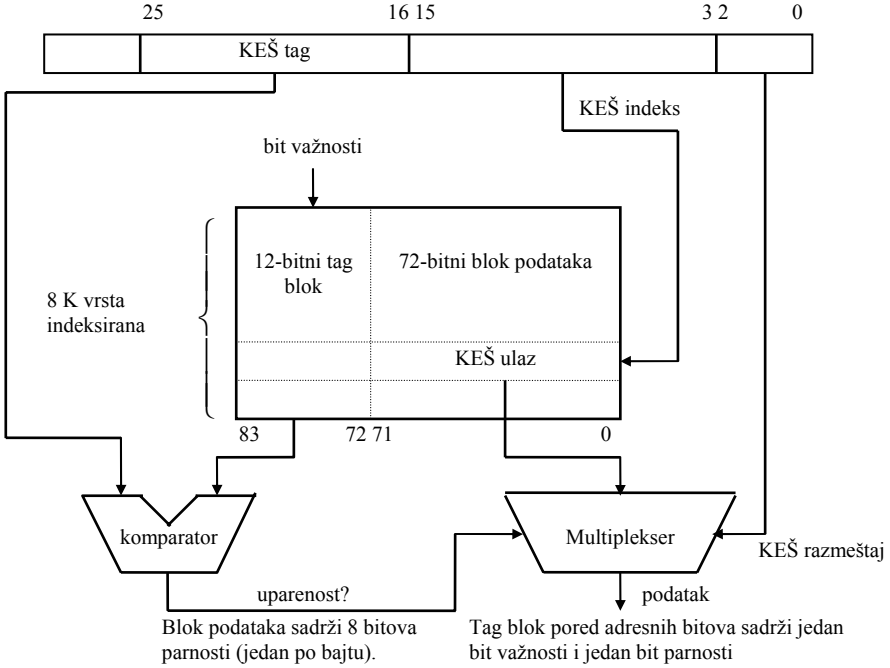
Sl. 6.22. Metod direktnog preslikavanja.

Na slici 6.24 je prikazan jedan primer korišćenja keš memorije sa direktnim preslikavanjem čiji je kapacitet 8192 ulaza. Svaki ulaz deli 26-bitnu adresu na 10-bitno tag polje (bitovi 25-16), 13-bitno indeksno polje (bitovi 15-3) i 3-bitno polje za selekciju bajtova (bitovi 2-0). Svaki keš ulaz čini: 10-bitna adresna informacija, 8 bajtova podataka pribavljenih iz glavne memorije i parnost. Bajtovi u okviru ulaza (keš ulazi) biraju se pomoću polja za selekciju bajtova.



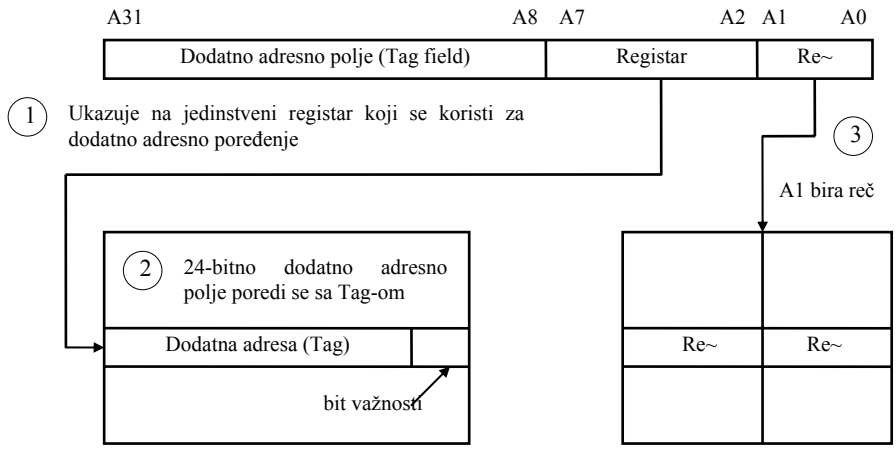
Sl. 6.23. Organizacija direktno preslikane keš memorije.

Indeksnim poljem (keš indeks) bira se jedan od 8192 keš ulaza. Da bi se odredilo da li se kod obraćanja memoriji javlio pogodak, poredi se adresno "tag" polje sa "tag" poljem keš ulaza. Ako se oba"tag"-a upare, javlja se keš pogodak.



Sl. 6.24. Organizacija direktno preslikanog keša kapaciteta 8192 ulaza.

Implemetacija direktno preslikane keš memorije kod mikroprocesora MC68020 prikazana je na slici 6.25.



6.25. Direktno preslikani keš kod MC68020.

Direktno preslikani keš kod mikroprocesora MC68020 deli memorijsku adresu na polja. Drugo polje ukazuje na jedinstvenu keš liniju u kojoj se može naći ciljna vrednost. "Tag" polje za tu liniju se poredi sa dodatnim adresnim poljem logičke adrese. Ako se javi uparivanje, a linija je važeća, adresni bitovi najmanje težine ukazuju na ciljnu vrednost.

