3. Neki komercijalni procesori

3.1. Karakteristike i performanse računala

Tržište računala u posljednjih pedesetak godina naraslo je praktički od nule do veličine od nekoliko stotina milijardi dolara godišnje i pokazujući tendenciju daljnjeg velikog rasta. Računala su postala dio svakodnevnog života zahvaljujući svojim mogućnostima koje su dostupne za relativno prihvatljivu cijenu. Performanse računala, njihovo vrednovanje, odnos cijena/performanse, kompatibilnost s prethodnim generacijama neke su od tema koje će se prve razmotriti.

3.1.1. Smanjenje cijene i povećanje performansi

Projektanti računala i inženjeri računarstva stalno su zaokupljeni cijenom svojih proizvoda kao i omjerom cijena/performanse. Cijena i performanse komponenata u domeni implementacije određuju performanse uređaja razvijenog da se s određenim karakteristikama prodaje za određenu cijenu. Cijena određenih komponenata koje tvore računalo opada strahovitom brzinom. Za usporedbu elektroničke cijevi korištene u prvim računalima između 1940. g. do 1950 g. koštale su nekoliko dolara po komadu. Diskretni tranzistori koji su ih zamijenili u 60.tim godinama koštali su manje od dolara po komadu. Ipak stvarni pad u cijeni po komponenti počinje s pojavom integriranih sklopova.

Integrirani sklop, s tisućama tranzistora, sadrži više funkcionalnih blokova. Cijenu integriranog sklopa ne čini broj komponenata koje on sadrži već trošak njegovog razvoja koji se mora amortizirati već u ranoj fazi njegovog izlaska na tržište. U kasnijim fazama cijena istog integriranog znatno pada i nije neuobičajeno da podnožje košta više od samog integriranog sklopa. Tako npr. tipkovnica, a naročito monitor koštaju više nego matična ploča računala.

S gospodarstvenog gledišta u ovom području mnogo toga se drastično promijenilo. Cijena integriranog sklopa, odnosno silicija je zanemariva ili pak besplatna. Ono što čini njegovu cijenu je znanje potrebno za njegov razvoj kao i informacija sadržana u njemu. Slično vrijedi i za programske proizvode. Cijena CD ROMa je oko $2, ali ako je na njemu zapisan MS Office tada je njegova cijena $500. Grubo govoreći uskoro se može očekivati da će se platiti operacijski sustav i aplikacije a sklopovlje će se dobiti besplatno. Ipak do tada projektanti računala i sklopova suočavaju se s kompromisom između cijene i performansi sklopova i uređaja koje projektiraju i izrađuju. U računalnim sustavima uobičajeno je određivati performanse računala pomoću određenih programa ili niza programa koje se na njima izvode.

3.1.2. Kompatibilnost s prethodnim generacijama (*Upward Compatibility*)

IBM 360 računala je su prva porodica računala sa svojstvom da programi razvijeni za prethodna, starija računala mogu se bez problema izvoditi na novijim moćnijim računalima iz te porodice. Ovo svojstvo potaklo je mnoge potrošače da brže mijenjaju svoja računala. Razlog je bio jednostavan budući su oni znatno više investirali u programe nego u sklopovlje. Nabavka sklopovlja, računala, s boljim performansama bez potrebe da se nabavljaju ili razvijaju novi programski paketi sigurno je neusporedivo opravdanija u usporedbi sa slučajem kada je potrebno nabaviti ili ponovo razviti čitavu programsku podršku. Stoga proizvođači računala projektiraju računala na način da imaju bolje performanse od prethodnih ali da je moguće potpunu ili uz manje modifikacije koristiti

programsku podršku razvijenu za prethodne generacije. Kompatibilnost s prethodnim generacijama podrazumijeva:

- Binarna kompatibilnost. Binarne datoteke generirane na prethodnim verzijama računala mogu se nesmetano izvoditi i na novim računalima. Obično je dobro ponovno prevesti izvorne programe kako bi poboljšanja novih računala došla do izražaja.
- Binarna kompatibilnost uz određene ograde. Binarne datoteke s prethodnih verzija mogu se nesmetano izvoditi na novim verzijama računala uz uvjet da su prilikom pisanja izvornog koda poštovana određena pravila.
- Kompatibilnost izvornog koda (*source code compatibility*). Izvorni kod napisan za prethodne verzije računala potrebno je samo ponovo prevesti te se novi binarni oblik programa može nesmetano izvoditi na novoj verziji računala.
- Kompatibilnost ostvarena emulacijom koda. Proizvođači računala u suradnji s programerskim firmama razviju programe koji prevode binarni kod starih u binarni kod novih verzija računala. Ovaj proces se odvija tijekom izvođenja programa. Izvođenje programa na ovaj način u pravilu je znatno duže. Prednost je da korisnik ipak može koristiti stare verzije programa dok se ne razviju nove.

3.1.3. Performanse računala

Procjena performansi računala varira od krajnje jednostavnih kvalitativnih ("Računalu je potrebno 15 sekundi da starta MS Word") do ozbiljnih objektivnih kvantitativnih testova temeljenih na programima projektiranim posebno za ispitivanje performansi računarskih sustava. Općenito testiranje jednog sustava započinje definiranjem prosječnog ili očekivanog opterećenja računala (*workload*). To je skup programa ili koji se na računalu izvode i čije vrijeme izvođenja se mjeri. Idealno je kada se radi o upravo onim programima koje korisnik namjerava koristiti. Tada je moguće najobjektivnije dobiti odgovor na pitanje koje računalo korisniku za tu aplikaciju ili skup aplikacija najbolje odgovara. Naravno u praksi je situacija znatno složenija pa se zato definiraju programi za ispitivanje i usporedbu performansi računala (*benchmark programs*). Oni simuliraju očekivano opterećenje računala.

Vrijeme izvođenja, frekvencija takta, broj taktova po naredbi (*Clocks per Instruction CPI*). Mjera performansi računala je vrijeme. Računalo koje u kraćem vremenu obradi zadani program ima najbolje performanse za taj tip opterećenja. Vrijeme izvođenja programa ovisi o mnogim čimbenicima. Obično se mnogi od njih previde kao npr. čekanje na obavljanje ulazno/izlaznih operacija koje ne ovisi o brzini procesora. Vrijeme potrebno da procesor izvede zadani program određeno je izrazom:

vrijeme izvođenja = IC * CPI * T

gdje je IC - broj naredbi programa, CPI – prosječan broj taktova po naredbi, T – trajanje takta. Procesor s kraćim taktom, manjim brojem taktova po naredbi, te manjim brojem naredbi koje su potrebne da bi se obavila zadana obrada ima bolje performanse. Nedostatak ovog izraza je što računa s prosječnim brojem taktova po naredbi. Naime u obradi dominiraju neki tipovi naredbi (npr. prebacivanje podataka), dok se drugi tipovi ne moraju uopće koristiti (npr. dijeljenje dva realna broje). Ako izvođenje prvih naredbi traje znatno kraće od izvođenje drugih znači da će i sam program manje trajati. Zato je važno

uzeti u obzir stvarnu distribuciju (raspodjelu) naredbi za određeni tip obrade. Također, zbog različitosti arhitektura procesora te različitosti njihovih skupova naredbi neke obrade na jednom procesoru zahtijevaju veći broj naredbi nego na drugom, a što može biti sasvim obratno za neki drugi tip obrade. Zato je moguće dobiti različite rezultate primjenom različitih ispitnih programa.

MIPS (*Millions Instruction per Second*). Jedno uobičajeno, ali dosta neprecizno mjerilo performansi računala je koliko milijuna naredbi računalo može izvesti u sekundi. Ovu vrijednost jednostavno se dobiva izvođenjem nekog programa ili skupine programa s poznatim brojem naredbi te mjereći vrijeme potrebno da se on ili oni izvedu. Praktično postoji više razloga nepouzdanosti ove mjere. Možda najvažnije je da mogućnosti naredbi znatno variraju od procesora do procesora. Npr. procesori koji koriste naredbe s tri adrese za aritmetičku naredbu zahtijevaju veći broj taktova (npr. dohvat naredbe 3 takta + dohvat operanada 2 takta + proračun i pohrana rezultata 1 takt = 6 taktova) u usporedbi s istom naredbom RISC procesora temeljenog na spremnicima opće namjene koja se praktički izvodi u jednom ciklusu (npr. 1 ciklus = 4 takta). U stvarnosti drugi procesor nije 6/4 puta brži od prethodnoga jer je za istu obradu ponekad i potrebno dohvatiti operande iz memorije i pohraniti ih u spremnike što dodatno produžuje trajanje izvođenja za četiri ciklusa. Ovo bi značilo da je drugi procesor realno sporiji 20/6 puta. Općenito broj naredbi za istu obradu kod RISC procesora je oko 20% veći nego broj naredbi kod CISC procesora. RISC procesori naprotiv rade i s nekoliko puta većom frekvencijom takta nego CISC procesori. MIPS kao mjerilo bilo je popularno u kasnim 70tim i ranim 80tim. U to vrijeme računalo VAX 11/780 uzimalo se za referencu, a ostala računala uspoređivala su se s njim kao MIPS relativno prema VAXu.

FLOPS (*Floating-Point Operations per Second*) i MFLOPS. Potrebe mnogih proračuna su obično takve da se češće barata s realnim (brojevi s pomičnim zarezom), a rjeđe s cijelim brojevima. Kod takvih proračuna mjera performansi računala je broj naredbi s realnim brojevima u sekundi koje ono može izvesti. FLOPS predstavlja za ove slučajeve puno realnije mjerilo performansi računala iz razloga što su operacije s realnim brojevima znatno složenije od operacija s cjelobrojnim veličinama te ukoliko procesor nema prilagođeno sklopovlje za takvu obradu obrada je znatno složenija i duže traje.

Whetstones i Dhrystones. Whetstone test program je prvi program za usporedbu performansi računala. Razvijen je sredinom 70tih i dobio je naziv prema Whetston Algol programu prevodiocu razvijenom u gradiću Whetstone u Engleskoj. Ovaj test program razvijen je prvenstveno da ispituje performanse obrade nad realnim brojevima. Performanse su davane u MWIPS (*Millions Whetstone instruction per Second*). Izvorno je pisan u programskom jeziku ALGOL 60, a naknadno je prenesen i na FORTRAN, Pascal i C.

Dhrystone je test program razvijen za ispitivanja performansi računala kod obrade nad cjelobrojnim veličinama. Isto kao Whetstone, on je mali program svega stotinjak naredbi u višem jeziku, odnosno 1 do 1.5 kB koda.

Oba test programa su mali programi koji su stimulirali programere programa prevodilaca da do-optimiraju svoje proizvode kako bi test programi pokazali bolje performanse od stvarnih.

SPEC – (*System Performance Evaluation Cooperative*) Zadruga za procjenu performansi sustava. Tijekom vremena pojavio se trend koji napušta sintetičke test

programe i okreće se ka programima u općoj uporabi. Konzorcij SPEC, koji se sastoji od predstavnika mnogih kompanija koje proizvode računala, ustanovljen je 1987 s ciljem definiranja standarda za test programe. Skup test programa koje je odredio SPEC sastoji se od programa prevodilaca, Booleovih minimizacijskih programa, tabličnih kalkulatora te niza drugih programa koji ispituju brzinu izvođenja. SPEC je registriran kao:

> *SPEC je neprofitabilna korporacija formirana s ciljem da uspostavi, održava i te nadgleda standardizirani skup test programa koji mogu biti izvođeni na najnovijim generacijama računala. SPEC razvija skup test programa koji imaju zadaću mjeriti performanse računala. Ovaj skup programa je prenosivi, dobro ispitani programski paket koji se sastoji od izvornog koda i alata. Oni su dostupni javnosti uz naknadu troška razvoja i administacijskih troškova. Prema licencnim pravima, SPEC članovi i potrošači suglasni da izvode test programski paket te objavljuju i dokumentiraju rezultate mjerenja.*

Danas dostupni SPEC test programi doživjeli su svoju posljednju preinaku 1992. kada su programskom paketu dodani test programi za cjelobrojnu aritmetiku i aritmetiku pomičnog zareza, SPECint92 i SPECfp92. Iako pažljivo odabrani test programi ipak su pojedini proizvođači računala i programske podrške posebno optimirali svoje programe kako bi iskazali bolje performanse od stvarnih. Neki od proizvođača su išli toliko daleko da su u okviru programa prevodilaca ubacivali posebne zastavice koje su imale samo svrhu da poboljšaju ocjenu prema SPECu bez stvarnih poboljšanja performansi računala i programa. Trenutna nastojanja SPECa su da onemoguće spomenuta nastojanja specificiranjem dodatnih pravila o ponašanjima programa prevodilaca. U novije vrijeme SPEC uvodi i čitav niz test programa namijenjenih ispitivanju klijent-poslužitelj (*client-server*) arhitekture.

Sve navedene metode mjerenja performansi računala nastoje odgovoriti korisniku koje je računalo najbolje za obradu koja je njemu potrebna. Korisne informacije dobivaju se i od korisnika koji su uspoređivali objavljene rezultate s test programima od strane proizvođača sa stvarnim poboljšanjima koje su oni uočili izvođenjem svojih aplikacija. Tako se najobjektivnije dobiva informacija koliko su poboljšanja u performansama, specificirana od strane proizvođača, u suglasnosti s poboljšanjima koja uočava krajni korisnik.

3.2. RISC ili CISC

Potrebno je naglasiti da je razlika između računala sa složenim skupom naredbi i računala sa smanjenim skupom naredbi u konceptu ili filozofiji projektiranja računala prije nego u svojstvima arhitekture. Naime današnji RISC procesori novih generacija imaju stotine naredbi koje nisu sasvim jednostavne nego čak i relativno složene u svakom smislu.

3.2.1. CISC računala

CISC procesori nisu rezultat određene filozofije u pristupu projektiranja računala. Naprotiv, oni su rezultat težnji projektanata da u procesoru inkorporiraju što više mogućnosti kao npr. adresnih modova, tipova naredbi i sl.

Opće karakteristike CISC procesora. Kod CISC procesora ISA se projektira s ciljem da naredbe imaju što veće mogućnosti. Kao rezultat CISC procesori imaju široki opseg adresnih modova, 14 kod Motorole 68 000 i 25 kod Motorole 68 020. Dalje ovi procesori dozvoljavaju korisniku da operandi i rezultat budu proizvoljno pohranjeni kako u memoriji

tako i u spremnicima opće namjene. Tako VAX ima naredbe s 0 do 3 adrese. VAX naredba za zbrajanje može specificirati 2 ili 3 lokacije koje se nalaze bilo u memoriji ili u spremnicima opće namjene. Rezultat ovakve arhitekture su naredbe različitih dužina i vremena izvođenja.

Povijesne činjenice. Veće mogućnosti naredbi naišle se na jako povoljan odaziv kod programera iz razloga što je moguće s manje naredbi realizirati zadanu obradu. Cijena memorije u vrijeme pojave mikroprocesora M6800 (1975) iznosila je $500 za 16 KB RAMa, a tvrdi disk od 40 MB koštao je $55000. Pet godina kasnije kad se na tržištu pojavio 16 bitovni procesor M68000 cijena 64 KB RAMa je još uvijek iznosila nekoliko stotina dolara, a tvrdi disk od 10 MB je koštao $5000. Zbog cijene memorije programeri su težili da njihov kod i podaci zauzimaju što manje prostora.

Novoprojektirani procesori nastojali su zadržati programsku kompatibilnost sa svojim prethodnicima. Prirodno je da njihove povećane mogućnosti rezultirale su u složenijem sklopovlju. Kompatibilnost sa prethodnim generacijama dodatno je povećavala tu složenost.

Nadalje, projektanti postali su svjesni raskoraka između simboličkog i viših jezika. Pojavilo se razmišljanje da se taj raskorak može uvođenjem složenijih naredbi kao i složenijih adresnih modova. Ipak projektanti programa prevodilaca odbijali su korištenje tih složenih naredbi i u glavnom ostajali su u okviru prvobitnih skupova naredbi. Istraživanja koja je 1971 proveo Donald Knuth i 1982 David Patterson pokazala su da 85 % su naredbe koje varijablama pridodjeljuju vrijednost, uvjeti i pozivi procedura. Od naredbi koje mijenjaju vrijednost varijablama 80% su samo jednostavne naredbe za prebacivanja (MOVE), a tek 20% aritmetičke i logičke operacije. Također kako su proizvođači dodavali nove opcije svojim procesorima suočili su se s problemom da dodatna složenost sklopovlja onemogućava povećavanje frekvencije takta odnosno veću brzinu izvođenja naredbi. Složene naredbe i adresni modovi u suprotnosti su s većim frekvencijama zbog velikog broja mikrokoraka koje je potrebno izvesti kako bi se oni realizirali.

Do nedavno je cijena memorije opadala 40% godišnje. Tako je brzo dostignuta razina na kojoj cijena memorije nije više kritična pa programeri svoje strateške ciljeve usmjeravaju prema brzini izvođenja dok memorijski zahtjevi programa postaju nebitni.

3.2.2. Prijelaz s CISC na RISC procesore

Već od prvih računala njihovi projektanti težili su povećanju brzine izvođenja naredbi, odnosno povećanju brzine obrade. Uvođenjem paralelizma u radu procesora moguće je istovremeno obrađivati više naredbi. Time se povećava propusna moć procesora, odnosno broj naredbi koje on može obraditi u jedinici vremena. Danas su uobičajena rješenja preddohvat naredbe(*prefetching*), cjevovod (*pipeline*) i superskalarne operacije.

U modelu obrade naredbe procesor u prvom trenutku dohvata naredbu, a zatim je izvodi. Nakon što se izvede naredba ide se u dohvat i izvođenje sljedeće naredbe. Kod modernijih procesora ove dvije aktivnosti se preklapaju. Brzina obrade može se značajno povećati ukoliko se sljedeća naredba dohvati već za vrijeme izvođenja tekuće naredbe (*prefetching*). Prvi 16 bitovni mikroprocesor Intel 8086 imao je red za preddohvat naredbe veličine 6 okteta. Preddohvat naredbe predstavlja preteču cjevovoda.

Tako su praktički prvi mikroprocesori primjenjivali primitivan oblik cjevovoda. Jednostavno rečeno cjevovod podrazumijeva dohvat i početak izvođenja sljedeće naredbe prije nego što je završila sljedeća naredba. Suvremeni procesori proces izvođenja sljedeće naredbe privode skoro kraju u trenutku završetka tekuće naredbe. Tako npr. Motorola PowerPC 601 ima 20 zasebnih odsjeka u kojima se odvijaju različite aktivnosti različitih naredbi.

Superskalarno računanje podrazumijeva istovremeno izvođenje više naredbi. Tako npr. Motorola PowerPC 601 ima tri nezavisne obradne jedinice: za cjelobrojne brojeve, brojeve s pomičnim zarezom i za grananja. Ove jedinice mogu istovremeno izvoditi svaka svoju naredbu. Kod ovih procesora događa se da se naredbe izvode i van redoslijeda određenog programom. Tako npr. vremenski kraća naredba koja slijedi dužu naredbu može biti izvedena prije njene prethodnice.

Projektanti CISC računala primjenjivali su navedene tehnike za povećanje brzine obrada, ali duge naredbe koje zahtijevaju dohvat operanada pomoću složenih adresnih modova koji koriste složenu aritmetiku za proračun efektivne adrese značajno su otežavali njihovu realizaciju i umanjivali njihove efekte. Nadalje, složene naredbe su u suprotnosti s većim frekvencijama takta u usporedbi s jednostavnim naredbama. Tako su RISC procesori projektirani kako bi učinkovito koristili prednosti cjevovoda, superskalarnog računanja, malih brzih međumemorija (*cach*) razvijenih u doba CISC procesora.