Primer 6.5:

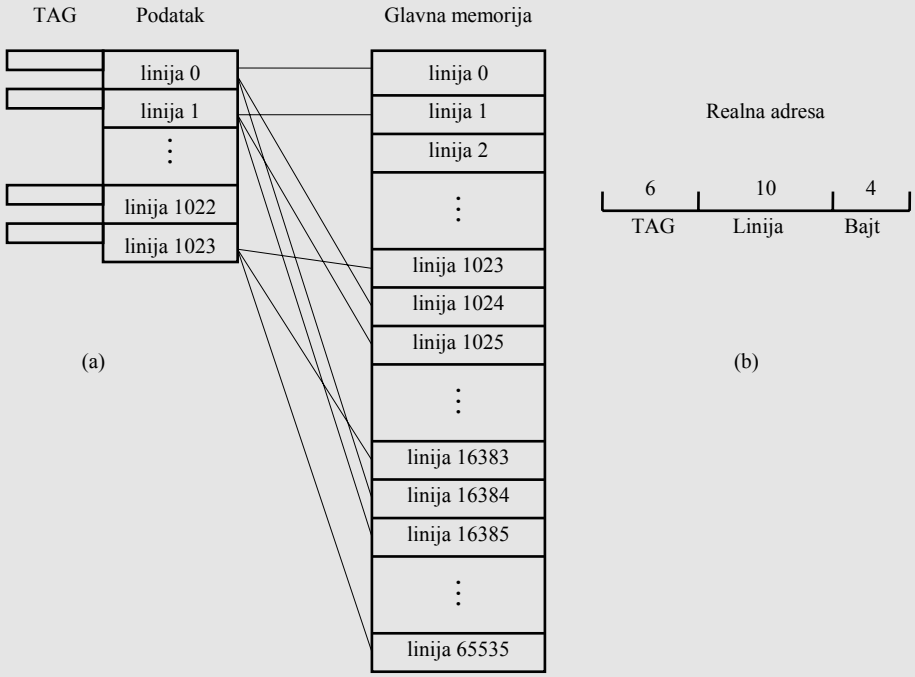
Neka je u računaru instaliran 1MB glavne memorije (adresibilna sa 20 bitova) i 16 kB keš. Svaka linija sadrži 16 bajtova. tako da glavnu memoriju čine 64K linije, a u keš se mogu smestiti 1024 linije. Prikazati kako izgleda organizacija direktno prslikane keš memorije.

Rešenje:

Organizacija direktno preslikanog keša prikazana je na slici 6.26. Shodno predloženoj šemi, linija I smeŠta se u liniju (I mod 1024) u kešu. Na slici 6.26b je prikazano kako se realna adresa deli na tri polja:

- 10-bitno polje linija koristi se za pristu kešu,
- 6-bitno polje TAG se koristi za proveru da li je adresirani ulaz u kešu,
- 4-bitno polje bajt koristi se da adresira deo linije.

Kod šeme sa slike 6.26 ne postoji asocijativno poređenje, a politika izmene je jednostavna što u suštini pojednostavljuje implementaciju hardvera. Ipak, u svakom keš ulazu može da se čuva samo jedna linija od ukupno 64 ($=2^6$) memorijskih linija koje se mogu preslikati u jednu keš liniju. Ako se dve memorijske linije koje se alternativno koriste preslikaju u istu keš liniju, stopa pogodaka će drastično opasti, čak i za slučaj da keš memorija nije puna.



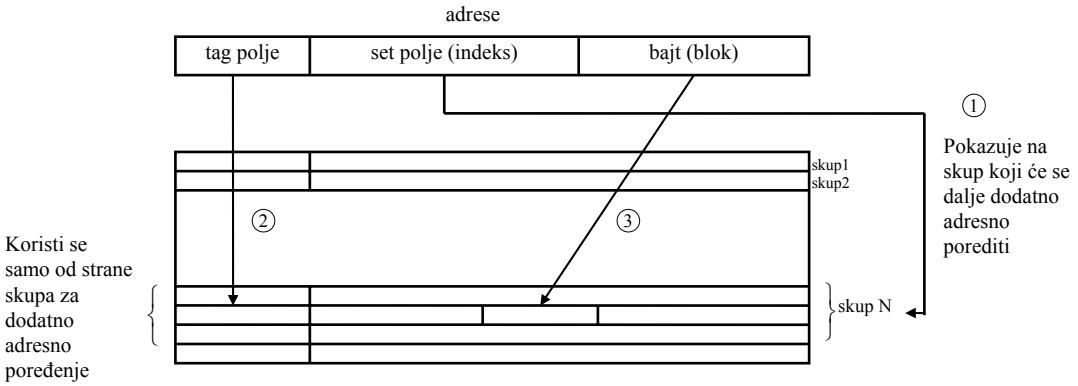
Sl. 6.26. Direktno preslikani keš.

Osnovne karakteristike direktnog preslikavanja mogle bi da iskažemo kroz sledeća zapažanja:

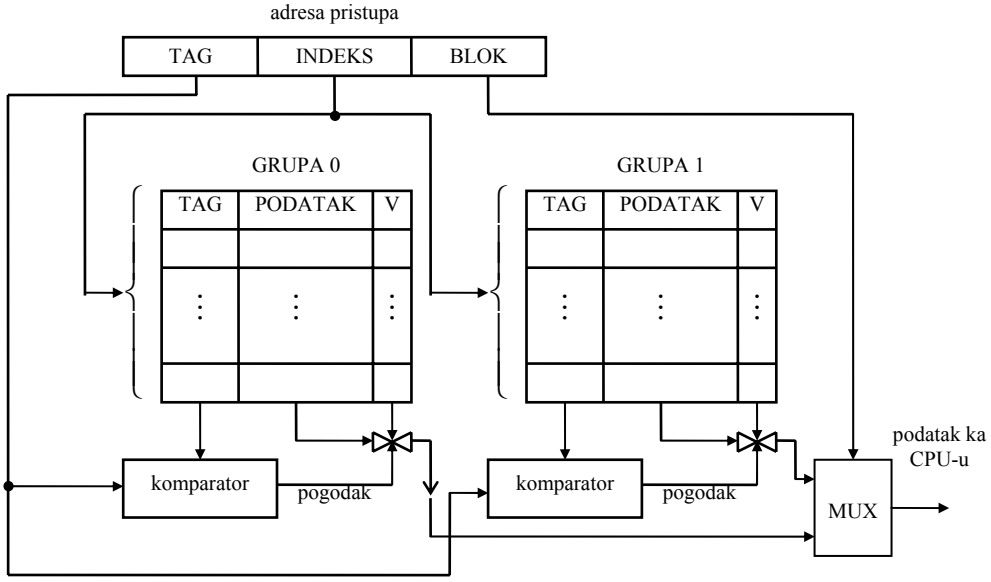
- Potrebna je ugradnja samo jednog komparatora. Kod direktno preslikane keš memorije, u odnosnu na asocijativnu keš memoriju, drastično je smanjen broj neophodnih poređenja zbog toga što je dozvoljeno svakom bloku (liniji) glavne memorije da koristi samo jednu moguću lokaciju keš memorije.
- Moguće je koristiti standardne SRAM-ove.
- Kapacitet keš memorije sa direktnim preslikavanjem u poređenju sa keš memorijom za asocijativno preslikavanje je znatno veći, cena nije tako visoka, a složenost hardvera nije tako velika.
- Ispitivanja su pokazala da je kod direktnog preslikavanja (kada je keš malog kapaciteta (<1kB)) stopa pogodaka keša veća u poređenju sa stopom pogodaka kod asocijativnog preslikavanja pri istim uslovima. Razlog ovako neočekivanog rezultata svakako treba tražiti u algoritmima zamene. Naime, koriste se algoritmi tipa LRU i FIFO koji po svemu sudeći nisu optimalni (za pomenute uslove rada) da izvrše nabolju zamenu odgovarajućih blokova.
- Ozbiljan problem se javlja kada mikroporcesor pristupa različitim memorijskim lokacijama koje imaju isti indeks. Kod ovakvih situacija stopa pogodaka opada, jer u keš memoriji postoji samo jedna lokacija (ili samo nekoliko ali je broj ograničen) za obraćanje memoriji (referenciranje memorije) u okviru jednog bloka. Drugim rečima, ako procesor izdaje česte zahteve za korišćenjem par lokacija koje fizički pripadaju onim delovima memorije koji se preslikavaju u istu keš liniju, tada kontroler mora često da pristupa glavnoj memoriji, jer se samo jedna od ovih lokacija može u datom trenutku smestiti u keš memoriju.
- Stopa promašaja može biti visoka, jer se neki programski segmenti mogu preslikati u isti keš blok.

6.5.3. Skupno asocijativna keš memorija

Organizacija ove memorije predstavlja proširenje koncepta direktnog preslikavanja, tj. to je hibridna metoda direktnog i asocijativnog preslikavanja. Skupno asocijativni sistemi sadrže veći broj banaka (grupa) tipa direktno preslikani keš koji se dinamički alociraju po principu LRU. Ovakvim konceptom rada obezbeđuje se kreiranje magacina koga čine direktno preslikane keš memorije. Ova tehnika koristi "lookup" procedure slične direktnom preslikavanju sa izuzetkom što se poljem "set" (slika 6.27) ukazuje na skup dodatnih adresnih ulaza. Drugim rečima, skupno asocijativni keš ima nekoliko grupa direktno preslikanih blokova (linija) kojima se pristupa paralelno. Blok lokacije, koje imaju isti keš indeks zovemo skup. Jednom keš indeksu se dodeljuje jedan skup, a u okviru skupa može postojati veći broj linija koje pripadaju različitim bankama direktno preslikanih blokova. Blok podataka koji se pribavljaju u keš može se smestiti u bilo koju blok lokaciju (liniju) koja, shodno adresnoj informaciji, pripada njegovom skupu.



Sl. 6.27. Skupno asocijativni keš funkcioniše kao direktno preslikani keš, sa izuzetkom da indeks polje ukazuje na skup keš ulaza, a dodatno adresno poređenje se vrši sa svakim elementom skupa.