3.2.3. Filozofija projektiranja RISCa

Naziv RISC (*Reduced Instruction Set Computer*) računalo sa smanjenim skupom naredbi, ukazuje na smanjenje i složenost naredbi procesora. U stvarnosti postoji niz strategija kojima se povećava učinkovitost cjevovoda, superskalarnog računanja i brzih međumemorija. Naravno, stvarni RISCovi koriste samo dio mogućih rješenja.

Jednako trajanje izvođenja svih naredbi. Ovaj jednostavan koncept praktički je najvažniji temelj u projektiranju RISCa. Prve definicije RISCa podrazumijevale su da se sve naredbe izvode u jednom ciklusu. Realizacija cjevovoda je jednostavnija ukoliko se u svakom taktu starta po jedna naredba. Kako vrijeme izvođenja ovisi o **propusnosti** procesora (koliko naredbi se izvede u jedinici vremena), a ne o vremenu izvođenja pojedine naredbe, izvođenje jedne naredbe po ciklusu je opravdan projektantski cilj. On se može ostvariti pojednostavljenjem naredbi ali ne i povećanjem broja taktova po ciklusu.

Jednaka dužina svih naredbi. Ako se izvodi jedna naredba po ciklusu i ako su naredbe jednostavne tada je logično da su sve naredbe i fiksne dužine, obično jedna riječ. Ta riječ specificira sve o naredbi, operacijski kod, gdje su smješteni operandi, gdje pohraniti rezultat i gdje se nalazi sljedeća naredba.

Pristup memoriji preko *load* i *store* naredbi. Ako je naredba dužine jedne riječi, odnosno 32 bita, te kako dohvat operanada iz memorije zahtijeva dodatno vrijeme, projektanti RISC računala ograničavaju se na izvođenje svih operacija nad operandima pohranjenim u spremnicima procesora. Pristup operandima u memoriji ograničen je na operacije njihova dohvata, **load**, i pohrane, **store**. Ova čvrsta podjela između pristupa operandima u memoriji i njihove obrade minimizira broj pristupa memoriji od strane procesora, odnosno reducira opterećenje vanjske sabirnice. Ovim se minimizira kašnjenje pri dohvatu sljedeće naredbe uzrokovano dohvatom operanada prethodne naredbe.

Jednostavni adresni modovi. Složeni adresni modovi zahtijevaju više taktova budući se mora provesti više aritmetičkih operacija. RISC procesori obično su ograničeni na samo dva adresna moda, indirektno pomoću spremnika (*register indirect*) i indeksno. Indeks može biti pohranjen ili u spremniku ili konstanta koja je sastavni dio naredbe.

Manje jednostavnih operacija. Jednostavne naredbe impliciraju kraće vrijeme izvođenja (manji broj taktova) budući da je za njihovu realizaciju potreban znatno manji broj radnji. Tako npr. RISC procesor neće imati naredbu POLY koju ima VAX računalo kojom se izračunavaju korijeni polinoma Hornerovom metodom. Svaka složena naredba može se podijeliti na niz jednostavnih naredbi. Projektanti su učestalo suočeni s dilemom dodati neke složenije naredbe koje imaju za posljedicu povećanje složenosti upravljačke jedinice te moguće povećanje broja taktova po ciklusu.

Odgođeni pristupi memoriji i grananja. Postoji jedna poslovica koja kaže:"Ako ne možeš pobijediti mijenjaj pravila". Kod određenih RISC arhitektura dohvat operanada (**load**), pohrana rezultata (**store**) i grananja zahtijevaju više ciklusa za izvođenje. Kod prve dvije naredbe produženje trajanja naredbe posljedica je potrebe ponovnog pristupa memoriji, dok kod grananja zbog dohvata naredbe koja je određene rezultatom ispitivanja. Zbog cjevovoda ovo praktično znači da dvije naredbe istovremeno završavaju (prva traje dva ciklusa, a sljedeća koja počinje ciklus kasnije i traje samo jedan ciklus). Kako bi se osiguralo da naredba koja slijedi neku od navedenih naredbi ne bi ovisila o njenom rezultatu uvodi se posebna naredba **nop** koja ništa ne radi. Ona samo zauzima mjesto u nizu naredbi i troši vrijeme kako bi se omogućilo prethodnoj naredbi da završi i dostavi rezultat koji mogu koristiti sljedeće naredbe. Programeri u simboličkom jeziku moraju poznavati način izvođenja svake pojedine naredbe te po potrebi ubacivati **nop** naredbu. Programi za prevođenje viših jezika imaju ugrađene automatizme za prepoznavanje i rješavanje spomenutih situacija.

Preddohvat naredbe i spekulativno izvođenje. Prednosti preddohvata naredbi naročito dolazi do izražaja kod RISCa. Kada su sve naredbe jednake dužine jednostavnije je analizirati naredbu kako uđe u cjevovod te zaključiti da li se radi o naredbi aritmetičkoj ili logičkoj naredbi, pristupu memoriji ili grananjima. Ukoliko se zaključi da se radi posljednjima može se odmah pristupiti dohvatu operanda ili naredbe koja slijedi ukoliko dođe do grananja. Ovim pristupom praktički ne dolazi do kašnjenja zbog grananja ili pristupa memoriji. Procesor ima više jedinica koje mogu istovremeno obavljati dijelove obrade naredbi, a kao je moguće znati adresu mogućeg grananja unaprijed, procesor započima obradu naredbe koja je rezultat grananja bez obzira da li je zadovoljen uvjet grananja (spekulira o rezultatu grananja). Rezultat dijela njene obrade će se odbaciti ukoliko se pokaže da nije potrebno izvesti grananja.

Značenje programa prevodioca. RISCovi imaju kraće naredbe koje izvode samo jednostavnije operacije. Razbijanje složenih naredbi CISCa na niz jednostavnih naredbi uspostavlja veću ovisnost između naredbi. Kako bi se što bolje iskoristile karakteristike RISCova potrebno je da se izvođenje naredbi izvodi i izvan redoslijeda određenog programom kad god je to moguće. Ovaj zahtjev ozbiljno opterećuje programera jer on ili ona ne mora samo paziti na ispravnost programa nego i da kod bude optimiran za određeni procesor u smislu da u potpunosti iskoristi karakteristike njegove arhitekture i sklopovlja (preddohvat naredbi, cjevovod, odgođen pristup memoriji i grananja, spekulativno izvođenje). Većina programa prevodilaca optimira kod da navedeni zahtjevi

budu ispunjeni. To je razlog da se RISCova programiraju obično u višem jeziku, a znatno rjeđe u simboličkom.

PowerPC. PowerPC je moderni procesor koji je primjer do koje razine se koriste opisane tehnike. PowerPC može izvesti preko 200 naredbi, od kojih su neke i složene. Npr. naredba **fctiwz**, pretvori broj s pomičnim zarezom u cjelobrojni sa zaokruživanje prema nuli, nije sasvim jednostavna naredba. Ovaj procesor pristupa memoriji samo pomoću naredbi **load** i **store**, naredbe su sve iste dužine, postoji manji broj adresnih modova te koristi preddohvat naredbi i cjevovod. Sve ovo ima za cilj da se izvede jedna naredba po ciklusu. Ovaj procesor ne koristi tehnike odgođenog pristupa memoriji i grananja, ali koristi spekulativno izvođenje grananja. Tako PowerPC koristi samo dio značajki RISCova.

3.3. CISC mikroprocesor: Motorola MC68000

Nakon što su opisane razlike između CISC i RISC procesora slijedi detaljan opis CISC mikroprocesora Motorola MC68000.

Motorola je postigla veliki uspjeh sa svojim 8 bitovnim mikroprocesorom M6800 koji se pojavio 1975, te poboljšanim sljedbenikom M6809 koji se pojavi nakon nekoliko godina. Prvi 32 bitovni mikroprocesor MC68000 pojavio se 1979. Radilo se o CISC procesoru sa 14 adresnih modova i velikim brojem naredbi. Nakon ovoga procesora Motorola je proizvela i druge procesore koji pripadaju ovoj porodici počevši od MC68010 koji je ima identičnu internu arhitekturu ali vanjsku 8 bitovnu podatkovnu sabirnicu u usporedbi sa 16 bitovnom vanjskom podatkovnom sabirnicom MC68000, pa do MC68040 koji uz mnoga poboljšanja koristi i 32 bitovnu vanjsku podatkovnu sabirnicu.

Ne postoji standardizirana definicija o veličini riječi procesora, ali je uobičajeno da se procesor definira kao n-bitovni procesor ako je većina njegovih internih podatkovnih sabirnica n-bitovna. Tako su svi procesori iz porodice MC68000 32 bitovni procesori budući da je većina operacija nad 32 bitovnim podacima, ali vanjske podatkovne sabirnice su od 8 do 32 bita.

Proces opisa ISA procesora, koji je već primijenjen u prethodnim poglavljima, opisan je sljedećom tablicom:

Memorija: struktura spremnika u računalu
Spremnici procesora
Organizacija glavne memorije

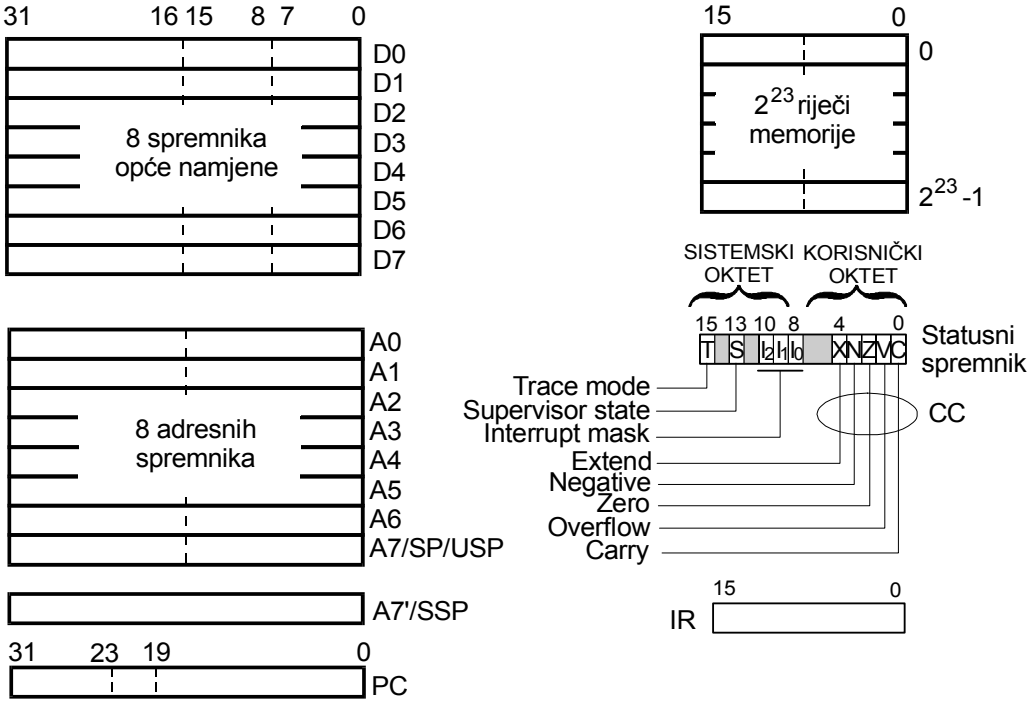
Formati i njihovo tumačenje: značenje pojedinih polja spremnika
Tipovi podataka
Formati naredbi
Interpretacija adresa naredbi

Interpretacija naredbi: što rade naredbe
Ciklus dohvati i izvedi
Obrada posebnih stanja i prekida

Izvođenje naredbe: ponašanje pojedinih naredbi
Grupiranje naredbi po vrstama
Aktivnosti koje izvode pojedine naredbe

3.3.1. Arhitektura CPUa i memorije

Na sljedećoj slici prikazani su spremnici procesora MC68000.



Slika pokazuje da je ovaj procesor temeljen na jednoj varijanti procesora sa spremnicima opće namjene. Razlika od klasične arhitekture sa spremnicima opće namjene je da se razlikuju spremnici za operande D0 – D7 i spremnici za adrese A0 – A7. Spremnici procesora mogu se podijeliti na:

- 32 bitovno programsko brojilo PC. Kako MC68000 ima 24 bitovnu adresnu sabirnicu samo se koriste posljednja 24 bita PC dok se bitovi 24 – 31 zanemaruju.
- 16 bitovni spremnik naredbe IR u koji se pohranjuje prvih 16 bita naredbe. Pri dohvatu naredbe PC se automatski poveća za 2 kako bi pokazivao na sljedeću naredbu. Neke naredbe su duže od 16 bita. U tom slučaju dohvata se dodatnih 16 bita te se dodatno inkrementira PC kako bi pokazivao na sljedeću naredbu ili na sljedeću riječ iste naredbe. Kada se pročita i posljednja riječ naredbe PC pokazuje na prvu riječ sljedeće naredbe.
- Spremnici opće namjene za pohranu operanada i adresa. D0 – D7 sadrže operande, a A0 – A7 adrese koje koristi tekuća naredba. Spremnik A7 posebno se koristi kao kazalo stoga, dok A7' kao kazalo stoga u nadzornom (*supervisor*) režimu rada.
- Statusni spremnik podijeljen je na sistemski i korisnički oktet. Sastoji se od niza jednobitovnih zastavica (*flag*) koje ukazuju na određena stanja procesora:

  T: Mod praćenja (*trace mode*)
  S: Nadzorni ili korisnički mod rada (*supervisor mode*)
  I_2 – I_0: Maska prekida (*interrupt mask*)

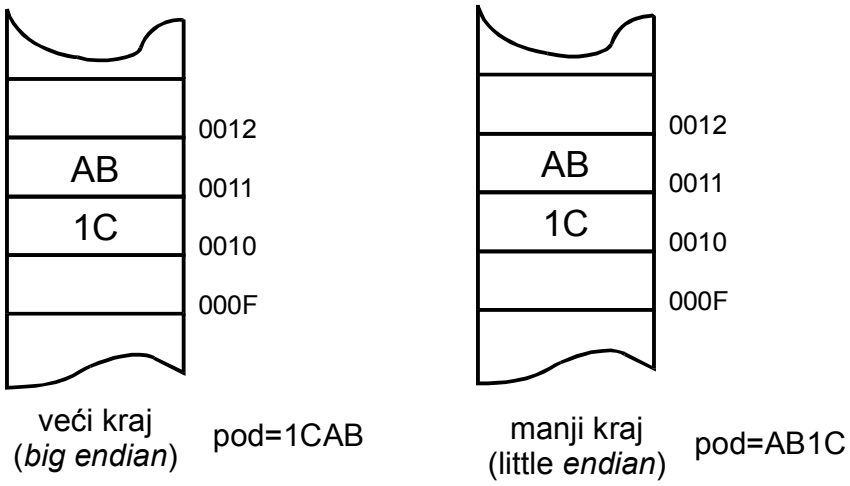
C: prijenos (*carry*)
V: pretek (*overflow*)
Z: nula (*zero*)
N: negativan (*negative*)
X: proširenje (*extend*)
Posljednjih pet bita poznati su kao uvjetni kod (*condition code* skr. CC).

Spremnici procesora MC68000 mogu se pomoću RTN opisati na sljedeći način:

D[0..7]⟨31..0⟩:	spremnici opće namjene za operande
A[0..7]⟨31..0⟩:	spremnici opće namjene za adrese
A7'⟨31..0⟩:	sistemsko kazalo stoga
PC⟨31..0⟩:	programsko brojilo
IR⟨15..0⟩:	spremnik naredbe
Status⟨15..0⟩:	statusni spremnik
SP:=A[7]:	korisničko kazalo stoga
SPP:=A7':	sistemsko kazalo stoga
C:=Status⟨0⟩:	zastavica prijenosa (*carry*)
V:=Status⟨1⟩:	zastavica pretka (*overflow*)
Z:=Status⟨2⟩:	zastavica nula (*zero*)
N:=Status⟨3⟩:	zastavica negativan (*negative*)
X:=Status⟨4⟩:	zastavica proširenja (*extend*)
INT⟨2..0⟩:=Status⟨10..8⟩:	maska prekida (*interrupt mask*)
S:=Status⟨13⟩:	nadzorni mod (*supervisor mode*)
T:=Status⟨15⟩:	mod praćenja(*trace mode*)

Glavna memorija. Glavna memorija ograničena je na 2^{24} okteta izvedbom procesora s 24 linije adresne sabirnice u cilju smanjenja cijene procesora. Procesor može pristupati oktetima, riječima (16 bita) i dugim riječima (32 bita). Okteti mogu biti smješteni proizvoljno u memoriji, riječi samo na parnim adresama, a duge riječi na adresama djeljivim sa četiri. Ukoliko ovi zahtjevi nisu ispunjeni, logika procesora signalizira grešku na adresnoj sabirnici i izvođenje programa se prekida. Ovaj način korištenja memorije poznat je pod nazivom **čvrsto svrstavanje** (*hard alignment*). Neki procesori nemaju ovakve čvrste zahtjeve kao npr. Intel 8086, tj. koriste tz. **makano svrstavanje** (*soft alignment*). Posljednje daje veću slobodu u smještanju operanada, ali npr. postavljanje riječi na neparnu adresu rezultira smanjenjem performansi jer su potrebna dva pristupa u memoriju te rekonstrukcija riječi od po dva okteta dviju pročitanih riječi.

Riječi i duge riječi pohranjuju se na način da je na nižoj odnosno najnižoj adresi pohranjen oktet najvećeg značaja, a na višoj odnosno najvišoj adresi oktet najmanjeg značaja. Ovakav način pohrane dužih podataka naziva se **veći kraj** (*big endian*), jer kraj podatka (manja adresa) sadrži oktet većeg značenja. Ima procesora koji koriste drugačiji način pohrane podataka tako da najmanja adresa sadrži oktet najmanjeg značenja, a najveća adresa oktet najvećeg značenja. Ovakav način pohrane naziva se **mali kraj** (*little endian*).



Kod RTN opisa glavne memorije fizički su prisutni samo okteti. Riječi i duge riječi tvore se od okteta kod kojih nije naglašen zahtjev da budu smješteni na parnim odnosno adresama djeljivim sa četiri.

Mb[0..2^{24}]⟨7..0⟩:	okteti u glavnoj memoriji
Mw[0..2^{23}]⟨15..0⟩:=Mb[ad]# Mb[ad+1]	riječi u glavnoj memoriji
Ml[0..2^{22}]⟨31..0⟩:= Mw[ad]# Mw[ad+2]	duga riječ u glavnoj memoriji

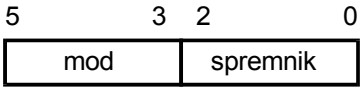
RTN može ograničiti čvrsti poredak riječi, primijenjen kod ovog procesora, na način da adresi ad definira bitove pa bi parnoj adresi posljednji bit morao biti nula, a adresi duge riječi posljednja dva bita bi morala biti nula. Ovakva dopuna izostavljena je radi jednostavnosti opisa.