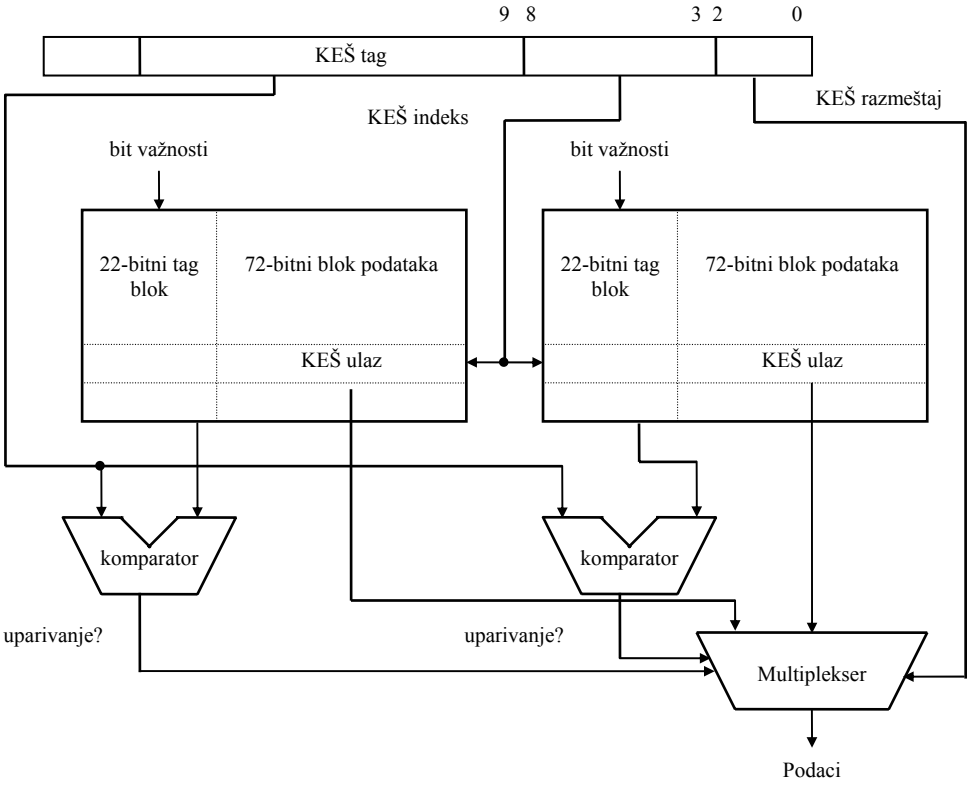
Sl. 6.28. Skupno asocijativni keš, obim skupa =2.

Organizacija tipične skupno asocijativne keš memorije prikazana je na slici 6.28. Struktura svake grupe slična je strukturi keš memorije kod koje postoji samo jedan skup (direktno preslikani keš), ali razlika je u tome što se može pristupiti različitim programima na memorijskoj lokaciji čijie adrese imaju isti indeks.

Skupno asocijativna keš memorija koja ima *n* banaka (grupa) direktno preslikanih keš memorija zove se ***n*-tostruka skupno asocijativna keš memorija**. Povećanje obima keš memorije korišćenjem asocijativno preslikavanje, u odnosu na povećanje veličine memorije kod direktnog preslikvanja, rezultira većoj stopi keš pogodaka. Zbog ovoga, kod većeg broja sistema skupno asocijativna memorija predstavlja kompromis između kompleksnosti i performansi.

Na slici 6.29 je prikazana jedna praktična realizacija dvostruke skupno asocijativne memorije. (Two-way set associative cache). Kao što se vidi sa slike 6.29, ulaz levog i desnog bloka imaju isti indeks. Logika za upravljanje i poređenje keš memorije poredi TAG polje memorijskih adresa sa TAG poljima oba ulaza koji pripadaju istoj indeksiranoj vrsti, tako da postoji veća verovatnoća da se dođe do uparivanja. Verovatnoća uparivanja raste sa brojem ulaza po vrsti. Kao što smo već naglasili i uočili detalje na slici 6.29, kod dvostruko asocijativnog keša postoje dve lokacije u kešu gde je moguće smestiti svaki blok podataka. Zbog toga keš kontroler mora da obavi dve komparacije kako bi utvrdio u kojoj je liniji smešten zahtevani podatak, ili da uopšte nije smešten. Realizacija skupno asocijativne memorije zahteva veći adresni TAG i veći SRAM prostor za smeštah TAG-ova. Zadatak keš kontrolera je da odredi koji blok keša treba zameniti kada se vrši pribavljanje novog bloka, s obzirom da postoji nekoliko lokacija (za slučaj na slici 6.29 samo dve) u koje se mogu smestiti podaci iz glavne memorije. Takođe, kod skupno asocijativne keš memorije logika keš kontrolera mora da odredi u koju liniju treba upisati podatak kada se u toku operacije upis javi keš promašaj.

Sa malim keš memorijama koristeći se dvostrukom ili četvororstrukom skupnom asocijativnošću dobija se skoro isti efekat (procenat pogodaka) kao i kod potpuno asocijativne keš memorije. Tipični primeri su realizacije on-chip keš memorija kod mikroporcesora MC68040, i486 i dr. Višestruka skupno asocijativna keš memorija može biti veoma efikasna ako se realizuje sa malim keš memorijskim blokovima koji se ugrađuju u čipu (on-chip cache).



Sl. 6.29. Dvostruka skupno asocijativna keš memorija smešta blok podataka u bilo koji od dva keš ulaza koja odgovaraju fizičkim adresnim indeks poljima. Ovaj pristup povećava procenat keš pogodaka bez povećanja kompleksnosti potpuno asocijativnog keša.

Primer 6.6:

Neka je u računaru instaliran 1MB glavne memorije (20 adresnih linija) i 16kB keša. Svaku liniju čini 16 bajtova. Glavnu memoriju čine 64K linija a keš 1024 linije. Prikazati kako izgleda organizacija četvororstruke skupno asocijativne keš memorije.

Rešenje:

Keš je podeljen na S skupova, asvaki sadrži L linija. Ako je M broj linija u kešu, tada je L=M/S. Na primer, kod četvorostrukog skupno asocijativnog keša (L=4) postoji S=M/L=1024/4=256 skupova, linija se može selektovati korišćenjem osam srednjih bitova adrese (slika 6.30) u toku faze selekcije skupa (linija iz glavne

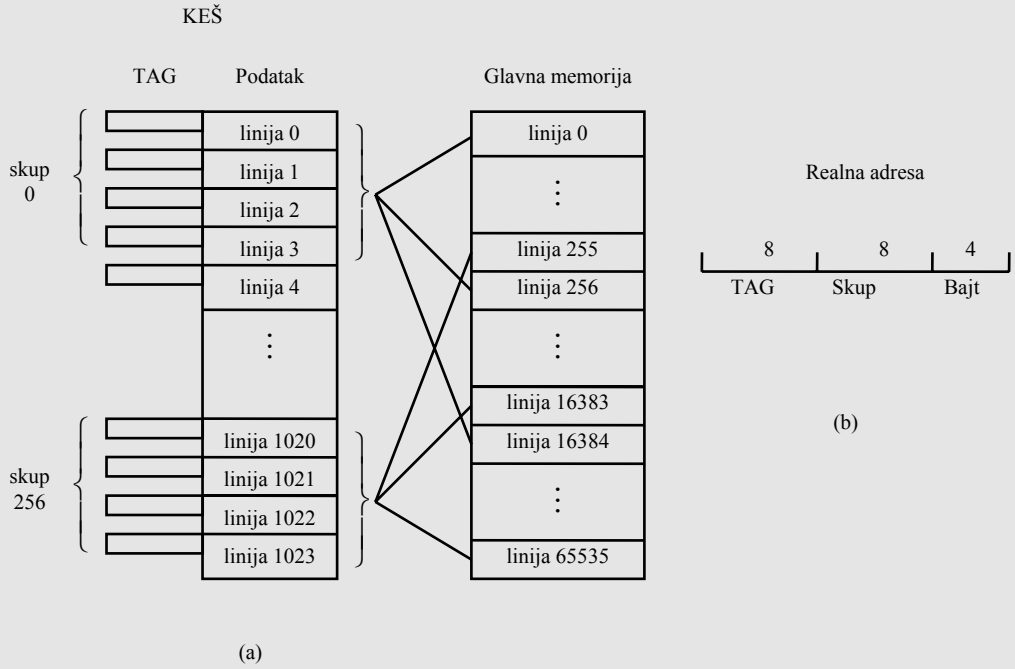
memorije se postavlja u skup I mod S = I mod 256), a zatim se vrši asocijativno poređenje TAG-ova. Prednost ove metode u odnosu na politiku potpune asocijativnosti je što je asocijativno pretraživanje proširuje (važi) samo nad L elemenata umesto nad M elemenata (M=1024), tako da je cena niža (potreban je mnogo manji broj komparatora).

Kada je S=1 (postoji samo jedan skup) dobija se potpuno asocijativni keš, a za L=1 (samo jedna linija po ulazu) imamo direktno preslikani keš. Zbog toga se, za dati obim keša M, mora napraviti kompromis između S i L, jer je proizvod S*L=*const*. Povećavanjem asocijativnosti L (veći broj linija po skupu) povećava se stopa pogodaka, ali se takođe povećava i cena. Eksperimentalni rezultati, koje je vršio Smith, pokazuju da asocijativnost od 2 do 16 daje najbolje rezultate kada se uzme u obzir stopa pogodaka i cena. Zbog ovog razloga najveći broj komercijalnih računara (VAX 11/780, IBM 3033, IBM 370/165 i dr.) koriste skupno asocijativni keš.

Na slici 6.30 je prikazana organizacija skupno asocijativnog keša memorijskog sistema za analizirani primer.

Osnovne karakteristike skupno asocijativnog preslikavanja se mogu iskazati kroz sledeća zapažanja:

1. Empirijska ispitivanja se pokazala da se skupno asocijativnom memorijom obima 4 može postići sita stopa pogodaka kao i kod potpuno asocijativnog preslikavanja.
2. Hardver skupno asocijativnog keša je kompleksniji u odnosu na hardver direktno preslikanog keša.
3. Brzina rada (odziva) skupno asocijativnog keša u poređenju sa brzinom potpuno asocijativnog i direktno preslikanog keša je manja.



Sl. 6.30. Skupno asocijativni keš.

6.6. Keš memorija kod MC68020

Deo arhitekture 32-bitnog mikroprocesora MC68020 čini instrukcijski keš čija je organizacija prikazana na slici 6.31.

U toku pribavljanja instrukcije adresnim bitovima A7-A2 se bira jedan od 64 keš ulaza. Stanje na liniji FC2 ukazuje da li se vrši obraćanje supervizorskom ili korisničkom adresnom prostoru.

Bitovi A31-A8 se upoređuju sa 25-bitnim TAG poljem izabranog ulaza. Pribavljena instrukcija je prisutna u kešu kada se javi iparivanje adresa i kada je bit važnosti (V) postavljen. Za slučaj kada postoji keš pogodak instrukciona rwč specificirana stanjem bita A1 se direktno čita iz "on-chip" keša. Kada se instrukcija ne nalazi u keš memoriji (javlja se keš promašaj, bilo zbog toga što ne dolazi do uparivanja TAG-ova, ili zbog toga što je bit važnosti V obrisan), obavljaju se sledeće aktivnosti:

- Upisuje se novi TAG u keš.
- Iz spoljne memorije se pribavlja nova instrukcija i upisuje u keš. Pri ovome se vrši upis u LS i MS reči i postavlja bit važnosti.