3.3.2. Formati operanada i naredbi te njihova implementacija

Kako je već spomenuto procesor obavlja operacije nad oktetima, riječima i dugim riječima. Tip operanda u simboličkom jeziku određen je oznakom **.B**, **.W** i **.L** za oktet, riječ i dugu riječ. Tako npr. za prebacivanje okteta koristi se naredba **MOVE.B**, a za prebacivanje riječi **MOVE.W**. Ukoliko se ekstenzija ne specificira uzima se da je operand riječ. Većina operacija izvodi se nad svim tipovima operanada. Aritmetičke operacije interpretiraju operande kao oktete, riječi ili dvostruke riječi bez i s predznakom (aritmetika dvostrukog komplementa).

Naredbe specificiraju uz operacijski kod i spremnike ili memorijske lokacije na kojima su pohranjeni operandi, odnosno gdje je potrebno spremiti rezultat ili gdje se nalazi adresa grananja. Interpretacija ovih podataka je logički najsloženiji dio pri opisu rada procesora. Kao prvo to je posljedica ograničenog broja bita naredbe pa projektanti koriste različite trikove da bi njima kodirali što veći broj informacija. Sljedeći problem je što formati naredbi nisu standardizirani. Konačno kako je kod CISC procesora tendencija da procesor podržava sve moguće varijante neke naredbe sam opis naredbe se time znatno komplicira u usporedbi s opisom naredbi RISC procesora.

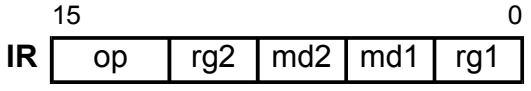
Put dohvata operanda, adresni modovi MC68000. Način pristupa memoriji i spremnicima (adresni mod) obično je određen 6 bitovnim poljem unutar naredbe.



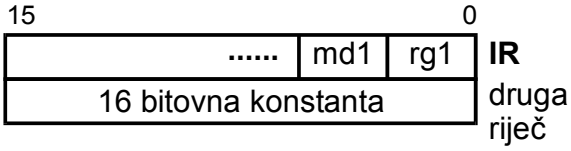
Naziv	Mod	Spr.	Simbolički	Dodatna Riječ	Kratak opis
Pod. u sprem. direktno	0	0-7	Dn	0	Dn
Adr. sprem. direktno	1	0-7	An	0	An
Adr. sprem. indirektno	2	0-7	(An)	0	M[An]
Autoinkrement	3	0-7	(An)+	0	M[An];An←An+d
Autodekrement	4	0-7	-(An)	0	An←An-d;M[An]
Baza+Pomak (Bazno)	5	0-7	disp16(An)	1	M[An+disp16]
Bazno indeksno kratko	6	0-7	disp8(An,XnLo)	1	M[An+XnLo+disp8]
Bazno indeksno dugo	6	0-7	disp8(An,Xn)	1	M[An+Xn+disp8]
Apsolutno kratko	7	0	addr16	1	M[addr16]
Apsolutno dugo	7	1	addr32	2	M[addr32]
Relativno	7	2	disp16(PC)	1	M[PC+disp16]
Rel. indeksno kratko	7	3	disp8(PC,XnLo)	1	M[PC+XnLo+disp8]
Rel. indeksno dugo	7	3	disp8(PC,Xn)	1	M[PC+Xn+disp8]
Trenutno	7	4	#data	1-2	data

Tablica prikazuje adresne modove određene šest bitovnim poljem naredbe.

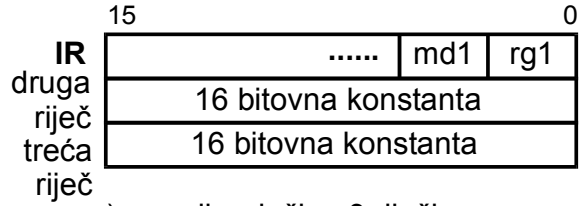
Prije nego što se pređe na detaljan opis adresnih modova opisati će se mogući formati naredbi. Procesor prvo dohvata prvu riječ naredbe i pohranjuje je u spremnik naredbe. Ova riječ sadrži specifikaciju (6 bita) jedne ili dviju adresa, kao što je prikazano na slici a). Kako je već spomenuto svaka specifikacija adrese sastoji se od moda i spremnika. Neke naredbe zahtijevaju dodatnu riječ ili riječi. Ove riječi obično su ili 8, 16 ili 32 bitovna konstanta, ali u slučaju indeksnog adresiranja druga riječ sadrži dodatne informacije. Kako se D i A spremnici mogu koristiti prilikom indeksnog adresiranja polje za specificiranje spremnika je dužine 4 bita. Na slici su prikazani neki formati naredbi.



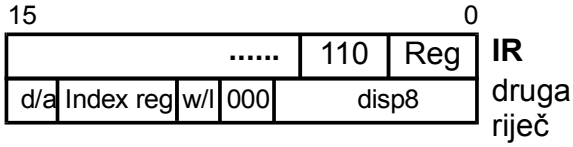
a) naredba za pomak dužine 1 riječi



b) naredba dužine 2 riječi



c) naredba dužine 3 riječi



d) naredba s indeksnim adresiranjem

op⟨3..0⟩:=IR⟨15..12⟩: operacijski kod
rg2⟨2..0⟩:=IR⟨11..9⟩: drugo polje spremnika prilikom prebacivanja
md2⟨2..0⟩:=IR⟨8..6⟩: drugo polje moda prilikom prebacivanja

```
rg1⟨2..0⟩:=IR⟨5..3⟩:                prvo polje spremnika prilikom prebacivanja
md1⟨2..0⟩:=IR⟨2..0⟩:                prvo polje moda prilikom prebacivanja
XR[0..15]⟨31..0⟩:=
      D[0..7]⟨31..0⟩#A[0..7]⟨31..0⟩:  Indeksni spremnik može biti D ili A
xr⟨3..0⟩:=Mw[PC]⟨15..12⟩:           polje specifikacije indeksa
wl:=Mw[PC]⟨11⟩:                     oznaka kratkog ili dugog indeksa
disp8⟨7..0⟩:=Mw[PC]⟨7..0⟩:          pomak pri indeksnom modu
index :=((wl=0)→XR[PC]⟨15..0⟩:
        (wl=1)→XR[PC]⟨31..0⟩:       kratka ili duga specifikacija indeksa
```

Adresni modovi grupirani su na one koji proračunavaju memorijsku adresu pomoću spremnika opće namjene, one koje ne koriste spremnike opće namjene ali sve jedno proračunavaju memorijsku adresu i one koji ne pristupaju glavnoj memoriji.

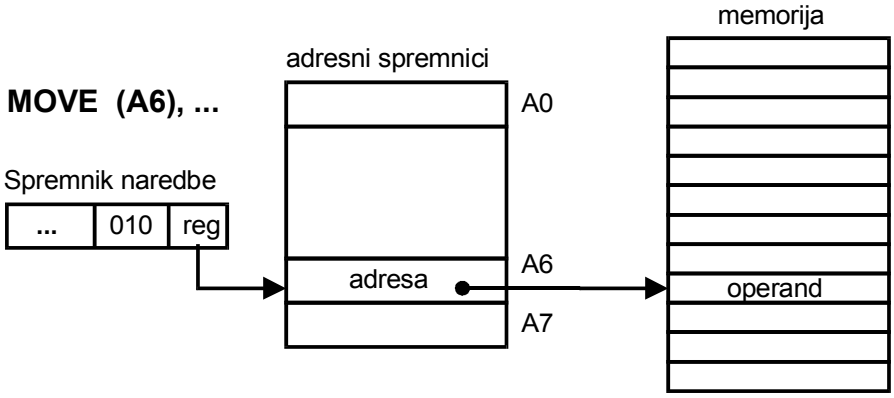
Adresni modovi koji koriste spremnike za proračun memorijske adrese. Adresni modovi koji koriste spremnike procesora za proračun efektivne adrese opisani su na sljedeći način:

```
ea(md, rg):=(
   (md=2)→A[rg⟨2..0⟩]:                  mod 2 je indirektno pomoću spremnika
   (md=3)→A[rg⟨2..0⟩]←A[rg⟨2..0⟩]+d;    mod 3 je autoinkrement
   (md=4)→A[rg⟨2..0⟩]←A[rg⟨2..0⟩]-d;    mod 4 je autodekrement
   (md=5)→A[rg⟨2..0⟩]+Mw[PC];           mod 5 je bazno adresiranje s pomakom
                PC←PC+2):
   (md=6)→A[rg⟨2..0⟩]+index+disp8;      mod 6 je bazno, indeksno adresiranje
                PC←PC+2):
```

Dužina operanada B, W ili L enkodira se različito u ovisnosti o naredbi ali to ovdje neće biti opisano. Kako dva adresna moda (mod 3 i 4) zahtijevaju informaciju o dužini podatka, uvesti će se funkcija dataln(IR) koja vraća vrijednost 1, 2 ili4 u ovisnosti da li se radi o operaciji nad oktetom, riječi ili dvostrukoj riječi.

```
d:=dataln(IR)                        broj okteta koji se dohvaćaju naredbom
```

U modu 2, indirektno adresiranje pomoću spremnika, efektivna adresa je sadržaj A spremnika, a operand ili rezultat su pohranjeni u memoriji kako je prikazano slikom:



Modovi 3 i 4, autoinkrement i aoutodekrement, identični su modu 2 uz razliku da se sadržaj adresnog spremnika poveća, odnosno umanji za dužinu podatka kojem se pristupa. I u ovom slučaju mogu biti specificirani samo adresni spremnici. Primjeri naredbi za prebacivanje podataka koje koriste modove 3 i 4 su:

```
MOVE.L    (A5)+, ...        ;A5 se poveća za 4 nakon prebacivanja podatka
MOVE.W    -(A4)+, ...       ;A4 se umanji za 2 nakon prebacivanja podatka
```

Kombinacija prebacivanja podataka s inkrementiranjem i dekrementiranjem korisna je prilikom pristupa stogu.

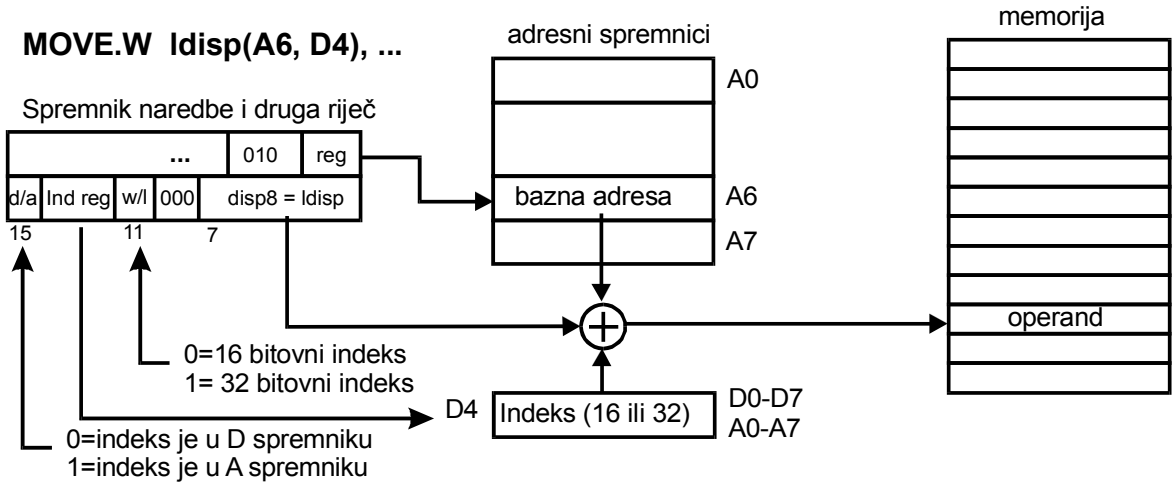
U modu 5, baznom adresiranju (*based addressing*), adresa u adresnom spremniku A koristi se kao baza kojoj se dodaje 16 bitovna konstanta upisana u riječi koja slijedi naredbu. Kod ovog adresnog moda naredbe su dužine dvije riječi. U tom slučaju programsko brojilo mora se dodatno inkrementirati što je opisano RTNom kao PC←PC+2.

Kako je već spomenuto manje su razlike između baznog i indeksnog adresiranja. Kod baznog adresiranja pomak je ograničen na 16 bita. Kod MC68000 ne postoji jednostavno indeksno adresiranje, ali kako je pomak 16 bitovna riječ, bazno adresiranje predstavlja adekvatnu zamjenu. Bazno adresiranje obično se koristi prilikom pristupa članu zapisa ili strukture. Sadržaj adresnog spremnika pokazuje na početak strukture dok pomak određuje njenog člana. Bazno adresiranje koristi se i kod prijenosa parametara čija bazna adresa je poznata.

Mod 6, bazno indeksno adresiranje (*based indexed addressing*) koristi tri elementa za proračun efektivne adrese:

adresa := sadržaj baznog spremnika + sadržaj indeksnog spremnika + 8 bitovna konstanta.

Opis disp8 := Mw[PC]⟨7..0⟩ označava da je 8 bitovna konstanta sastavni dio druge riječi naredbe (bitovi 0 do 7) upisane na memorijskoj lokaciji na koju pokazuje sadržaj programskog brojila. Adresa baze upisana je u adresnom spremniku A, ali indeks može biti upisan ili u adresni A ili podatkovni D spremnik što određuje zastavica a/d (15 bit druge riječi naredbe). Proračun adrese može obuhvaćati ili 32 ili 16 bita indeksnog spremnika što određuje zastavica w/l (11 bit druge riječi naredbe).



Adresni modovi koji ne koriste spremnike. Adresni mod 7 ne koristi za spremnike D ili A za adresiranje. Ovaj mod uključuje relativno adresiranje gdje programsko brojilo preuzima ulogu baznog spremnika. Polje spremnika unutar naredbe određuje vrstu relativnog adresiranja.

(md = 7 $\wedge$ rg = 0)$\rightarrow$ mod 7, spremnik 0: kratko apsolutno
(Mw[PC]{proširenje predznaka na 32 bita}; PC$\leftarrow$PC+2):

(md = 7 $\wedge$ rg = 1)$\rightarrow$ mod 7, spremnik 1: dugo apsolutno
(M[PC]; PC$\leftarrow$PC+4):

(md = 7 $\wedge$ rg = 2)$\rightarrow$ mod 7, spremnik 2: relativno prema PCu
(PC+Mw[PC]{proširenje predznaka na 32 bita}; PC$\leftarrow$PC+2):

(md = 7 $\wedge$ rg = 3)$\rightarrow$ mod 7, spremnik 3: relativno indeksno
(PC+indeks+disp8; PC$\leftarrow$PC+2):

Za primijetiti je da nedostaje mod md = 7, rg = 4, koji predstavlja neposredno adresiranje koje će biti objašnjeno naknadno.

Mod 7, rg = 0 ili 1, određuje kratko, odnosno dugo apsolutno adresiranje. Ovakvo adresiranje obično se naziva i direktno. Apsolutna adresa se ne mijenja ukoliko se program premjesti na drugu lokaciju u memoriji. Zato se koristi za pristup lokacijama koje nisu ovisne o položaju programa kao npr. ulazno/izlazne adrese, ili varijable operacijskog sustava. Kratko i dugo apsolutno adresiranje rezultat je nastojanja da se može ostvariti kompatibilnost s novim generacijama ovog procesora koje će koristit 32 bitovne adrese. Konstanta koja određuje apsolutnu adresu pohranjena je u drugoj odnosno i trećoj riječi naredbe. Primjeri ovih naredbi su:

```
MOVE.B   IOPORT.W, ...     ;prebaci oktet s IOPORT određen kratkom adresom
MOVE.W   INTVECT.L, ...    ;prebaci riječ s INTVECT određenog dugom adresom
```

Mod 7, rg = 2, je relativno adresiranje (*relative addressing*) kod kojeg se pomak specificira u odnosu na sadržaj programskog brojila. Kod premještanja programa na drugu memorijsku lokaciju adresa operanda se pomiče zajedno s programom. Kod ovog procesora nije dozvoljeno korištenje relativnog adresiranja za pohranu rezultata iz razloga što bi na taj način rezultat mogao prepisati kod programa, a programi koji sami sebi mijenjaju kod nisu poželjni.

Mod 7, rg = 3, relativno indeksno adresiranja (*relative indexed addressing*) je analogno baznom indeksnom adresiranju uz razliku da se kao bazni spremnik koristi programsko brojilo umjesto adresnog spremnika. Relativno indeksno adresiranje koristi se za pristup tablicama koje su sastavni dio programa. Primjer relativnog indeksnog adresiranja je:

```
MOVE.W   POMAK(PC, D4), ...
```

Adresni modovi koji ne pristupaju memoriji. Naravno nisu svi operandi ili rezultati pohranjeni u memoriji. Oni mogu biti ili u podatkovnim ili adresnim spremnicima. Njima se pristupa pomoću njihovog rednog broja. Također moguće je da je operand konstanta koja je sastavni dio naredbe. Iako za te operande nije potrebno računati efektivnu adresu ovakvo njihovo korištenje smatra se adresnim modom.

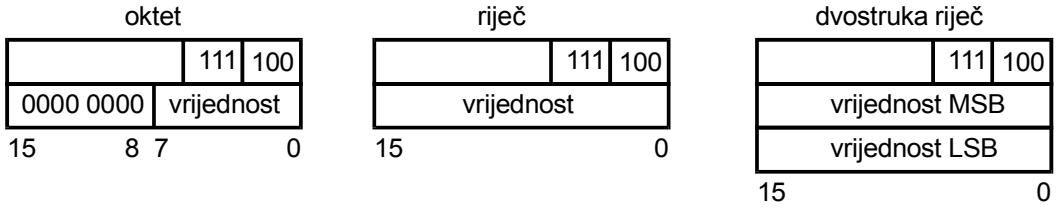
Zbog mogućnosti ovog procesora da obrađuje operande različitih dužina, različiti dijelovi spremnika koriste se za operande različitih dužina kao što je različit broj dodatnih riječi naredbe zbog konstante koja je njen sastavni dio. U preciznom RTN opisu potrebno je razlučiti ove situacije pomoću konstante d := dužinapod(IR). U nastavku slijedi opis ovog moda:

```
memval(md, rg) := (                                   memorijska adresa
    (md⟨2..1⟩ = 1) ∨ (md⟨2..1⟩ = 2) ∨ (md⟨2..0⟩ = 6) ∨  se koristi samo sa
    ((md⟨2..0⟩ = 7) ∨ (rg⟨2⟩ = 0))):                  ovim adresnim modovima

opndl(md, rg)⟨31..0⟩ := (                             dugi operand može biti:
    memval(md,rg) → Ml[ea(md,rg)]⟨31..0⟩:             u memoriji
    md = 0 → D[rg]⟨31..0⟩:                            u podatkovnom spremniku
    md = 1 → A[rg]⟨31..0⟩:                            u adresnom spremniku
    (md = 7 ∧ rg = 4) → (Ml[PC]⟨31..0⟩: PC ← PC + 4)) u okviru naredbe

opndw(md, rg)⟨15..0⟩ := (                             riječ (operand) može biti:
    memval(md,rg) → Mw[ea(md,rg)]⟨15..0⟩:             u memoriji
    md = 0 → D[rg]⟨15..0⟩:                            u podatkovnom spremniku
    md = 1 → A[rg]⟨15..0⟩:                            u adresnom spremniku
    (md = 7 ∧ rg = 4) → (Mw[PC]⟨15..0⟩: PC ← PC + 2)) u okviru naredbe

opndb(md, rg)⟨7..0⟩ := (                              oktet (operand) može biti:
    memval(md,rg) → Mb[ea(md,rg)]⟨7..0⟩:              u memoriji
    md = 0 → D[rg]⟨7..0⟩:                             u podatkovnom spremniku
    md = 1 → A[rg]⟨7..0⟩:                             u adresnom spremniku
    (md = 7 ∧ rg = 4) → (Mb[PC]⟨7..0⟩: PC ← PC + 2)) u okviru naredbe

opnd(md, rg) := (                                     Dužina operanda u
    (d = 1) → opndb(md, rg):                          naredbi ukazuje koji
    (d = 2) → opndw(md, rg):                          tip naredbe je potrebno
    (d = 4) → opndl(md, rg)):                         koristiti
```

Modovi 0 i 1, direktno podatkovni odnosno adresni spremnik (*data register direct, address register direct*) pristupaju operandima i pohranjuju rezultat direktno u spremnike opće namjene. Primjeri ovih naredbi su:

```
MOVE    D6, ...
MOVE    A6, ...
```

Mod 7, rg = 4, neposredno adresiranje (*immediate addressing*) koristi se za pristup konstanti koja je sastavni dio programa. Konstanta može biti oktet, riječ ili dvostruka riječ. Ovakva konstanta pohranjena je neposredno nakon prve riječi naredbe. Kod pristupa konstanti dužine okteta zanemaruje se osma bita većeg značenja druge riječi naredbe. Naredbe koje koriste neposredno adresiranje prikazane su na sljedećoj slici:



MOVE.B #12, ... MOVE.W #1200, ... MOVE.L #120000, ...