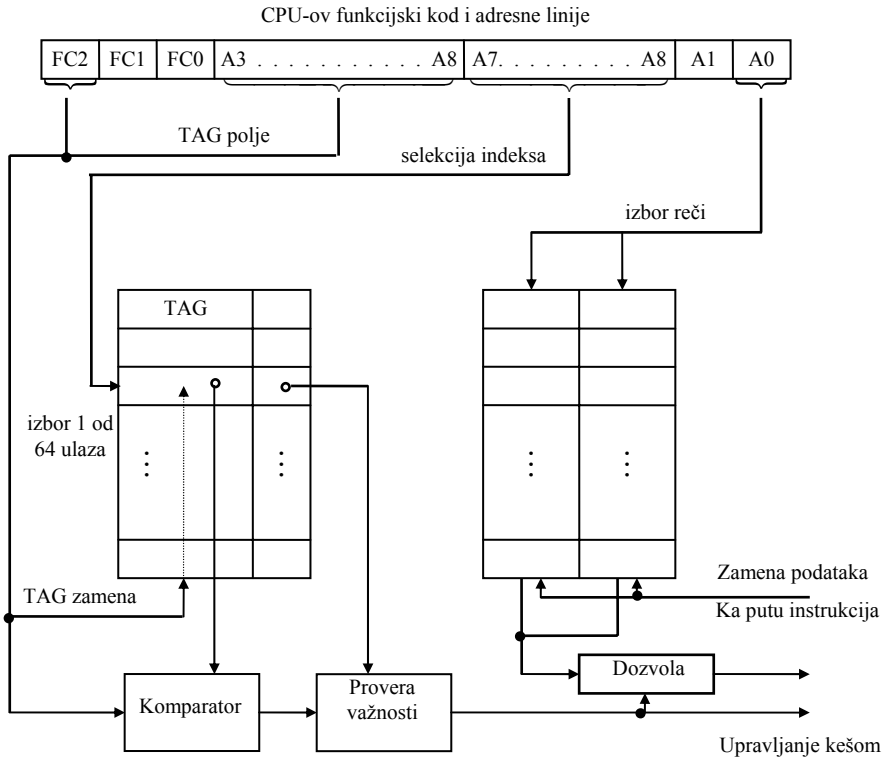
Sistemski programer ima uticaj na upravljanje kešom modifikovanjem bitova CACR-a (Cache Control Register) i modifikovanjem sadržaja CAAR-a (Cache Address Register). CCAR je 32-bitni registar koji upravlja kešom i čuva status keša. Bit pozicije 4-31 se ne koriste i čitaju se kao nula. Ostali bitovi imaju sledeću ulogu (značenje):

EC - dozvola rada keša; brisanjem ovig bita zabranjuje se rad keša a CPU prisiljava da uvek pristupa spoljnoj memoriji.

FC - zamrtavanje keša; zadržava dozvolu rada keša, ali keš promašajima nije dozvoljeno da vrše zamenu važećih keš podataka. Drugim rečima, rad keša je dozvoljen ali se njegov sadržaj ne može promeniti tako da je moguće da se privremeno zaustavi izvršenje programa, obavi neka funkcija upravljanja, a zatim da se nastavi sa prekinutim zadatkom a da se ne izvrši modifikacija internog procesorskog statusa. Ova mogućnost obično mogu da koristi projektanti emulatora sa ciljem da se emulator učini transparentnim za korisnika.

CE - brisanje keš ulaza; omogućava programeru da resetuje bit važnosti (V) koji je pridružen keš ulazu, a koji se bira od strane CAAR-a. Shodno tome, CE bit (kada je resetovan) uslovljava generisanje keš promašaja prilikom obraćanja tom keš ulazu.

CC - potpuno brisanje; ovaj bit se koristi za brisanje svih bitova važnosti (pripadaju keš ulazima) tako da se sadržaj ukupnog keša može poništiti. Ova mogućnost se koristi u toku čišćenja programa od grešaka (debugging).



Sl. 6.31. Organizacija "on-chip" instrukcionog keša kod MC68020.

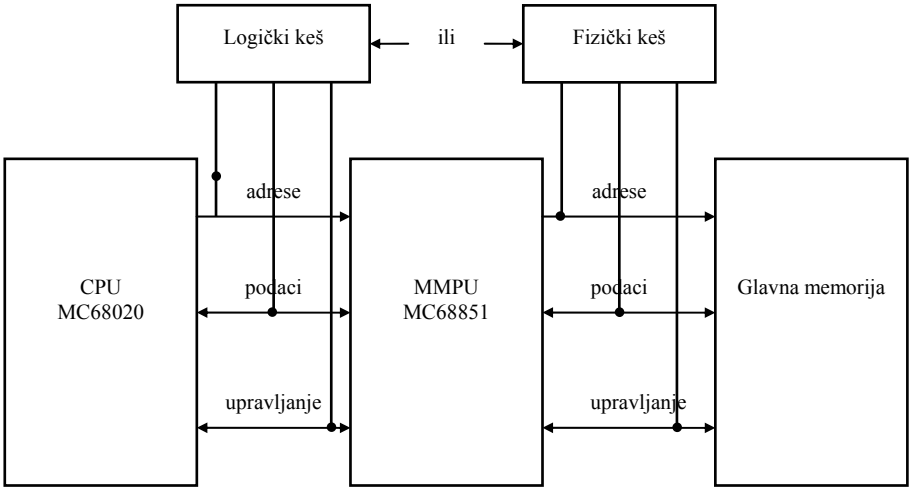
I pored toga što MC68020 ima ugrađenu "on-chip" keš memoriju, da bi se postigle maksimalne performanse mikroračunarskog sistema zasnovanog na ovom mikroporcesoru, potrebno je ugraditi spoljni keš čiji je kapacitet obično 4kB. Pred projektante sistema se sada postavlja jedna dilema: gde postaviti keš memoriju, na logičkoj ili fizičkoj strani MMU-a, kao što je prikazano na slici 6.32.