Sažetak adresnih modova MC68000

- Postoji veći broj adresnih modova kao i mogućnost njihovog korištenja kod mnogih različitih naredbi što je jedna od obilježja CISC procesora.
- Osnovni elementi koje koriste adresni modovi su spremnici procesora, programsko brojilo te konstanta koja je sastavni dio naredbe.
- Kako je u okviru naredbe može postojati promjenjivi broj konstanti tako su i naredbe različitih dužina. Praktički ukoliko naredba za pomak koristi direktnu adresu za operand i rezultat ta naredba je dužine pet riječi.
- Za primijetiti je da su A spremnici privilegirani prilikom proračuna efektivne adrese u usporedbi s D spremnicima. Samo A spremnici mogu se koristiti kod indirektnog adresiranja pomoću spremnika te kod auotinkrementa i autodekrementa. Iako A i D spremnici se koriste kod indeksnog adresiranja, samo A spremnik može biti baza.
- Pojedini adresni modovi imaju i dodatnog utjecaja na neke spremnike. Tako autoinkrement i autodekrement mijenjaju sadržaj adresnog spremnika, a modovi koji zahtijevaju dodatne riječ naredbe povećavaju za odgovarajuću vrijednost sadržaj programskog brojila.

3.3.3. Skup naredbi MC68000

Slijedi opis skupa naredbi procesora MC68000. Razmatranje će započeti opisom aktivnosti koja je zajednička svim naredbama, a to je interpretacija naredbe. Pojam interpretacija naredbe je skraćenica za ciklus dohvata i izvođenja naredbe. Posebna stanja (*exceptions*) također se mogu promatrati kao interpretacija naredbe.

Skup naredbi dijeli se u podskupove, klase naredbi sličnih aktivnosti. Zbog velikog broja naredbi opisati će se samo najčešće korištene naredbe za prebacivanje podataka, aritmetičke i logičke naredbe te naredbe za upravljanje programskim tokom. Naredbe koje se relativno rijetko koriste kao i neke posebne naredbe biti će preskočene. Njihov opis dostupan je u literaturi koju daje proizvođač ovog procesora.

Ciklus dohvati i izvedi (*fetch-execute*). Neke aktivnosti zajedničke su svim naredbama procesora. Temelj procesora je ciklus dohvati i izvedi. Za svaku naredbu potrebno je dohvatiti riječ koja predstavlja naredbu, pohraniti je u spremnik naredbe, povećati sadržaj programskog brojila da pokazuje na sljedeću riječ te poduzeti aktivnosti u ovisnosti o sadržaju spremnika naredbe, odnosno izvesti naredbu. Za vrijeme izvođenja naredbe programsko brojilo može biti dodatno povećano ukoliko je potrebno dohvatiti konstantu

koja je sastavni dio naredbe. Kod RTN opisa ovog ciklusa ponovo se koristi Run zastavica koja je praktički vanjski signal koji se koristi da zaustavi izvođenje programa.

interpretacija_naredbe := (
Run → ((IR⟨15..0⟩ ← Mw[PC] ⟨15..0⟩: dok je izvođenje dohvati naredbu
PC ← PC + 2); izvođenje_naredbe); inkrementiraj PC i izvedi naredbu

Naknadno kada se budu opisivala posebna stanja i prekidi ustanoviti će se da se interpretacija naredbe ne sastoji samo od ciklusa dohvati i izvedi. Dodatno uz obradu posebnih stanja postoje i određene informacije o stanju procesora koje utječu na interpretaciju naredbe.

Upravljanje procesorom i razine privilegija. Oktet većeg značenja statusnog spremnika posvećen je definiranju statusa procesora kao i njegovim upravljanjem. Procesor MC68000 ima dvije razine privilegija, nadzorni mod rada i korisnički. U nadzornom modu, kada je S bit statusnog spremnika postavljen mogu se izvoditi sve naredbe kojima raspolaže procesor. Tu spadaju i privilegirane naredbe koje mogu mijenjati sadržaj statusnog spremnika i sistemskog kazala stoga A7'. U korisničkom modu rada privilegirane naredbe se ne mogu izvesti. Mod u kojem se procesor nalazi signaliziran je logičkom razinom na jednom izlazu sklopa. Ovaj signal može koristiti npr. sklop za upravljanje memorijom kako bi dodatno ograničio pristup određenim memorijskim lokacijama.

Naredbe za prebacivanje podataka. Sljedeća tablica prikazuje neke naredbe za pomak podataka:

Naredba	Operandi →	Prva riječ	X N Z V C	Operacija	Veličina
MOVE.B	EAs, EAd	0001ddddddssssss	- x x 0 0	dst ← src	oktet
MOVE.W	EAs, EAd	0011ddddddssssss	- x x 0 0	dst ← src	riječ
MOVE.L	EAs, EAd	0010ddddddssssss	- x x 0 0	dst ← src	dvostruka riječ
MOVEA.W	EAs, An	0011rrr001ssssss	- - - - -	An ← src	riječ
MOVEA.L	EAs, An	0010rrr001ssssss	- - - - -	An ← src	dvostruka riječ
LEA.L	EAc, An	0100aaa111ssssss	- - - - -	An ← EA	adresa
EXG	Dx, Dy	1100xxx1mmmmmyyy	- - - - -	Dx ↔ Dy	dvostruka riječ

EAs: Adresa izvora (*source*), svi adresni modovi osim prebacivanje okteta u adresni spremnik
EAd: Adresa odredišta (*destination*), svi adresni modovi osim neposrednog ili relativnog
EAc: Upravljačka adresa (*control*), svi adresni modovi osim pomoću spremnika, autoinkrement i autodekrement i neposredno
ssssss, dddddd: specifikacija adresnog moda izvorišne i odredišne adrese
rrr, yyy: specifikacija jednog od osam spremnika
aaa: jedan od osam adresnih spremnika
An, Dn: jedan od osam adresnih ili podatkovnih spremnika
mmmmmm: polje moda. 01000 - izmjeni sadržaj podatkovnihspremnika, 01001 – izmjeni sadržaj adresnih spremnika, 10001 izmjeni sadržaj adresnih i podatkovnih spremnika, xxx određuje podatkovni a yyy adresni spremnik
Uvjetni kod: - nepromijenjena predhodna vrijednost, x –operacija mijenja zastavicu, 0 ili 1 - vrijednost

Prve tri naredbe, MOVE, podrazumijevaju prijenos podataka između dviju memorijskih lokacija. Druge dvije naredbe, MOVEA, podrazumijevaju da je jedan ili oba operanda pohranjeni u spremnicima. Značenje pojedinih polja prve riječi naredbe detaljno je opisano legendom iza tablice. Tablica pokazuje da prve tri naredbe mijenjaju stanje zastavica u spremniku stanja. Ako se prebacuje negativna vrijednost ili nula tada se postavljaju

zastavice N i Z. Kako se ne radi o aritmetičkim ili logičkim operacijama nije moguće da dođe do preteka ili prijenosa pa se V i C bitovi postavljaju u nulu. Sljedeći RTN opis MOVE naredbi koristi dodatni privremeni spremnik tmp⟨31..0⟩ kako bi opisao utjecaj naredbe na zastavice spremnika stanja. Ovaj spremnik ne postoji u praktičnoj implementaciji. Sljedeći je RTN opis naredbe MOVE:

op⟨3..0⟩ := IR⟨15..12⟩: rg1⟨2..0⟩ := IR⟨2..0⟩: md1⟨2..0⟩ := IR⟨5..3⟩:
rg2⟨2..0⟩ := IR⟨11..9⟩: md2⟨2..0⟩ := IR⟨8..6⟩:

tmp⟨31..0⟩:
move (:= op⟨3..2⟩ := 0) → (
    tmp ← opnd(md1, rg1);
        (Z ← (tmp=0): N ← (tmp<0): V ← 0: C ← 0):
        rslt(md2, rg2) ← tmp):

Programeri nisu bili jedinstveni u ocjeni da li je potrebno da naredbe za prebacivanje podataka mijenjaju sadržaj spremnika stanja, odnosno da li to dozvoliti samo aritmetičkim i logičkim naredbama.

Naredba MOVEA omogućava prebacivanje vrijednosti, obično adrese, u jedan od adresnih spremnika bez utjecaja na spremnik stanja. Ovo je razumno rješenje s obzirom na činjenicu da pri prebacivanju adrese programer nije zainteresiran za njenu aritmetičku vrijednost. Pažljivo razmatranje operacijskog koda ukazuje da MOVEA je praktički MOVE naredba s adresnim modom 1. Ovo rješenje istog operacijskog koda dviju naredbi uz dodatnu provjeru polja specifikacije adresnog moda koji određuje da li naredba utjeće ili ne na spremnik stanja, usložnjava RTN opis naredbi kao i postavlja dodatne zahtjeve na sklopovsku realizaciju procesora

LEA naredba (*load effective address*) omogućava programeru da provodi složeniji proračun adrese operanda, pohrani rezultat proračuna u adresni spremnik, a sve bez pristupa toj adresi. RTN opis ove naredbe je:

lea (:=op⟨3..0⟩ = 0100) ∧ md2 = 7)) → A[rg2] ←ea(md1,rg1):

Opis pokazuje da se adresa, a ne sadržaj adrese upisuje u odredišni spremnik. Nisu dozvoljeni svi adresni modovi jer npr. adresiranje pomoću spremnika ili neposredno adresiranje ne izvodi proračun adrese.

Naredba EXG (*exchange registers*) zamjenjuje sadržaj dvaju spremnika. Pored oblika naredbe prikazanog u tablici kada se zamjenjuje sadržaj dvaju podatkovnih spremnika, moguće je i zamijeniti sadržaj dvaju adresnih spremnika te jednog podatkovnog i jednog adresnog spremnika. Ova naredba ne utječe na sadržaj spremnika stanja.

Osim navedenih postoje i druge naredbe za prebacivanje podataka kao npr. naredba kojom se sadržaj kratke konstante koja je sastavni dio naredbe upisuje u spremnik MOVEQ, odnosno naredba koja zamjenjuje oktete jedne riječi SWAP, i druge.

I bez detaljnije analize prikazanih naredbi uočava se nedostatak pravilnosti u obliku naredbi. To je posljedica namjere projektanta da sa što manje bita kodira što veći broj naredbi i adresnih modova.

Aritmetičke i logičke naredbe. U sljedećoj tablici prikazan je dio aritmetičkih i logičkih naredbi procesora MC68000:

Naredba	Operandi	Naredbena riječ	X N Z V C	Operacija	Veličina
ADD	EA,Dn	1101rrrmmmaaaaaa	x x x x x	dst ← dst + src	b, w, l
SUB	EA,Dn	1001rrrmmmaaaaaa	x x x x x	dst ← dst - src	b, w, l
CMP	EA,Dn	1011rrrmmmaaaaaa	- x x x x	dst-src	b, w, l
CMPI	#dat,EA	00001100wwaaaaaa	- x x x x	dst-immed.data	b, w, l
MULS	EA, Dn	1100rrr111aaaaaa	- x x 0 0	Dn←Dn*src	l←w*w
DIVS	EA,Dn	1000rrr111aaaaaa	- x x x 0	Dn←Dn/src	l←l/w
AND	EA,Dn	1100rrrmmmaaaaaa	- x x 0 0	dst←dst∧src	b, w, l
OR	EA,Dn	1000rrrmmmaaaaaa	- x x 0 0	dst←dst∨src	b, w, l
EOR	EA,Dn	1011rrrmmmaaaaaa	- x x 0 0	dst←dst⊕src	b, w, l
CLR	EAs	01000010wwaaaaaa	- 0 1 0 0	dst∧dst	b, w, l
NEG	EAs	01000100wwaaaaaa	- x x x x	dst←0 - dst	b, w, l
TST	EAs	01001010wwaaaaaa	- x x 0 0	dst–0	b, w, l
NOT	EAs	01000110wwaaaaaa	- x x x x	dst← ¬dst	b, w, l

rrr je D spremnik

mmm 3 bitovno polje moda koje specificira dst, EA ili Dn i operande kao b, w, ili l

oktet	riječ	dvostruka riječ	odredište
000	001	010	Dn
100	101	110	EA

EA efektivna adresa

aaaaaa 6 bitovno polje koje specificira adresiranje. Nisu svi modovi dozvoljeni kod ovih naredbi

ww polje koje specificira veličinu riječi: 00 – oktet, 01 – riječ, 10 – dvostruka riječ

CMPI naredbu slijedi jedna ili dvije riječi koje sadrže neposredni operand za usporedbu

Naredbe za zbrajanje ADD, oduzimanje SUB i usporedbu CMP imaju isti oblik, a razlikuju se samo u četiri bita najvećeg značenja naredbene riječi. Ova četiri bita određuju o kojoj se operaciji radi. Ove operacije provode se nad operandima dužine okteta, riječi ili dvostruke riječi u ovisnosti o vrijednosti polja op-moda mmm. Jedan od operanada mora biti u podatkovnom spremniku Dn, dok je drugi određen šest bitovnim EA poljem. mmm polje također određuje da li se rezultat operacije (odredište) pohranjuje u podatkovni spremnik Dn ili u memoriju EA. Naredbe CMPI uspoređuje neposrednu vrijednost koja je sastavni dio naredbe s podatkom pohranjenim u memoriji.

Naredbe za zbrajanje i oduzimanje su tipične aritmetičke naredbe. Kod MC68000 rezultat uvijek prepisuje vrijednost drugog operanda. Ukoliko se zanemari postavljenje zastavica spremnika stanja, RTN opis naredbe za oduzimanje SUB je:

```
sub(:= op = 9) →
      (md2⟨2⟩ = 0) → D[rg2] ← D[rg2] – opnd(md1,rg1):
      (md2⟨2⟩ = 1) → (memval(md1,rg1) → (tmp ← ea(md1,rg1);
                        M[tmp] ← M[tmp] – D[rg2]):
                        ¬memval(md1,rg1) → rslt(md1,rg1) ← rslt(md1.rg1) – D[rg2])):
```

Naredba SUB definirana je operacijskim kodom 9. U ovisnosti o polju md2 može se od spremnika odbiti vrijednost iz memorije (2 bit md2 je 0) ili od vrijednosti memorije odbiti sadržaj spremnika. U drugom slučaju polja md1 i rg1 mogu specificirati gdje se nalazi adresa operanda, kada je potrebno efektivnu adresu dohvatiti iz memorije i pohraniti je u privremeni spremnik tmp, a tek onda dohvatiti operand i pohraniti rezultat. Ukoliko ova

polja direktno određuju adresu operanda bez pristupa memoriji moguće je preskočiti prvi korak.

Naredbe CMP i CMPI razlikuju se od ostalih aritmetičkih i logičkih naredbi budući da one nigdje ne upisuju rezultat obrade. Ove naredbe koriste se kako bi se postavile zastavice u spremniku stanja bez ikakve preinake operanada.

Naredba MULS množi dvije riječi s predznakom. Rezultat se pohranjuje kao dvostruka riječ u jedan od podatkovnih spremnika. Slično naredba DIVS dijeli jednu dvostruku riječ s predznakom iz podatkovnog spremnika Dn s jednom riječi s predznakom iz memorije pohranjene na adresi EA. Rezultat dijeljenja, odnosno kvocijent upisuje se u riječ manjeg značaja spremnika Dn, dok se ostatak dijeljenja upisuje u riječ većeg značaja spremnika Dn. Također postoje naredbe za množenje i dijeljenje bez predznaka, MUL i DIV.

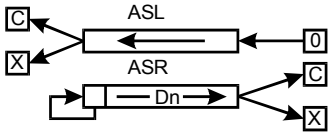
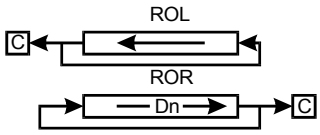
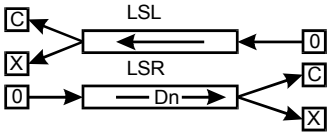
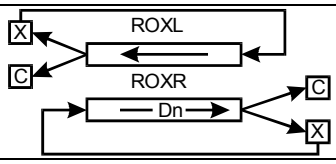
Logičke operacije AND (I), OR (ILI) i EOR (isključivo ILI) imaju sličan oblik naredbi ADD. Razlika je samo u četiri bita većeg značenja koji specificiraju operaciju.

Četiri unarne operacije CLR, NEG, TST i NOT također su međusobno sličnog oblika. Osam bita većeg značenja specificira o kojoj se operaciji radi, sljedeća dva bita, ww, specificiraju tip operanda (oktet, riječ, dvostruka riječ), a posljednjih šest bita specificira efektivnu adresu, EA. Naredba CLR (*clear*), očisti, postavlja sve bitove operanda u nulu. Naredba NEG (*negate*), negativno, određuje negativnu vrijednost, odnosno 2'ki komplement operanda. Naredba TST (*test*), ispitaj, ispituje vrijednost operanda i postavlja zastavice spremnika stanja ukoliko je operand negativan N zastavicu, odnosno ako je nula Z zastavicu. Naredba NOT, ne, izvodi logičku NE operaciju nad operandom mijenjajući nule jedinicama te jedinice nulama, odnosno određuje 1'vi komplement.

Postoje još i naredbe koje izvode aritmetičke i logičke operacije nad neposrednim vrijednostima, ADD, SUB, CMP, AND, OR i EOR.

Logičke naredbe mogu se koristiti za "čišćenje" (AND) (postavljanje u nulu), postavljanje (OR) ili invertiranje (EOR) bita operanada. Dodatno MC68000 može obavljati operacije nad pojedinačnim bitovima operanada, postavi u nulu (BCLR) ili jedinicu (BSET), invertiraj (BCHG), ispitaj (BTST). Ove naredbe rade s oktetima ili dvostrukim riječima. Redni broj bita koji se obrađuje određen je direktnim poljem naredbe ili može biti sadržaj podatkovnog spremnika. Redni broj bita za operand dužine okteta specificiran je s tri bita manjeg značenja, a za operand dužine dvostruke riječi s pet bita manjeg značenja.

Naredbe za posmak i rotaciju koriste se za pomicanje ili rotiranje sadržaja spremnika u lijevo ili desno za određeni broj mijesta. Bit najvećeg, odnosno najmanjeg značenja pohranjuje se pri svakom posmaku u spremnik statusa na mjesto C ili X bita. Rotacija se izvodi preko X bita. Naredbe za posmak i rotaciju prikazane su u tablici. Pri tome d bit je oznaka smjera posmaka, odnosno rotacije.

Naredba	Operandi	Naredbena riječ	X V	Operacija
ASd	EA	1110000d11aaaaaa	x x	
ASd	#cnt,Dn	1110cccdww000rrr	x x	
ASd	Dm,Dn	1110RRRdww100rrr	x x	
ROd	EA	1110011d11aaaaaa	- 0	
ROd	#cnt,Dn	1110cccdww011rrr	- 0	
ROd	Dm,Dn	1110RRRdww111rrr	- 0	
LSd	EA	1110001d11aaaaaa	x 0	
LSd	#cnt,Dn	1110cccdww001rrr	x 0	
LSd	Dm,Dn	1110RRRdww101rrr	x 0	
ROXd	EA	1110010d11aaaaaa	x 0	
ROXd	#cnt,Dn	1110cccdww010rrr	x 0	
ROXd	Dm,Dn	1110RRRdww110rrr	x 0	

d bit smjera: 0 ili D – posmak/rotacija u desno, 1 ili L - posmak/rotacija u lijevo
rrr jedan od podatkovnih spremnika Dn
ccc broj pomaka: 0 = 8, 1 – 7 = broj
RRR spremnik gdje je broj pomaka pohranjen
EA efektivna adresa određena s aaaaaa
aaaaaa šest bitovno polje koje specificira efektivnu adresu. Svi su modovi dozvoljeni osim neposrednog, direktno adresni spremnik, relativno i indeksno relativno
ww veličina operanda: 00 – oktet, 01 – riječ, 10 – dvostruka riječ
N, Z i C bitovi spremnika statusa mijenjaju svim naredbama

Naredbe za upravljanje programskim tokom. MC68000 koristi uvjetna grananja temeljen stanja jednog ili više zastavica, bitova, spremnika statusa, C, N , V i Z. Sljedeća tablica prikazuje naredbe za upravljanje programskim tokom.

Naredba	Operandi	Naredbena riječ	Operacija
Uvjetna grananja			
Bcc	disp	0110ccccdddddddd DDDDDDDDDDDDDDDD	**if** (cond) **then** PC ← PC + disp
DBcc	Dn,disp	0101cccc11001rrr DDDDDDDDDDDDDDDD	**if** ¬(cond) **then** (Dn ← Dn – 1 **if** Dn ≠ -1 **then** PC ← PC + disp) **else** PC ← PC + 2
Scc	EA	0101cccc11aaaaaa	**if**(cond) **then** (EA) ← FFH **else** (EA) ← 00H
Bezuvjetna grananja			
BRA	disp	01100000dddddddd DDDDDDDDDDDDDDDD	PC ← PC + disp
BSR	disp	01100001dddddddd DDDDDDDDDDDDDDDD	-(SP) ← PC; PC ← PC + disp
JMP	EA	0100111011aaaaaa	PC ← EA
JSR	EA	0100111010aaaaaa	-(SP) ← PC; PC ← EA
Naredbe za povrat iz potprograma			

RTR		0100111001110111	CC ← (SP)+; PC ← (SP)+
RTS		0100111001110101	PC ← (SP)+

rrr adresni spremnik An

Ako je 8 bitovni pomak ddddddddd jednak nuli, tada je pomak DDDDDDDDDDDDDDDD

EA efektivna adresa

aaaaaa 6 bitovno polje za specifikaciju adrese. Nisu dozvoljeni svi adresni modovi

ww dužina operanda, 00 – oktet, 01 – riječ, 10 – dvostruka riječ

cccc kod uvjeta definiran sljedećom tablicom

Kod uvjeta cccc je četiri bitovno polje koje je dio naredbe, čije značenje je definirano sljedećom tablicom:

Naziv	Značenje	Kod	Zastavice
T	istina (true)	0000	1
F	nije istina (false)	0001	0
CC	C nije postavljen (carry clear)	0100	C
CS	C postavljen (carry set)	0101	C
EQ	jednako (equal)	0111	Z
NE	nije jednako (not equal)	0110	Z
MI	minus	1011	N
PL	plus	1010	N
LS	manje ili jednako (low or same)	0011	C+Z
LT	manje od (less than)	1101	$(N\wedge\neg V)\vee(\neg N\wedge V)$
GE	veće ili jednako (greater or equal)	1100	$(\neg N\wedge\neg V)\vee(N\wedge V)$
GT	veće od (greater than)	1110	$(\neg N\wedge\neg V\wedge\neg Z)\vee(N\wedge V\wedge Z)$
LE	manje od (less or equal)	1111	$(N\wedge\neg V)\vee(N\wedge\neg V)\vee Z$
HI	visok (high)	0010	$C\cdot Z$
VC	nema preteka (overflow clear)	1000	V
VS	pretek postavljen (overflow set)	1001	V

U tablici koja prikazuje naredbe za upravljanje programskim tokom koristi se opis kao u višim jezicima. Razlog je složen i dug opis pomoću RTNa. Tako npr. –(SP) ← označava da se kazalo stoga, SP, dekrementira a zatim koristi kao adresa za upis vrijednosti. Praktički ovo označava operaciju stavljanje varijable (podatka ili adrese) na stog. Slično ← (SP)+ označava da se očitava vrijednost s adrese na koju pokazuje kazalo stoga, a zatim se kazalo stoga inkrementira. Ovo je operacija skidanja varijable sa stoga.

Naredba za uvjetno grananje, Bcc, koristi se za upravljanje programskim tokom temeljen stanja jedne ili više zastavica uvjeta (bitova spremnika stanja). Na ovaj način moguće je realizirati različita uspoređivanja, veće, veće ili jednako, jednako, manje ili jednako, manje. Primjer ovih naredbi je ispitivanje da li je varijabla X jednaka nuli, te ako je, izvođenje programa se nastavlja na drugoj lokaciji (simbolički označena s LOC). Primjena ove naredbe prikazana je primjerom:

if (X = 0) **goto** LOC

```
TST   X          ;postavlja Z ili N zastavice
BEQ   LOC        ;grananje na LOC ako je Z postavljen (Z = 1)
```

Naredba za dekrementiranje i grananje ukoliko je uvjet ispunjen, DBcc (*decrement and branch if condition*), koristi se kod petlji. Ovu naredbu određuju tri parametra, spremnik koji

služi kao brojilo, Dn, adresa grananja, disp, i uvjet grananja, cc. Ova naredba djeluje na sljedeći način:

1. Ispituje se uvjet grananja cc. Ako je zadovoljen, naredba se prekida nakon što se uveća programsko brojilo koje sada pokazuje na sljedeću naredbu.
2. Ako uvjet nije zadovoljen, sadržaj spremnika Dn se dekrementira. Ako je rezultat –1 (što pokazuje da je odbrojen broj prolaza kroz petlju) naredba se završava i prelazi se na izvođenje sljedeće naredbe, odnosno izlazi iz petlje.
3. Ako rezultat nakon dekrementiranja nije –1, tada se izvodi relativno grananje na adresu početka petlje.

Opis naredbe ukazuje da se ona može koristiti za upravljanje petljom na način ili da petlja završava kada je sadržaj spremnika jednak 0 ili kada je zadovoljen uvjet određen naredbom. Ako uvjet nije specificiran tada se petlja izvodi sve dok sadržaj spremnika nije jednak nuli.

Naredba Scc ispituje uvjet (zastavice stanja) te ako je uvjet zadovoljen upisuje na adresu FFH, a ako nije upisuje 00H. FFH može se tumačiti kao logička konstanta “istina” (*True*), a 00H kao logička konstanta “neistina” (*False*). Ovo je proizvoljna interpretacija Booleove algebre od strane projektanta procesora. Mogućnost postavljanja Booleovih varijabli korisna prilikom proračuna složenijih Booleovih funkcija.

Za bezuvjetna grananja koriste se naredbe BRA (*Branch*) i JMP(*Jump*). Naredba BRA ima za posljedicu bezuvjetan nastavak izvođenja programa na adresi koja se nalazi udaljena za vrijednost konstante, koja je sastavni dio naredbe, od trenutne vrijednosti programskog brojila. Ukoliko je polje prve riječi naredbe ddddddddd $\neq$ 0 tada ono određuje 8 bitovni pomak, ukoliko je ddddddddd = 0 tada pomak određuje druga riječ naredbe. JMP naredba razlikuje se od naredbe za grananje utoliko što koristi standardnu efektivnu adresu (apsolutnu adresu) za nastavak programa.

Naredbe BSR (*Branch to Subroutine*), JSR (*Jump to Subroutine*), RTS (*Return from Subroutine*), i RTR (*Return*) koriste se za poziv i povratak iz procedura. Naredbe za poziv procedura ili potprograma, BSR i JSR, prvo stavljaju programsko brojilo na stog, a zatim izvode relativno ili apsolutno grananje. Adresa naredbe koja je sljedila poziv potprograma sada se nalazi na vrhu stoga. Po završetku potprograma izvođenjem naredbe za povratak iz potprograma, RTS, skida se sadržaja s vrha stoga i prebacuje u programsko brojilo. Nastavak programa je na naredbi koja je slijedila poziv potprograma. Naredba RTR ima slično djelovanje kao i naredba RTS s tim da prvo se sa stoga skida sadržaj koji se pohranjuje u spremnik stanja (samo korisnički oktet), a zatim sadržaj koji se pohranjuje u programsko brojilo. Ova naredba posljedica je potrebe da se ponekad nakon izvođenja potprograma obnovi ne samo stanje programskog brojila nego i stanje spremnika stanja (korisničkog okteta) iz trenutka grananja u potprogram. Da bi ova naredba ispravno djelovala potrebno je kao prvu naredbu potprograma staviti sadržaj spremnika stanja na stog.

3.3.4. Primjer programa procesora MC6000

Iako je prethodnim tekstom opisan samo dio naredbi procesora MC68000 ipak su obrađene sve značajnije naredbe koje se koriste za prebacivanje podataka, aritmetičke i logičke naredbe te za upravljanje programskim tokom. Jednostavni programski odsječak

pokazati će korištenje tih naredbi, kao i neke druge karakteristike simboličkog jezika procesora MC68000.

Uz naredbe kod simboličkog jezika postoje i direktive programuprevodiocu. One pomažu programeru da čitljivije i jednostavnije napiše program te da definira apsolutne adrese. Direktive su npr.

- **EQU** (*equate*), direktiva koja pridodaje konstante simboličko ime. Tako u programu nije više potrebno pisati vrijednost konstante nego je dovoljno se pozivati na njeno ime.
- **DS.B/W/L** (*data storage*), koristi se za rezerviranje memorije za varijable.
- **ORG** (*origin*), specificira da odsječak programa koji slijedi počinje na adresi definiranoj ovom direktivom. Kod upisa programa u memoriju odsječci programa moraju biti počinjati upravo na specificiranoj adresi. Temeljen ove direktive program prevodilac određuje apsolutne adrese.
- **#** je oznaka za vrijednost. Ako se napiše naredbe MOVE.W #10, D0 to znači da je nova vrijednost spremnika D0 = 10, za razliku MOVE.W 10,D0 kada se u D0 upisuje riječ s memorijske adrese 10.
- **$** je oznaka za heksadecimalnu vrijednost. $10 je 10H = 16.

Zadatak je napisati program koji pretražuje polje od 132 znaka koji predstavljaju liniju teksta tražeći znak za kraj linije i povratak na početak (*carriage return CR*), odnosno ASCII vrijednost 13.

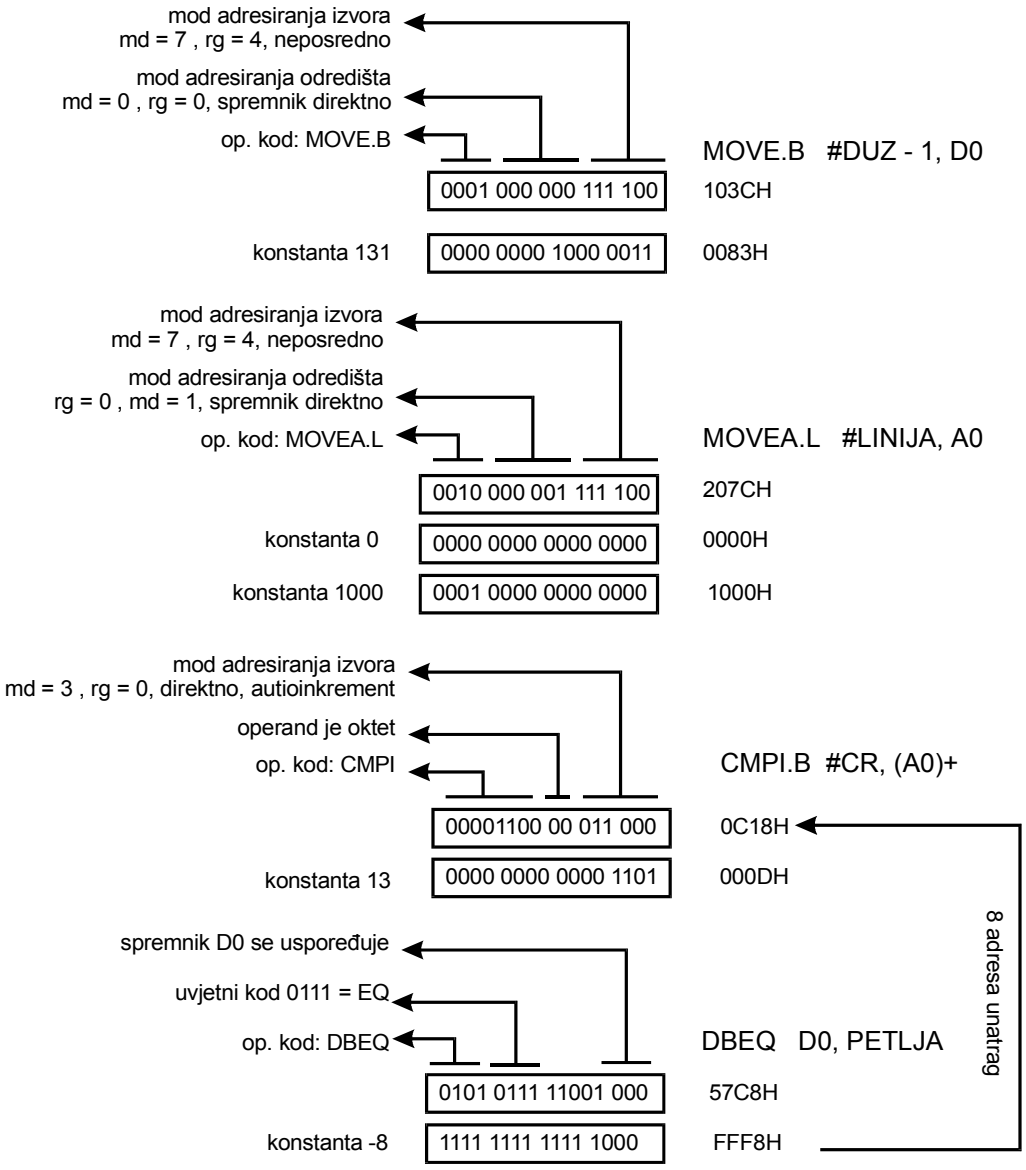
```
CR        EQU     13          ;definira 13 kao CR
DUZ       EQU     132         ;definira 132 kao DUZ
          ORG     $1000       ;LINIJA je na adresi 1000H
LINIJA    DS.B    DUZ         ;rezervira se DUZ (132) okteta memorije
          MOVE.B  #DUZ-1, D0  ;u D0 spremnik upisuje DUZ-1 (131), brojilo
          MOVEA.L #LINIJA, A0 ;u A0 upisuje LINIJU (1000H), početak linije
PETLJA    CMPI.B  #CR, (A0)+  ;uspoređuje oktet sa CR (13) i inkrementira A0
          DBEQ    D0,PETLJA   ;Dvostruko ispitivanje
                              ;ako nije postavljena Z zastavica
                              ;(rezultat prethodne usporedbe nije istina) i
                              ;ako je sadržaj D0 ≠ 0 (nije ispitana cijela linija)
                              ;tada grananje na adresu PETLJA
                              ;inače nastavi sa sljedećom naredbom
```

Ovaj jednostavan programski odsječak može poslužiti za analizu pojedinih direktiva i naredbi. Za uočiti je razliku između direktive **EQU** i **DS.B**. Prva direktiva samo dodjeljuje simboličkoj konstanti vrijednost CR = 13 i DUZ = 132, dok druga direktiva rezervira 132 okteta memorije. Direktivom **ORG** postavlja se početak područja rezerviranog za 132 okteta na adresu 1000H. Također, program prevodilac automatski labeli odnosno imenu LINIJA dodjeljuje vrijednost 1000H.

Sljedeće što je važno uočiti je da je sastavni dio naredbe MOVE.B #DUZ - 1, D0 izraz DUZ – 1 koji se praktički sastoji od dvije konstante, simboličke DUZ = 132 i neposredne 1. Ova naredba je praktički upisivanje neposredne konstante (131) u spremnik D0. Vrijednost

neposredne konstante izračunava se za vrijeme prevođenja programa, a ne tijekom njegova izvođenja.

U programu postoji još i labela PETLJA koja predstavlja adresu na kojoj je smještena naredba CMPI.B #CR, (A0)+. Jedan od zadataka programa prevodioca je da i ovoj labeli dodjeli vrijednost. Postupak se sastoji u sljedećem. Programom je određeno da se 132 okteta rezerviraju počevši od adrese 1000H (memorijski segment od 1000H do 1083H). Iza ovog segmenta slijedi program, što znači da prva naredba programa MOVE.B #DUZ-1, D0 počinje na adresi 1084H. Radi se o naredbi koja neposrednu konstantu dužine okteta upisuje u spremnik. Ova naredba je dužine dvije riječi, odnosno zauzima memorijske lokacije od 1084H do 1087H. Sljedeća naredba MOVEA.L #LINIJA, A0 je naredba koja neposrednu konstantu dužine dvostruke riječi prebacuje u adresni spremnik A0. Ova naredba je dužine tri riječi, odnosno nalazi se na adresama od 1088H do 108DH. Sljedeća naredba počinje na adresi 108E što je ujedno i vrijednost labele PETLJA.