Nezavisno od toga kakav je izbor učinjen, i kod jednog i kod drugog rešenja javljaju se problemi. Ako se izabere rešenje sa fizičkom memorijom, tada i pored toga što se u sistem mogu ugraditi veoma brzi memorijski čipovi, obavezno se uvode stanja čekanja u toku svakog ciklusa tokom kojeg se vrši pristup memoriji. Ubacivanje stanja čekanja se uvodi zbog vremenskog kašnjenja koje nosi MMU. Kod velikog broja realizacija kašnjenje obično ne predstavlja neki ozbiljan problem, čak i kada se ubaci jedno stanje čekanja. Naime, vreme pristupa memoriji je u tom slučaju jednako četiri, umesto tri, CPU-ova taktna intervala.

Ugradnjom logičke keš memorije projektanti nailaze na problem *alijaze* kada sistem radi u višeprocesnom režimu rada. Naime, različiti zadaci (procesi) u toku izvršavanja koriste iste logičke adrese. Kod logičkog keša se

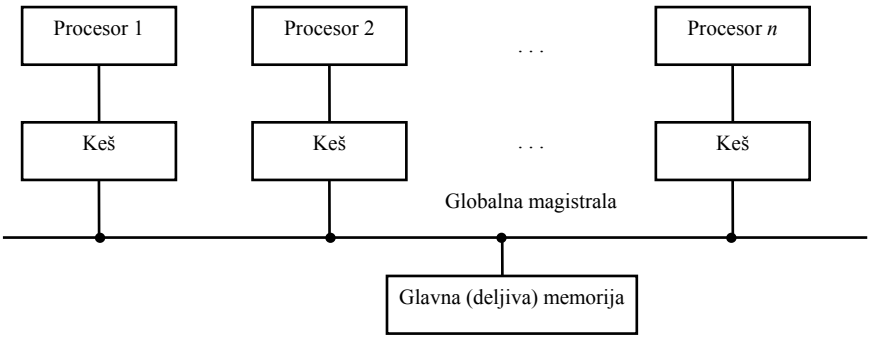
javlja dodatni problem poznat kao *čuvanje ustajalih podataka* a javlja se kada je ovakav tip memorije implementiran kod višeprocesorskih sistema ili gospodara magistrale (misli se na CPU-ove, a ne na DMA kontrolere koji takođe mogu biti gospodari magistrale). Jasno je da se kod ovog rešenja ne javlja problem ubacivanja stanja čekanja koja se uvode zbog toga što je MMU-u potrebno vreme da obavi adresnu translaciju.

Integrisano kolo MC68851 je tako projektovano da može da podržava rad kako logičke tako i fizičke memorije.



Sl. 6.32. Logička ili fizička memorija kod MC68020/MC68851.

Analizirajmo najpre logički keš. Kod multiprocesorskih sistema određene memorijske oblasti su zajedničke (deljive) za veći broj procesora. Svaki od tih procesora (slika 6.33) poseduje svoju programsku memoriju i lokalnu keš memoriju. Drugim rečima, bez ugradnje nekog tipa lokalne memorije (programske i keš) fizički je nemoguće povećati performanse multiprocesorskog sistema koji je organizovan oko jedinstvene (deljive) magistrale.

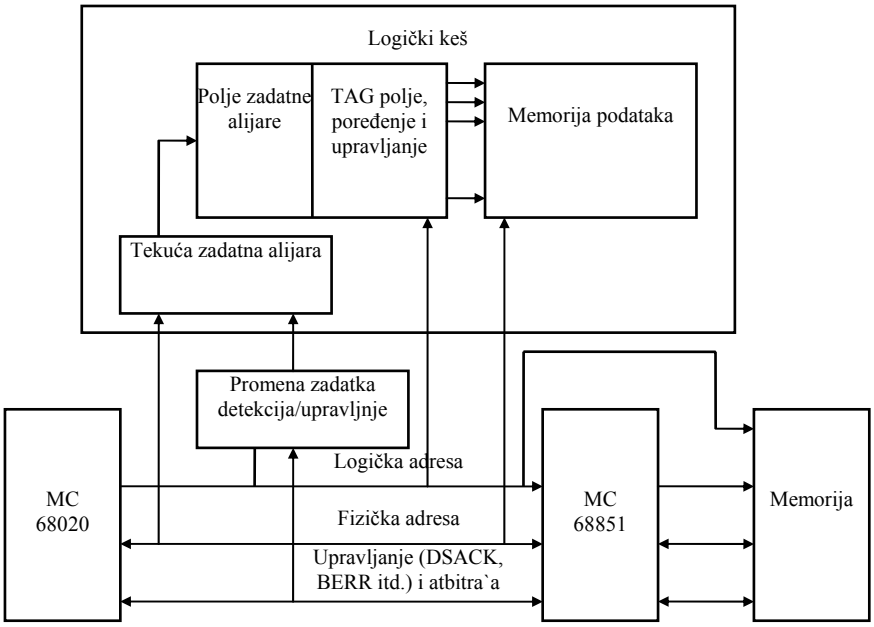


Sl. 6.33. Organizacija multiprocesorskog sistema oko jedinstvene magistrale.