Ovi primjeri pokazuju da postoji niz aktivnosti koje se odvijaju tijekom prevođenja programa. Obično program prevodilac prevodi program u dva prolaza (*two pass assembler*). U prvom prolazu generira se tablica simboličkih imena i njima se dodjeljuju vrijednosti. U drugom prolazu kodiraju se naredbe i upisuju u naredbe stvarne vrijednosti.

```
1000                                    LINIJA  DS.B      DUZ
1084    103C    0083                            MOVE.B    #DUZ-1, D0
1088    207C    0000    1000                    MOVEA.L   #LINIJA, A0
108E    0C18    000D                    PETLJA  CMPI.B    #CR, (A0)+
1093    57C8    FFF8                            DBEQ      D0,PETLJA
```

Drugi primjer je potprogram CLRBLK koji blok riječi postavlja u nulu. Za potprogram potrebno je inicjalizirati broj riječi koje je potrebno postaviti u nulu te početnu adresu bloka. Posebno je potrebno obratiti pažnju na mogućnost da je inicjaliziran broj riječi jednak nuli. Tada se ne poništava ni jedna riječ bloka.

```
            ...
            MOVE.L  #POLJE, A0   ;u A0 početna adresa polja
            MOVE.W  BROJ, D0     ;u D0 broj podataka
            JSR     CLRBLK       ;skok u potprogram
            ...
CLRBL       K                    ;početna adresa potprograma
            BRA     PETLJA1      ;bezuvjetni skok na adresu PETLJA1
PETLJA      CLR.W   (A0)+        ;postavljanje riječi na adresi A0 u 0 i inkr. A0
PETLJA1     DBF     D0, PETLJA   ;dekrementiranje D0 i ako je ≥ 0 grananje na PETLJA
            RTS                  ;povratak iz potprograma
```

3.3.5. Obrada posebnih stanja

Posebna stanja (*exceptions*) su nepredviđeni događaji koji se zbivaju tijekom izvođenja programa. Za njihovu obradu posebne su posebne procedure. Moguće su tri vrste posebnih stanja: prekidi (*interrupts*), zamke (*traps*) i praćenje (*traces*). Prekide uzrokuju vanjski događaji (tipka s tipkovnice, miš, nestanak napajanja i sl.) koji zahtijevaju od procesora da prekine trenutnu obradu i izvede posebnu proceduru (*exception handler*). Zamke su interno generirana posebna stanja kao npr. aritmetički pretek, dijeljenje s nulom, dohvat naredbe koju je nemoguće dekodirati, i sl. Praćenje je zamka najvećeg prioriteta koja se koristi za izvođenje programa naredbu po naredbu. Ovo je značajno prilikom ispravljanja programa.

Sekvenca obrade posebnih stanja. Četiri su koraka pri obradi posebnih stanja:

1. Pratiti i prilagoditi stanje procesora.
2. Odrediti adresu procedure za obradu posebnog stanja.
3. Spremiti staro stanje procesora na stog.
4. Granati se u proceduru za obradu posebnog stanja.

U prvom koraku radi se privremena kopija spremnika stanja. Zatim se postavlja S bit u jedinicu, odnosno procesor u nadzorni mod. Ujedno se T bit postavlja u nulu onemogućavajući mod praćenja tijekom izvođenja procedure za obradu prekida.

Za svako posebno stanje mora postojati procedura za njegovu obradu (*exception handler*). Ova procedura slična je običnom potprogramu. Početna adresa ove procedure određena je

rednim brojem vektora posebnog stanja (*exception vector number*). U drugom koraku određuje se redni broj vektora posebnog stanja. U slučaju prekida ovaj podatak vanjski uređaj dostavlja procesoru odmah nakon što je procesor prihvatio zahtjev za prekidom. U slučaju zamki broj vektora posebnog stanja se generira interno. Ovaj broj je oktet koji nakon posmaka u lijevo za dva mjesta određuje adresu memorijske lokacije na kojoj je upisana adresa prve naredbe procedure za obradu tog posebnog stanja, vektor posebnog stanja (*exception vector*). Ovaj vektor je duga adresa dužine četiri okteta.

Prije nego što se izvrši grananje u proceduru za obradu posebnog stanja, potrebno je prvo zapamtiti stanje procesora koje je bilo prije pojave posebnog stanja kako bi se prekinuti program naknadno mogao nesmetano nastaviti. Minimalno je potrebno zapamtiti stanje programskog brojila (adresa prekinute naredbe) te stanje spremnika stanja. Ove dvije veličine stavljaju je na sistemski stog na koji pokazuje spremnik A7'. U posljednjem koraku vektor prekida upisuje se u programsko brojilo što praktički predstavlja grananje u proceduru za obradu posebnog stanja.

Po završetku obrade potrebno je obnoviti staro stanje procesora na način da se sa stoga prvo obnovi sadržaj spremnika stanja, a zatim i sadržaj programskog brojila. Ovo se izvodi naredbom RTE (*return from exception*) koja radi na sličan način kao RTR naredba. Razlika je što kod RTR naredbe se obnavlja samo sadržaj korisničkog okteta, dok kod naredbe RTE se obnavlja cijela riječ.

Prioriteti prekida. MC68000 dozvoljava podjelu prekida u sedam razina prioriteta. Trenutni prioritet određen je bitovima 8-10 spremnika stanja. Procesor ima tri izlaza na kojima signalizira vanjskim jedinicama trenutnu razinu prioriteta. Prekidi jednakog ili manjeg prioriteta se zanemaruju odnosno odbacuju. Procedura za obradu prekida upisuje razinu prioriteta prekida kojeg obrađuje u spremnik stanja.

Procesor sklopovski ispituje po završetku svake naredbe ulaz za postavljanje zahtjeva za prekidom. Ovakav redoslijed je potreban zato što je potrebno da se tekuća naredba izvrši, a tek onda da se prekine izvođenje. Drugačije rješenje moglo bi ostaviti procesor u nepoznatom stanju te ne bi moguće poslije obrade prekida nastaviti izvođenje prekinutog programa.

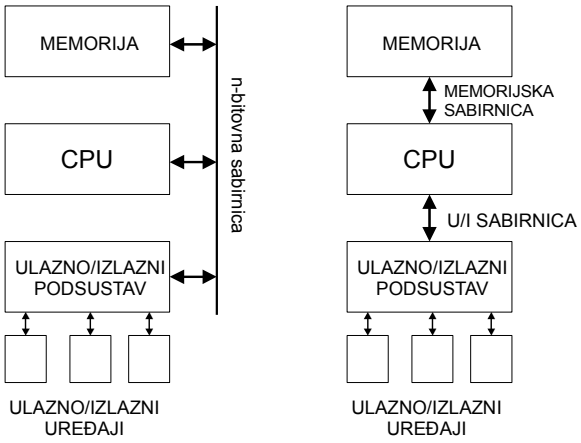
Obrada posebnih stanja unosi određene preinake u RTN opisu procesa interpretacije naredbe. Proširenje opisa zahtijeva dodatne deklaracije u opisu stanja procesora:

```
Reset:                                  Reset ulaz
exc_req:                                zastavica: zahtjev za posebnim stanjem
exc_lev⟨2..0⟩:                          razina posebnog stanja
vect⟨7..0⟩:                             broj vektora posebnog stanja
exc := exc_req ∧ (exc_lev > INT):       postoji zahtjev s prioritetom većim od trenutnog
tmp⟨15..0⟩:                             privremeni spremnik za spremnik stanja

Interpretacija_naredbe := (
    Run ∧ ¬(Reset ∨ exc) → (IR ← Mw[PC]: PC ← PC + 2);      Normalno izvođenje
        Izvođenje_naredbe);
    Reset → (INT⟨2..0⟩ ← 7: S ← 1: T ← 0:                   Reset
        SSP ← Ml[0]: PC ← Ml[4]:
        Reset ← 0: Run ← 1);
```

```
Run ∧ ¬Reset ∧ exc →
(Tmp ←Status: S ← 1: T ← 0;
      (SSP ← SSP - 4; Ml[SSP] ← PC;                PC na stog
      SSP ← SSP - 2; Mw[SSP] ← Tmp;                spremnik stanja na stog
      INT⟨2..0⟩ ← exc_lev⟨2..0⟩:
      PC ← Ml[vect⟨7..0⟩#00₂]);
Interpretacija_naredbe:
```

3.3.6. Ulazno/izlazne aktivnosti kod MC68000

Kod procesora moguće je ulazno/izlazne aktivnosti obavljati na dva načina: preko memorijski mapiranog adresnog prostora ili preko posebnog U/I adresnog prostora, slika.



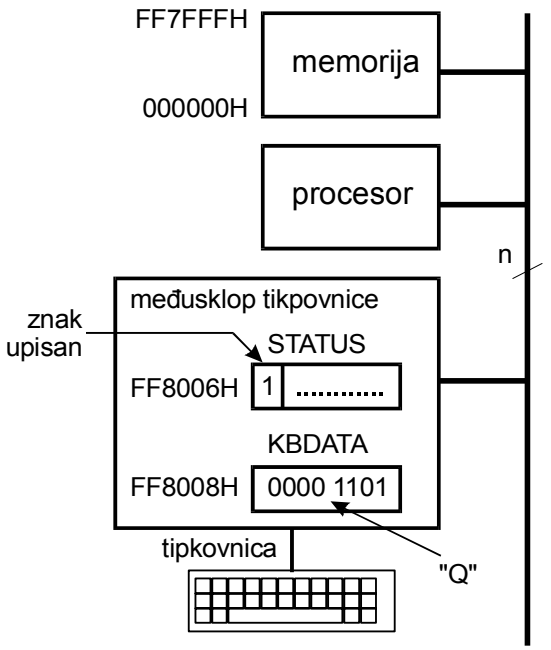
a) memorijski mapiran U/I **b) poseban U/I prostor**

Memorijski mapiran U/I prostor koristi istu sabirnicu za pristup memoriji i ulazno/izlaznim uređajima. MC68000 koristi ovaj koncept. Dio memorijskih adresa dodijeljen je U/I uređajima. Obično se radi o neprekinutom segmentu najviših memorijskih adresa (npr. FF8000H – FFFFFH). Kod ovakvog koncepta pisanje na ili čitanje s U/I uređaja identično je pristupu memoriji.

Poseban U/I prostor koristi jednu sabirnicu za pristup memoriji, a drugu za pristup U/I uređajima. Ovi procesori imaju posebne naredbe za pristup U/I uređajima. Prednost ovakvog pristupa je da se može iskoristiti za memoriju cijela adresna sabirnica te se koristi kod nekih procesora sa 16 ili 24 bitovnom adresnom sabirnicom. Kako većina današnjih procesora ima 24 ili 32 bitovnu adresnu sabirnicu tako da postoji dovoljno mjesta za memoriju i U/I uređaje. Poseban problem je što zasebna U/I sabirnica zahtijeva dodatne izlaze na integriranom sklopu. Projektanti ako već povećavaju broj nožica integriranog sklopa, radije dodjeljuju ove ulaze ili izlaze podatkovnoj ili adresnoj sabirnici. Tako je MC68000 u svojoj prvoj verziji imao 64 nožice, a PowerPC ima sve ukupno 304 nožice od kojih 64 samo za podatkovnu sabirnicu.

Primjer očitanja znaka s tipkovnice. Primjer memorijski mapiranog ulazno/izlaznog uređaja uzeti će se tipkovnica. Kako MC68000 ima 24 bitovnu adresnu sabirnicu adresni prostor će se podijeliti na memorijski, 000000H do FF7FFFH, i U/I, FF8000H do FFFFFFH.

To znači da je gornjih 64K adresnog prostora dodijeljeno U/I uređajima. Tipkovnica je spojena na sabirnicu preko U/I međusklopa koji se sastoji od spremnika stanja i spremnika za podatak svaki veličine jednog okteta. Neka je spremnik stanja postavljen na adresi FF8006H, a spremnik podatka na adresi FF8008H. Kada se s tipkovnice upiše podatak u spremnik podatka tada se ujedno postavlja i msb spremnika stanja u jedinicu. Procesor čita spremnik stanja, ispituje njegov msb i kada ustanovi da je msb = 1 čita podatak iz spremnika podatka. Sklopovi ujedno kada je očitan podatak postavljaju automatski msb spremnika stanja u nulu.



Sljedeći programski odsječak može se koristiti za čitanje znaka s tipkovnice:

```
STATUS   EQU     FF8006H      ;adresa spremnika stanja tipkovnice
KBDATA   EQU     FF8008H      ;adresa spremnika podatka tipkovnice
         ...
RDCHAR   TST.B   STATUS       ;ispituje se stanje tipkovnice
         BPL     RDCHAR       ;ako je pozitivno ponovo ispitivanje
         MOVE.B  KBDATA, D0   ;stanje negativno, čita se znak
         RTS                  ;povratak iz potprograma
```

Za primijetiti je da procesor stalno ispituje status tipkovnice. Ako je msb statusa 0, odnosno njegova vrijednost je pozitivna, podatak nije upisan u spremnik podatka. Ispitivanje se ponavlja sve dok vrijednost spremnika stanja nije negativna (msb = 1). tj. upisan je znak s tipkovnice u spremnik podatka. Procesor sada može pročitati podatak.

Ovakav način komunikacije gdje procesor stalno ispituje stanje U/I uređaja da bi ustanovi da li je U/I uređaj spreman za prijenos informacije naziva se aktivno čekanje. Ako se uzme da je moguće utipkati 5 znakova u sekundi, a neka procesor u prosjeku obrađuje 5 milijuna naredbi u sekundi, slijedi da procesor provede 99.9999% vremena u čekanju. U tom

vremenu možda je bilo moguće provoditi neku drugu obradu. Zato je bolje rješenje da se ovakav prijenos odvija preko zahtjeva za prekidom. Kada se s tipkovnice upiše znak u podatkovni spremnik U/I međusklopa, postavi se procesoru znak za prekidom. Procesor prihvati prekid i pomoću procedure za obradu prekida se pročita znak iz U/I međusklopa. Dok se utipka novi znak procesor može izvoditi neku drugu obradu. Ovo je tzv. obavljanje U/I operacije pomoću zahtjeva za prekidom.

3.4. RISC procesor, SPARC

SPARC (*Scalable Processor ARChitecture*) razvio je Sun Microsystems 1987. godine, ne kao mikroprocesor nego kao posebno rješenje arhitekture računala. SPARC je otvoren sustav u smislu da su brojni proizvođači poluvodičkih sklopova dobili licence za proizvodnju sklopova namijenjenih ovom računalu korištenjem različitih tehnologija, CMOS, ECL, GaAs itd. SPARC International je međunarodni konzorcij koji se sastoji od proizvođača računala, a članstvo je otvoreno i drugim proizvođačima. SPARC specifikacije definiraju spremnike opće namjene, spremnike za podatke prikazane s pomičnim zarezom (realne brojeve), spremnike proširenja procesora (*coprocessor*) te druge jedinice koje definiraju stanje procesora i 69 osnovnih naredbi. Memorijski model je linearan, s 32 bitovnom virtualnom adresom. Adresiraju se okteti memorije, ali se podaci prenose preko 32 bitovne podatkovne sabirnice. Svi novi SPARC procesori su binarno kompatibilni s prethodnom generacijom.

Kroz sljedeća razmatranja opisati će se programerska arhitektura ovog procesora. Kod razmatranja na razini implementacije ograničiti će se na prvi procesor ove arhitekture, Fujitsu MB86900, koji je radio s frekvencijom takta 16.6 MHz i imao prosječne performanse od oko 10 MIPSa. Prema tome prosječno izvođenje naredbe, ciklus, iznosi 1.66 taktova.

Originalna arhitektura SPARCa definira tri procesne jedinice: jedinicu za obradu cjelobrojnih veličina (*integer unit*) IU, jedinicu za obradu brojeva prikazanih s pomičnim zarezom (*floating-point unit*) FPU i proširenje procesora (*coprocessor*) CP. IU je srce procesora. Ona izvodi proračune efektivnih adresa, cjelobrojnu aritmetiku i upravlja programskim tokom. FPU koja neće biti detaljnije analizirana izvodi obradu nad brojevima prikazanim s pomičnim zarezom. Može biti zasebni integrirani sklop ili dio IU. CP izvodi različite korisnički definirane obrade. SPARC arhitektura ne definira u detalje koje su to obrade nego samo definira veze između CP i IU.

Istraživanja provedena 80-tih godina pokazala su da značajan dio prijenosa podataka između memorije i procesora otpada na postupak pohrane stanja procesora prilikom poziva procedura te obnovu starog stanja po povratku iz procedura. Ovaj problem projektanti arhitekture riješili su tako da procesor ima veći broj spremnika opće namjene koji je podijeljen na grupe ili prozore (*register windows*). Samo je jedan prozor u određenom trenutku vidljiv programeru. Samo jedan u dio spremnika opće namjene upisuju se globalne varijable, u drugi ulazni parametri procedure, u treći lokalne varijable procedure, a u četvrti izlazni parametri procedure. Prilikom poziva procedure stvara se novi prozor na način da izlazni spremnici stare procedure postaju ulazni spremnici nove procedure. Ovakav prijenos parametara između procedura ne zahtijeva nikakav fizički prijenos podataka te ne troši vrijeme procesora. Ukoliko je potrebno prenjeti iz jedne procedure u drugu veći broj parametara tada je potrebno koristiti memoriju, odnosno stog.

Povećani broj spremnika opće namjene rezultirao je potrebom da polja naredbe koja ih specificiraju budu veća.

Iako definicija SPARC arhitekture ne zadire u razinu implementacije ipak se kompromis napravi kako bi se mogao realizirati cjevovod. Tako ova arhitektura definira odgođeno grananje.

3.4. RISC procesor, SPARC

SPARC (*Scalable Processor ARChitecture*) razvio je Sun Microsystems 1987. godine, ne kao mikroprocesor nego kao posebno rješenje arhitekture računala. SPARC je otvoren sustav u smislu da su brojni proizvođači poluvodičkih sklopova dobili licence za proizvodnju sklopova namijenjenih ovom računalu korištenjem različitih tehnologija, CMOS, ECL, GaAs itd. SPARC International je međunarodni konzorcij koji se sastoji od proizvođača računala, a članstvo je otvoreno i drugim proizvođačima. SPARC specifikacije definiraju spremnike opće namjene, spremnike za podatke prikazane s pomičnim zarezom (realne brojeve), spremnike proširenja procesora (*coprocessor*) te druge jedinice koje definiraju stanje procesora i 69 osnovnih naredbi. Memorijski model je linearan, s 32 bitovnom virtualnom adresom. Adresiraju se okteti memorije, ali se podaci prenose preko 32 bitovne podatkovne sabirnice. Svi novi SPARC procesori su binarno kompatibilni s prethodnom generacijom.

Kroz sljedeća razmatranja opisati će se programerska arhitektura ovog procesora. Kod razmatranja na razini implementacije ograničiti će se na prvi procesor ove arhitekture, Fujitsu MB86900, koji je radio s frekvencijom takta 16.6 MHz i imao prosječne performanse od oko 10 MIPSa. Prema tome prosječno izvođenje naredbe, ciklus, iznosi 1.66 taktova.

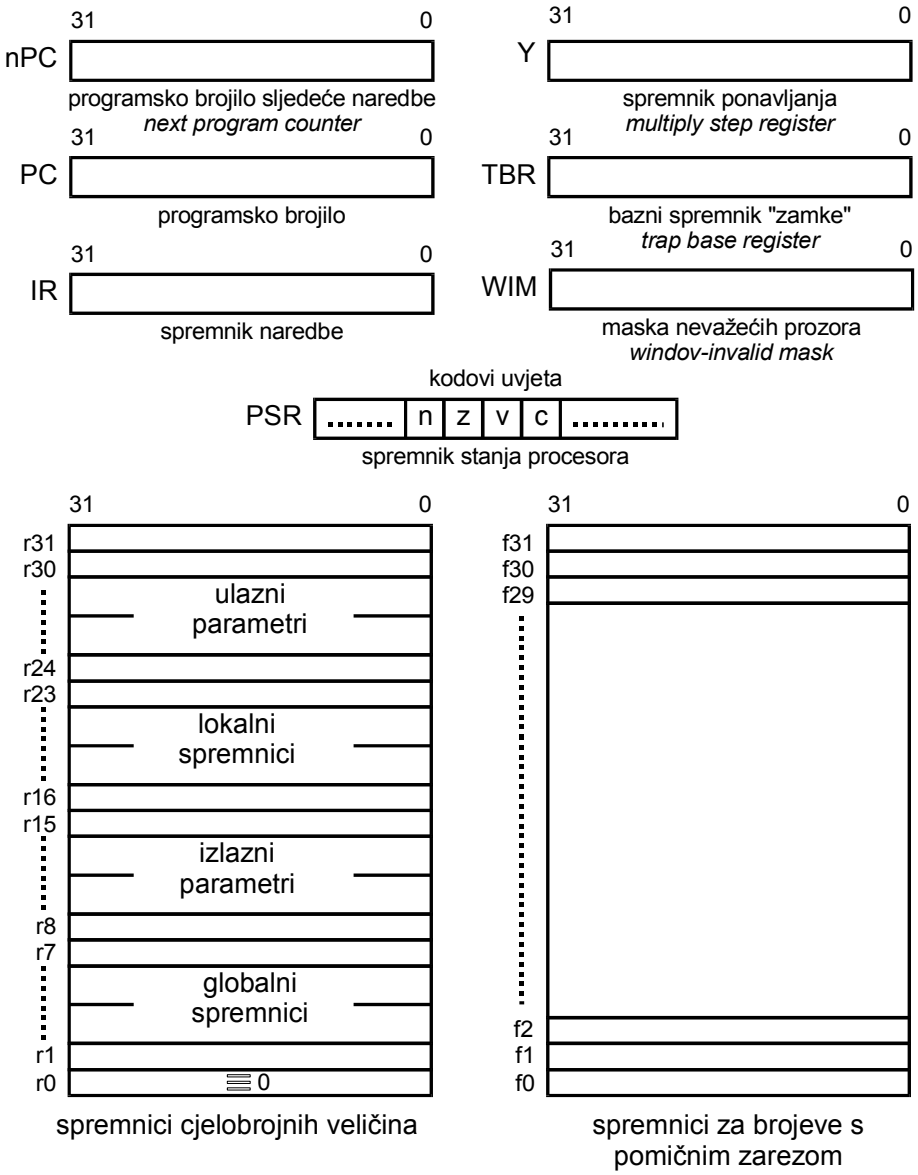
Originalna arhitektura SPARCa definira tri procesne jedinice: jedinicu za obradu cjelobrojnih veličina (*integer unit*) IU, jedinicu za obradu brojeva prikazanih s pomičnim zarezom (*floating-point unit*) FPU i proširenje procesora (*coprocessor*) CP. IU je srce procesora. Ona izvodi proračune efektivnih adresa, cjelobrojnu aritmetiku i upravlja programskim tokom. FPU koja neće biti detaljnije analizirana izvodi obradu nad brojevima prikazanim s pomičnim zarezom. Može biti zasebni integrirani sklop ili dio IU. CP izvodi različite korisnički definirane obrade. SPARC arhitektura ne definira u detalje koje su to obrade nego samo definira veze između CP i IU.

Istraživanja provedena 80-tih godina pokazala su da značajan dio prijenosa podataka između memorije i procesora otpada na postupak pohrane stanja procesora prilikom poziva procedura te obnovu starog stanja po povratku iz procedura. Ovaj problem projektanti arhitekture riješili su tako da procesor ima veći broj spremnika opće namjene koji je podijeljen na grupe ili prozore (*register windows*). Samo je jedan prozor u određenom trenutku vidljiv programeru. Samo jedan u dio spremnika opće namjene upisuju se globalne varijable, u drugi ulazni parametri procedure, u treći lokalne varijable procedure, a u četvrti izlazni parametri procedure. Prilikom poziva procedure stvara se novi prozor na način da izlazni spremnici stare procedure postaju ulazni spremnici nove procedure. Ovakav prijenos parametara između procedura ne zahtijeva nikakav fizički prijenos podataka te ne troši vrijeme procesora. Ukoliko je potrebno prenjeti iz jedne procedure u drugu veći broj parametara tada je potrebno koristiti memoriju, odnosno stog. Povećani broj spremnika opće namjene rezultirao je potrebom da polja naredbe koja ih specificiraju budu veća.

Iako definicija SPARC arhitekture ne zadire u razinu implementacije ipak se kompromis napravi kako bi se mogao realizirati cjevovod. Tako ova arhitektura definira odgođeno grananje.

3.4.1. SPARC arhitektura procesora i memorije

Ako se zanemari koncept prozora spremnika, programerski model SPARC procesora je tipičan model RISCa. Njegovi spremnici opće namjene dijele se na spremnika za pohranu cjelobrojnih veličina i na spremnike za pohranu brojeva s pomičnim zarezom. Odgođeno grananje zahtijeva upotrebu dvaju programskih brojila: PC koje sadrži adresu naredbe koja se trenutno izvodi, te nPC koji sadrži adresu sljedeće naredbe koju treba izvesti. Stanje procesora, koje uključuje 32 spremnika opće namjene trenutno dostupnih programeru, jedan prozor spremnika, je prikazan na sljedećoj slici.

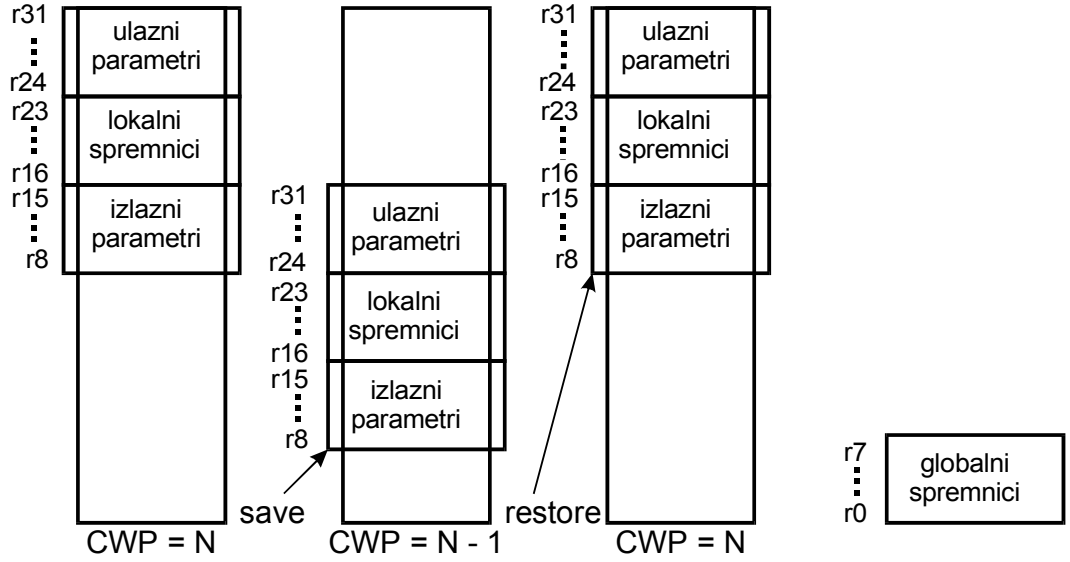


Kao kod svih RISCova sve naredbe SPARC procesora jednake su dužine, 32 bita i upisuju se u spremnik naredbe. Spremnik za opis stanja procesora (*processor state register*) PSR, sadrži kodove, zastavice uvjeta: negativan (n), nula (z), pretek (v), prijenos (c).U spremniku stanja procesora između ostaloga upisan je i pokazivač na trenutni prozor spremnika te zastavice koje koriste prekidi i zamke. Spremnik maske nevažećih prozora (WIM) koristi sustavu za rad s prozorima spremnika, spremnik baze "zamki" (TBR) kod obrade "zamki" i prekida, te spremnik ponavljanja (Y) kod naredbi koje se izvode u više koraka. Spremnici za pohranu brojeva s pomičnim zarezom mogu uskladištiti 32 podatka jednostruke preciznosti (32 bita) ili 16 podataka dvostruke preciznosti (64 bita) ili 8 podataka četverostruke preciznosti (128 bita).

Spremnici ovog procesora mogu se opisati sljedećim RTN opisom:

PC⟨31..0⟩:	pokazivač na naredbu koja se trenutno izvodi
nPC⟨31..0⟩:	pokazivač na sljedeću naredbu
IR⟨31..0⟩:	spremnik naredbe
r[0..31]⟨31..0⟩:	spremnici opće namjene za cjelobrojne varijable
f[0..31]⟨31..0⟩:	spremnici za varijable prikazane s pomičnim zarezom
WIM⟨31..0⟩:	spremnik maske nevažećih prozora
TBR⟨31..0⟩:	spremnik baze zamki
Y⟨31..0⟩:	spremnik ponavljanja
PSR⟨..⟩:	spremnik stanja procesora
CCR⟨3..0⟩:= PSR⟨..⟩:	spremnik kodova uvjeta
n:=CCR⟨3⟩:	zastavica negativan
z:=CCR⟨2⟩:	zastavica nula
v:=CCR⟨1⟩:	zastavica pretek
c:=CCR⟨0⟩:	zastavica prijenos

Prozori spremnika. Koncept prozora spremnika temelj je programerskog modela SPARC procesora. Sljedeća slika prikazuje kako funkcionira sustav prozora spremnika.



Ukupan broj spremnika ovisi o realizaciji procesora, ali kod prve verzije SPARC procesora bilo je 120 spremnika. Osam spremnika, r0 – r7, rezervirani su za globalne varijable i dostupni su svim procedurama. U spremnik r0 trajno je zapisana vrijednost nula pa čitanje iz tog spremnika uvijek daje nulu, a pisanju u njega nema učinka.

Preostalih 112 spremnika pripadaju sustavu prozora spremnika. Osam posljednjih spremnika prozora koriste se za ulazne varijable u proceduru. Sljedećih osam procedura koristi za lokalne varijable, a prvih osam spremnika za prijenos varijabli proceduri koju poziva, odnosno za izlazne varijable. Sustav ima pokazivač na aktivni prozor (*current window pointer*), CWP, koji je nevidljiv korisniku, a vidi ga samo sistem programer. CWP se dekrementira naredbom **save** nakon koje izlazni spremnici postaju ulazni spremnici nove procedure. Naredbom **restore** inkremetira se sadržaj CWP, odnosno obnavlja se stanje procesora prije poziva procedure. Prilikom poziva procedure sadržaj programskog brojila, odnosno adresa na koju se potrebno vratiti nakon završetka procedure, pohranjena je u r15, odnosno nalazi se u r31 ulaska u proceduru.

Pokušaj izvođenja naredbe **save** kada su svi prozori u upotrebi uzrokuje posebno stanje ili zamku te rezultira u izbacivanju sadržaja prvog prozora i stavljanju istoga na stog. Slično kada se izvede **restore** naredba, a aktivan je prvi prozor, ponovo dolazi do posebnog stanja i stanje procesora se obnavlja skidanjem sadržaja spremnika sa stoga.

Neki detalji aktivnosti vezanih uz promjene stanja procesora opisane su pomoću RTN na sljedeći način:

WR[0..511]⟨31..0⟩:	maksimalni broj prozora spremnika
WIM⟨31..0⟩:	maska nevažećih prozora
CWP⟨4..0⟩:	tekući pokazivač prozora (dio PSR)
wrb⟨8..0⟩:=CWP#0000_2:	baza prozora spremnika
r[0..7]:=g[0..7]:	spremnici r0 do r7 su za globalne varijable g0 do g7
g[0]⟨31..0⟩:=0:	spremnik g0 za globalne varijable je jednak nuli
r[8..31]:=WR[wrb..wrb+23]	posljednja 24 spremnika mijenjaju se sa CWP

save → ((WIM⟨CWP-2⟩ = 1) → window_ov:
((WIM⟨CWP-2⟩ = 0) → (WIM⟨CWP-2⟩ ← 1): (CWP ← CWP - 1)):

restore → ((WIM⟨CWP+2⟩ = 1) → window_un:
((WIM⟨CWP+2⟩ = 0) → (WIM⟨CWP-2⟩ ← 0): (CWP ← CWP + 1)):

Naredbe **save** i **restore** koriste maku nevažećih prozora kako bi detektirali da nema više slobodnih prozora (*windows overflow*) windows_ov, odnosno da su svi prozori prazni (*windows underflow*) windows_un. Ove naredbe mijenjaju sadržaj WIM i CWP kako bi pratile koji je prozor aktivan te koji su prozori zauzeti. Kako izlazni parametri novog prozora mogu kod **store** naredbe prepisati ulazne parametre nekog prozora koji pripada nekoj starijoj proceduri potrebno je ispitati ne samo prozor iza, nego i dva prozora iza aktivnog. To se realizira ispitivanjem bita maske nevažećih prozora WIM⟨CWP-2⟩ koji se nalazi na mjestu CWP –2. Ako je on jednak jedinici tada je taj prozor zauzet te ga se mora prvo sačuvati, a tek onda dozvoliti prijelaz na novi prozor. Slično razmatranje vrijedi za naredbu **restore**. Pokazivač CWP ima pet bita te kod inkrementiranja vrijednosti 11111 prelazi automatski na vrijednost 00000. Slično vrijedi za njegovo dekrementiranje.

Glavna memorija. Memorijska adresa je određena s 32 bita. Memorijski prostor je linearan i sastoji se od 2^{32} okteta. Varijable duže od jednog okteta pohranjuju se po principu većeg kraja (*big endian*). Oktet na najmanjoj memorijskoj lokaciji sadrži msb, a oktet na navišoj memorijskoj lokaciji lsb. Memorija se opisuje na način:

Mb[0..2^{32}-1]⟨7..0⟩:	memorija u oktetima
Mh[a]⟨15..0⟩:=Mh[a]⟨7..0⟩#Mh[a+1]⟨7..0⟩:	pola riječi
Mw[a]⟨32..0⟩:=Mh[a]⟨7..0⟩#Mh[a+1]⟨7..0⟩#Mh[a+2]⟨7..0⟩#Mh[a+3]⟨7..0⟩:	riječ

SPARC arhitektura definira i jedinicu za preslikavanje (mapiranje) memorije (*Memory Mapping Unit*) MMU. Ova jedinica omogućava proširenje memorije na više adresnih prostora ili segmenata. Samo jedan segment dostupan je korisniku dok operacijski sustav može pristupati cijeloj memoriji. Pojedinim adresnom segmentu pristupa se pomoću osam bitovnog indikatora (*address space indicator*). Osim funkcije proširenja memorijskog prostora ova jedinica ima i funkciju zaštite dijelova memorije od neautoriziranog pristupa.

3.4.2. Operandi, formati naredbi i njihova interpretacija

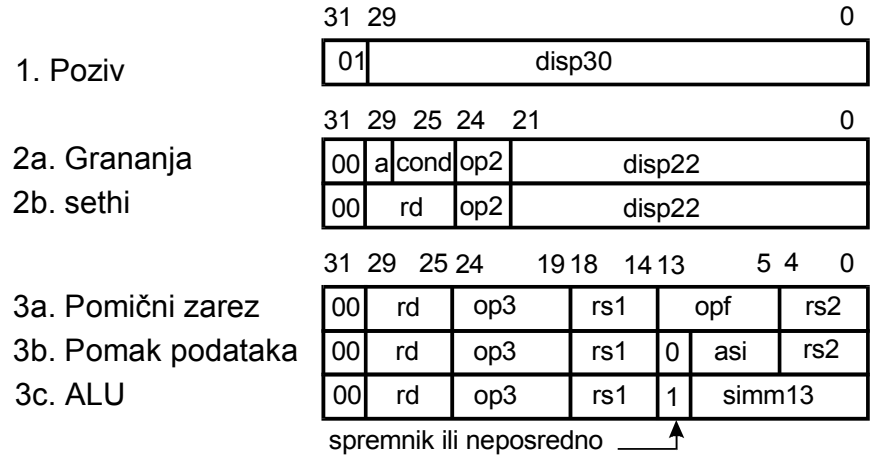
Naredbe mogu pristupati podacima dužine 16 bita (pola riječi), 32 bita (riječ) i 64 bita (dvostruka riječ). Riječi i podaci veće dužine moraju biti smješteni na adresama djeljivim sa četiri, odnosno posljednja dva bita adrese najmanjeg značenja moraju biti jednaka nuli. Naredbe obrađuju cjelobrojne veličine dužine okteta, pola riječi, riječi i dvostruke riječi bez i s predznakom u aritmetici 2'kog komplementa te 32, 64 i 128 bitovne brojeve s pomičnim zarezom.

SPARC također podržava sklopovski i oznaku tipa podataka. Posljednja dva bita označenog podatka rezervirana su za oznaku tipu podatka. Postoje naredbe koje ispituju oznaku tipa podatka i u ovisnosti o rezultatu postavljaju u jedinicu zastavicu preteka, a konačni rezultat može biti prekid zbog preteka. Ovo svojstvo korisna je podrška višim jezicima kao što su Lisp, Smalltalk i Prolog kod kojih se tipiziraju podatci za vrijeme izvođenja programa.

Formati naredbi. I kod ovog procesora format naredbe zahtijeva složeni postupak interpretacije iako ne u istoj mjeri kao kod CISC procesora. SPARC ima 55 osnovnih naredbi sa cjelobrojnim veličinama kao i 14 naredbi s brojevima s pomičnim zarezom. Također postoji nekoliko formata naredbi za dodatni procesor koje neće biti razmatrane. Većina naredbi koristi jedan od tri formata prikazana na slici. Informacije o poljima naredbe opisana su na sljedeći način:

op⟨1..0⟩:=IR⟨31..30⟩:	operacijski kod
disp30⟨29..0⟩:= IR⟨29..0⟩:	pomak za poziv
a:= IR⟨29⟩:	bit poništavanja kod grananja
con⟨3..0⟩:= IR⟨28..25⟩:	uvjet grananja
rd⟨4..0⟩:= IR⟨29..25⟩:	odredišni spremnik
op2⟨2..0⟩:= IR⟨24..22⟩:	operacijski kod
disp22⟨29..0⟩:= IR⟨21..0⟩:	konstanta kod grananja ili sethi
op3⟨5..0⟩:= IR⟨24..19⟩:	operacijski kod
rs1⟨4..0⟩:= IR⟨18..14⟩:	izvorišni spremnik br. 1
opf⟨8..0⟩:= IR⟨13..5⟩:	dopunski operacijski kod za pomični zarez
i:= IR⟨13⟩:	indikator neposrednog operanda

simm13⟨12..0⟩:= IR⟨12..0⟩: direktni operand s predznakom
rs2⟨4..0⟩:= IR⟨4..0⟩: izvorišni spremnik br. 2



Adresni modovi. Prema svojoj prirodi, SPARC dozvoljava pristup memoriji samo preko naredbi upiši, spremi. Broj mogućih adresnih modova je značajno manji nego kod CISC procesora kao npr. MC68000. Dozvoljena su samo dva adresna moda:

1. spremnik + spremnik
2. spremnik + direktna 13 bitovna konstanta {proširenje predznaka, 2'ki komplement}

Dva dodatna moda koriste kod grananja i poziva procedura.

Procedura koristi 8 globalnih i 24 njoj pripadajuća spremnika za proračun adrese direktno, preko pomaka, spremnika indirektno, indeksnog adresiranja. Tako neposredno adresiranje realizira se drugim adresnim modom korištenjem spremnika g0. Tim modom moguće je pristupiti samo prvim i posljednjim 4K memorije ($\pm 2^{12}$ adresa). Ukoliko se spremnik g0 koristi u prvom adresnom modu adresu operanda određuje samo sadržaj drugog spremnika. Za realizaciju relativnih adresnih modova potrebno je pročitati vrijednost programskog brojila. To je moguće pomoću naredbe skoči i poveži (*jump & link*).