Ali, osnovni problem koji se javlja kod multiprocesorskih sistema je problem *koherencije keša*. Naime, evidentno je da ako se neki deljivi podatak može kešovati, tada i drugi procesor može da mu pristupi i modifikuje početnu vrednost podatka tako da će kešovani podatak, koji se nalazi u keš memoriji prvog procesora, postati nevalidan. Jedno od mogućih rešenja ovog problema, kada se koristi fizički keš, je da se ugradi DMA logika za nadgledanje koje će detektovati pristup alternativnog gospodara magistrale kešovanim podacima. Logika za nadgledanje mora biti u stanju da invalidira keš ulaze koji su bili modifikovani od strane drugih procesora, i na taj način sačuva konzistentnost podataka. Ovo izgleda prihvatljivo kada je keš memorija instalirana u fizičkom adresnom prostoru, gde svi sklopovi koji mogu pristupiti istom operandu generišu istu adresu (kada pristupaju deljivom operandu).

Kod aplikacija gde je keš memorija instalirana u logičkom prostoru, nepraktično je obavljati obrnuto fizičko→logičko adresno preslikavanje od strane logike za nadgledanje magistrale.

Zadovoljavajuće rešenje, za obe varijante instaliranja keš memorije, se dobija kada operativni sistem odredi stranice koje se kao globalno deljive ne mogu smeštati u keš memorije. Keš inhibitorskim bitom, koji je deo opisivača stranice, obezbeđuje se da MC68851 generiše keš inhibitorski signal u trenutku kada se pristupa toj stranici. Kod višeprocesnog režima rada (multitasking), razdvajanje procesa u logičkoj keš memoriji realizuje se pomoću polja "Task Alias" koje MC68851 koristi za razdvajanje procesa u okviru ugrađenog translacionog keša. Translacioni keš je deo MC68851. Ovu vrednost CPU može čitati i upisati u TAG adresno polje logičke keš memorije (slika 6.34). Tekuća "Task Alias"-a se smešta u hardverski registar, koja se ažurira nakon svakog upisa u CRP registar. (CRP-CPU Root Pointer je adresno translacioni upravljački registar koji pripada čipu MC68851).



Sl. 6.34. Logička keš memorija kod tipičnog virtuelnog memorijskog sistema.

U toku narednih pristupa keš memoriji ova vrednost se upoređuje sa vrednošću TA koja je deo TAG polja svakog ulaza. Takođe, kod upisa u CRP, vrši se provera da bi se ustanovilo da li je novom procesu (zadatku) bio definisan "root" pokazivač. Ako se ovo desi, tada se ulazi koji odgovaraju ovom TA-u automatski poništavaju u translacionom kešu čipa MC68851, ali se oni takođe moraju poništiti u logičkom kešu.

Mehanizam zaštite je podjednako važan kako kod logičkog tako i kod fizičkog keša. Zbog toga je od vitalne važnosti da metodi zaštite važe nezavisno od toga da li se:

1. pristupa keš memoriji na fizičkom ili virtuelnom nivou;
2. pristupa keš memoriji u korisničkom ili supervizorskom načinu rada;
3. pristupa keš memoriji u toku operacije upis ili čitanje.

Brigu o zaštiti uglavnom vodi MC68851 koji pored funkcije adresne translacije, uvek proverava važnost logičke adrese sa vrednošću koja je upisana u njegov opisivač (opisivač je deo translacione keš memorije čipa MC68851). Imajući u vidu da MC68851 vrši ovakav tip provere za prekršaje kažemo da se lako i efikasno otklanjuju.

Jedan drugi aspekt na koga smo jedanput ukazali odnosi se na mogućnost nezavisnog izvršavanja operacija sa logičke i sa fizičke strane čipa MC68851. Na primer, MC68020 može da izvršava instrukcije iz logičkog keša, a drugi allternativni DMA gospodar da obavlja prenos u fizičkom adresnom prostoru. Ovakav način rada može u značajnoj meri da poveća efikasnost onih sistema kod kojih postoji veći broj potencijalnih gospodara magistrale.

Analize koje se odnose na realizaciju fizičke keš memorije ukazuju da se glavni problem sastoji u tome što je potrebno ugraditi veoma brze memorijske čipove, kako se ne bi ubacivao veći broj stanja-čekanja. Par koga čine MC68020 i MC68851 nudi rešenje kojim se olakšava ovaj problem.

Spoljni uređaj (u ovom slučaju keš) može signalizirati završetak ciklusa u toku koga se vrši prenos, aktiviranjem signala DSACK-u pre nego što je validiran pristup. (DSACK0 i DSACK1 su ulazno-izlazni signali čipa MC68851). Ovakvim načinom rada, obezbeđeno je dodatno vreme da TAG logika za potvrđivanje važnosti obavi svoj zadatak. U slučaju da se javi greška (jedan ciklus kasnije) BEER (Bus Error signalom koji je ulazno-izlazni signal čipa MC68851) se invalidira pristup.

Konačno, da li ugraditi fizičku ili logičku keš memoriju zavisi od velikog broja projektantskih zahteva. Jasno je da se kod konačnog rešenja mora napraviti dobar kompromis između: vreme pristupa/cene/obima keš memorije - kako bi se postigla najveća sistemska propusnost, najbolja propusnost magistrala i najveća efikasnost, tj. najveća stopa pogodaka.