Grananja i pozivi procedura koriste relativne adresne modove. Kako naredbe su smještene na adresama djeljivim sa četiri (ograničenje na pristup riječi u memoriji) tako 30 bitovna konstanta, koja je sastavni dio naredbe za poziv procedure, dovoljna je za pristup cijelom adresnom prostoru (posljednja dva bita su uvijek nula). Slično vrijedi i kod grananja gdje 22 bitovna konstanta omogućava grananje unutar $\pm 2^{23}$ = 8Mb od trenutne naredbe. Adresni modovi mogu se opisati na sljedeći način:

adr⟨31..0⟩ := (i = 0 → r[rs1] + r[rs2]: prvi adresni mod
i = 1 → r[rs1] + simm13⟨12..0⟩): drugi adresni mod

calladr⟨31..0⟩ := PC⟨31..0⟩ + disp30#00_2: poziv procedure

bradr⟨31..0⟩ := PC⟨31..0⟩ + disp22#00_2: grananje

3.4.3. Skup naredbi SPARC procesora

Interpretacija naredbe kod SPARC procesora složena je iz razloga postojanja posebnih stanja te dva programska brojila potrebna zbog odgođenog grananja. Obrada posebnih stanja ili zamki te odgođena grananja naknadno će biti opisani. Tako je grubi opis procesa interpretacije naredbe kod SPARC procesora sličan kao i kod MC68000:

interpretacija_naredbe := (IR ← M[PC]; izvođenje_naredbe;
nova_vrijednost_PC_ili nPC; interpretacija_naredbe):

SPARC procesor također ima tri osnovna tipa naredbi: za prebacivanje podataka, aritmetičke i logičke naredbe i upravljanja programskim tokom. Naredbe s brojevima prikazanim s pomičnim zarezom se neće posebno razmatrati. Ove naredbe slične su naredbama s cjelobrojnim veličinama s razlikom što koriste posebne spremnike. Također neće se obrađivati naredbe koje koristi proširenje procesora (*coprocessor*) kao ni privilegiranje naredbe koje koristi operacijski sustav u nadzornom modu rada.

Naredbe za prebacivanje podataka. SPARC procesor pristupa memoriji samo pomoću naredbi za čitanje i pisanje. Jedina iznimka je naredba za zamjenu, **swap**, kao i neke privilegirane naredbe koje upravljaju zavisnim procesima.

Naredba	Op.	Op3	Značenje
ldsb	11	00 1001	Pročitaj oktet s predznakom
ldsh	11	00 1010	Pročitaj pola riječi s predznakom
ldsw	11	00 1000	Pročitaj riječ s predznakom
ldub	11	00 0001	Pročitaj oktet bez predznaka
lduh	11	00 0010	Pročitaj pola riječi bez predznaka
ldd	11	00 0011	Pročitaj dvostruku riječ
stb	11	00 0101	Upiši oktet
sth	11	00 0110	Upiši polariječi
stw	11	00 0100	Upiši riječ
std	11	00 0111	Upiši dvostruku riječ
swap	11	00 1111	Izmjeni sadržaje spremnika i memorije
or	10	00 0010	r[d] ← r[s1] OR (r[rs2] ili trenutno)
sethi	00	Op2=100	Gornjih 22 bita od Rdst ← disp22

Osim za **or** i **sethi**, vrijede oba adresna moda

Za primijetiti je da SPARC procesor podržava prijenos okteta, pola riječi, riječ i dvostruke riječi. Dozvoljena su oba adresna moda specificirana 13 bitom naredbe. Podaci se upisuju u spremnik rd na način da kada se upisuje oktet on zauzima osam bita spremnika najmanjeg značenja a ostatak spremnika se postavlja u nulu ili se proširi s predznakom u ovisnosti da li se radi o prebacivanju bez ili s predznakom. Naredba **ldd** mora specificirati parni spremnik, a dvostruka riječ se upisuje u rd (riječ većeg značenja) i u rd + 1 (riječ manjeg značenja). Naredba **swap** zamjenjuje sadržaj spremnika i memorijske lokacije.

Podrška prijenosu dijelova riječi, okteta i pola riječi, rezultirala je sa šest naredbi za upis u i čitanje iz memorije kao i složenijim sklopovima koji povezuju procesor i memoriju. Ukoliko bi međusklop koji povezuje procesor i memoriju podržavao samo 32 bitovni prijenos podataka tada bi naredba za upis okteta u memoriju, **stb**, zahtijevala da procesor prvo pročita riječ iz memorije, zamijeni odgovarajući oktet, te ponovo upiše riječ u memoriju. Zato neki RISC procesori, kako bi izbjegli dodatne naredbe i složenije sklopovlje za povezivanje procesora i memorije, ne podržavaju prijenos dijela riječi. Kod navedenih

procesora prijenos dijela riječi iz spremnika u memoriju realizira se čitanjem memorijske riječi, njenom obradom logičkim operacijama i posmakom i upisom rezultat na odabranu lokaciju. Alpha arhitektura, Digital Equipment Corporation je primjer RISCa koji koristi opisano rješenje.

Može se postaviti pitanje zašto je naredba **or** smještena s naredbama za pomak podataka. Praktički ukoliko se napravi logička operacija ILI sadržaja spremnika rs1 i sadržaja spremnika r0, koji je uvijek jednak nuli, te se rezultat spremi u rezultat u spremnik r2 je identično prebacivanju sadržaja spremnika r1 u spremnik r2.

Naredba **sethi** koristi se za upis neposredne konstante u 22 bita većeg značenja spremnika. Bitovi manjeg značenja 0 do 9 postavljaju se u nulu. Ovom naredbom može se realizirati naredba **nop** odabirom spremnika r0 (rd = 0 i disp22 = 0).

Aritmetičke naredbe. Sljedeća tablica prikazuje aritmetičke naredbe SPARC procesora.

Naredba	Op.	op3	Značenje
add	10	0S 0000	Zbroji, postavi ili ne uvjetni kod (CC)
addx	10	0S 1000	Zbroji s prijenosom: postavi ili ne CCs
sub	10	0S 0100	Oduzmi, postavi ili ne uvjetni kod
subx	10	0S 1100	Oduzmi s posudbom,i postavi ili ne CC
mulscc	10	10 1100	Jedan korak množenja

Postavi Kod uvjeta CC ako je S = 1. Naredbe **addcc**, **addxcc**, **subcc**, **subxcc**.
Naredbe koriste format 3: r[rd] $\leftarrow$ r[rs1] op r[rs2] ili neposredno

Drugi operand cjelobrojne aritmetike određuje se na identičan način kao kod proračuna adrese kod naredni za prijenos podataka. Ako je 13 bit naredbe (format 3c) postavljen u nulu tada je drugi operand sadržaj spremnika r[rs2], a ako je jedinica tada je neposredna 13 bitovna konstanta, simm13. Određivanje drugog operanda tipične aritmetičke operacije, **sub**, opisan je na sljedeći način:

opnd2$\langle$31..0$\rangle$:= ((i = 0) $\rightarrow$ r[rs2]: (i = 1) $\rightarrow$ simm13$\langle$12..0$\rangle${proširenje predznaka}):
sub (:= op = $10_2 \wedge$ op3 = 000100_2) $\rightarrow$ r[rd] $\leftarrow$ r[rs1] – opnd2:

Kod SPARC arhitekture prihvaćen je koncept da programer odlučuje hoće li ili ne naredba postavljati kod uvjeta. Ako je S bit op3 polja postavljen u nulu naredba ne mijenja kod uvjeta, a ako je S bit postavljen u jedinicu tad naredba mijenja četiri bita koda uvjeta (n, z, v, i c). Naredbe **addx** i **subx** koriste se za realizaciju aritmetike veće preciznosti. Tako **addx** zbraja r[rs1] + opnd + c, a **subx** r[rs1] - opnd – c.

Originalna SPARC arhitektura ne podržava naredbe za množenje i dijeljenje. Njih se realizira ili programski ili pomoću jedinice za obradu brojeva s pomičnim zarezom. Postoji samo određena podrška programskoj realizaciji množenja pomoću naredbe **mulscc** koja izvodi jedan korak u operaciji množenja tehnikom posmaka i zbrajanja. Ova procedura biti će detaljnije objašnjena kada se bude obrađivala aritmetička i logička jedinica procesora.

RISC procesori imaju tendenciju da naredbe koje se obavljaju u više koraka prebace na jedinicu za obradu brojeva s pomičnim zarezom gdje su takve naredbe neizbježne. Iz tog razloga ne postoje naredbe za množenje i dijeljenje u originalnoj SPARC arhitekturi. Realizacija množenja i dijeljenja sklopovski, čime bi se ove naredbe mogle realizirati u jednom koraku, dosta je složena i zauzima dosta prostora na integriranom sklopu. Kako se

stupanj integracije povećao, sve više RISC procesora imalo je sklopovsko množenje i dijeljenje kao sastavni dio svoje arhitekture.

Sljedeća tablica prikazuje SPARC logičke naredbe i naredbe za posmak.

Naredba	Op	op3	Značenje
AND	10	0S 0001	I, postavi (S=1) ili ne (S=0) CCs
ANDN	10	0S 0101	NI, postavi ili ne CCs
OR	10	0S 0010	ILI, postavi ili ne CCs
ORN	10	0S 0110	NILI, postavi ili ne CCs
XOR	10	0S 0011	Ekskluzivno ILI, postavi ili ne CCs
XNOR	10	0S 0011	NE Ekskluzivno ILI, postavi ili ne CCs
SLL	10	10 0101	Posmak u lijevo logički, broj u RSRC2 ili imm13
SRL	10	10 0110	Posmak u desno logički, broj u RSRC2 ili imm13
SRA	10	10 0111	Posmak u desno aritmetički, broj u RSRC2 ili imm13

Naredbe za posmak pomiču sadržaj spremnika r[rs1] za broj mjesta specificiran ili spremnikom ili neposrednom konstantom i upisuje rezultat u spremnik r[rd]. Broj mjesta posmaka određen je samo s pet bita najmanjeg značenja spremnika ili neposredne konstante. Za razliku od aritmetičkih naredbi, naredbe za posmak ne mijenjaju kod uvjeta. SPARC procesori nemaju naredbe za rotaciju.

Broj logičkih naredbi je smanjen bez značajnijeg utjecaja na mogućnosti procesora. TAko npr. **not** naredba je izostavljena jer je istu moguće realizirati pomoću **orn** naredbe spremnika sa spremnikom g0.

Naredbe za upravljene programskim tokom. SPARC procesor ima bogat skup naredbi za grananja kao i nekoliko naredbi za pozive procedura. tablica.

Naredba	Format	Op	Op2 or Op3	Značenje
ba	2	00	010	Bezuvjetno grananje
bcc	2	00	010	Uvjetno grananje
call	1	01		Poziv i spremi PC u R15
jmpl	3	10	11 1000	Granaj na EA, spremi PC u Rdst
save	3	10	11 1100	Novi prozor spremnika, Izvedi ADD
restore	3	10	11 1101	Vrati prozor spremnika, Izvedi ADD

Kodovi uvjeta prikazani su sljedećom tablicom

Naredba	Uvjet	Naredba	Uvjet	Naredba	Uvjet	Naredba	Uvjet
ba	1000	bne	1001	be	0001	ble	0010
bcc	1101	bcs	0101	bneg	0110	bvc	1111
bvs	0111						

Kodovi uvjeta su četiri bitovno polje koje je dio naredbe (format 2). Bezuvjetno grananje, **ba**, je samo jedna od naredbi uvjetnog grananja s uvjetnim kodom 1000. Adresa grananja dobiva se kao pomak u odnosu na trenutnu vrijednost programskog brojila.

SPARC koristi odgođeno grananje, ali uz jednu razliku od drugih procesora. Kod primjene cjevovoda naredbe koja slijedi naredbu za grananje je u određenom stadiju obrade u trenutku završetka obrade naredbe za grananje. U slučaju da je potrebno izvesti grananje potrebno je odbaciti sljedeću naredbu i izvođenje prebaciti na adresu grananja. Kako je sljedeća naredba već u fazi obrade i ona je možda promijenila stanje procesora (sadržaj nekog spremnika), program se neće ispravno nastaviti. Zato, ako naredba koja slijedi naredbu za grananje može utjecati na ispravan nastavak izvođenja programa potrebno je

između ovih naredbi ubaciti naredbu koja ništa ne radi, **nop**. Njena zadaća je da odgodi izvođenje sljedeće naredbe.

Kod SPARC procesora naredba za grananje koristi i zastavicu za poništavanje (*annul bit*), a, koji omogućava programeru da upravlja izvođenjem naredbe koja slijedi iza naredbe za grananje. Rješenje se sastoji u sljedećem. Ako je rezultat usporedbe da se grananje izvodi, izvodi se naredba koja je upisna iza naredbe za grananje. Ako je rezultat usporedbe negativan i grananje se ne izvodi tada, ako je a zastavica 1, odbacuje se sljedeća naredbe, odnosno ako je a zastavica 0 sljedeća naredba se izvodi.

Postavljenjem a zastavice u 1, programer ili program prevodilac umetanjem iza naredbe za grananje naredbu koja se izvodi ukoliko dolazi do grananja praktički realizira da ukoliko je rezultat ispitivanja pozitivan program se nesmetano nastavlja na adresi grananja, a ukoliko je rezultat negativan i ne dođe do grananja naredba se odbacuje. Ako ja a zastavica nula, tada programer ili program prevodilac može iza naredbe za grananje ubaciti naredbu koja inače prethodi naredbi za grananje. Ova naredba će se uvijek izvršiti. Iznimaka ja **ba** naredba. Ako je a zastavica 1 naredba koja slijedi naredbu za grananje se poništava, a ako je 0 sljedeća naredba se izvršava. Programer postavlja a zastavicu u 1 ubacujući oznaku a u naredbu: **bcc**, a, disp.

Postoje dvije naredbe koje se koriste za pozive procedura: **call** i **jmpl**. Naredba **call** koristi 30 bitovni pomak upisan neposredno u naredbi koji se pomakne u lijevo za dva mjesta i određuje 32 bitovnu adresu procedure koja se poziva. Ovom naredbom pristupa se cijelom adresnom prostoru. Programsko brojilo odnosno povratna adresa upisuje se u spremnik r15 tekućeg prozora. Naredba **jmpl** koristi format broj 3 i računa adresu grananja za vrijeme izvođenja programa bilo indirektno pomoću spremnika ili indeksnim adresiranjem. Povratna adresa upisuje se u spremnik specificiran s rd. **jmpl** naredba koristi se i za povratak iz procedure pozvane naredbom **call**. Praktički naredba **ret** sintetizira se naredbom: **jmpl** %r15, %r0, granaj se na adresu određenu spremnikom r15 i upiši povratnu adresu u r0 (u r0 se ne prihvaća nikakav upis).

Naredbe **save** i **restore** upravljaju prozorima spremnika kako je već opisano. **save** naredba prebacuje na novi prozor, a **restore** vraća stari. Ako nisu svi prozori popunjeni ili nijedan iskorišten tada naredba izvodi i zbrajanje, **add** sadržaja spremnika specificiranih s rs1 i rs2 iz tekućeg prozora i rezultat upisuje u rd novog prozora. Ova naredba prikladna je za prilagodbu kazala stoga kako bi se rezervirao prostor za parametre koji se prenose preko stoga.

3.4.4. Primjer programa za SPARC procesor

Linija simboličkog programa izgleda na sljedeći način:

labela: naredba !komentar

Labela nije obavezna kao ni komentar koji koristi samo korisniku. Naredba se sastoji od oznake operacije iza koje slijede operandi i rezultat. Rezultat je upisan skroz desno. Spremnici se označavaju s oznakom %r, %g (globalni), %o (izlazni), %l (lokalni) i %i (izlazni) iza koje slijedi redni broj spremnika. Također moguće su sljedeće alternative: izlazni spremnici %o0 - %o7 su i %r8 - %r15, lokalni spremnici %l0 - %l7 su %r16 -%r23 te ulazni spremnici %i0 - %i7 su %r24 - %r31. Primjer naredbe za zbrajanje je:

add %l2, %g1, %o3 ! r[11] ← r[18] + r[1]:

Memorijska adresa zatvorena uglatim zagradama označava pristup memoriji kao npr.:

```
ld      [%r4 + %r5], %r6        ! r[6] ← M[r[4] + r[5]]:
st      %r2, [%r3 + 28]         ! M[r[3] + 28] ← r[2]:
```

Adrese u naredbama za grananja i pozive procedura nisu omeđene uglatim zagradama. Adrese grananja i poziva su obično labele čiju vrijednost proračuna program prevodilac i tijekom prevođenja je upiše u polje naredbe određeno za tu namjenu.

Pseudo operacije ili direktive programu prevodiocu počinju s točkom. Neke od njih su **.begin**, početak prevođenja, **.org**, postavi sljedeći kod na adresu, **.equ**, pridruži simbolu vrijednost, **.end**, kraj prevođenja.

Procedura za zbrajanje dva broja opisana je sljedećim kodom:

```
        .begin
        .org
prog:   ld      [x], %r1        !upiši riječ iz memorije M[x] u spremnik r1
        ld      [y], %r2        !upiši riječ iz memorije M[y] u spremnik r2
        addcc   %r1, %r2, %r3   ! r3 ← r1 + r2, postavi uvjetni kod
        st      %r3, [z]        !upiši riječ iz spremnik r3 u memoriju M[z]
        jmpl    %r15, +8, %r0   !vrati se u program koji je pozvao proceduru
        nop                     !trošenje vremena zbog grananja
x:      15                      !rezervira se memorija za varijable
y:      9
z:      0
        .end
```

Ovaj primjer prikazuje i korištenje **nop** naredbe koja je ubačena zbog grananja. S obzirom na obradu koju program izvodi moglo se na njeno mjesto i premjestiti **st** naredba i tada **nop** ne bi bio potreban. Kod ovog primjera pretpostavka je da moguće generirati neposredne adrese za x, y i z. U strojnom jeziku postoje funkcije %hi i %lo koje proširuju 22 bita većeg značenja i 10 bita manjeg značenja argumenta kako bi se moglo pristupiti memorijskoj adresi koja nije neposredno adresabilna. Primjer čitanja sadržaja s adrese koja nije neposredno adresabilna w i upis rezultata u spremnik r2 realizira se sljedećim naredbama:

```
sethi   %hi(w), %r1         !u r1⟨31..9⟩ ← adr_w⟨21..0⟩#10@0
ld      [%r1+%lo(w)], %r2   !u r2 ← w, adr_w = r1 + 22@0#adr_w ⟨9..0⟩
```

Sljedeći programski odsječak ilustrira povezivanje glavnog programa s potprogramom.

```
        .begin
        .org
prog:   ld      [x], %o0        !upiši prvi parametar u izlazni spremnik o0
        ld      [y], %o1        !upiši drugi parametar u izlazni spremnik o1
        call    add3            !pozovi proceduru za zbrajanje tri broja
        ld      -17, %o2        !upiši treći parametar u izlazni spremnik o2
        st      %o0, [z]        !spremi rezultat u memoriju
        ...
x:      15                      !rezervira se memorija za varijable
```

```
y:      9
z:      0
add3:   save    %sp, -(16*4), %sp   !otvori novi prozor i pripremi kazalo stoga
        add     %i0, %i1, %l0       !zbroji prva dva ulazna parametra
        add     %i2, %l0, %l0       !pribroji međurezultatu i treći parametar
        ret                         !povratak, skraćeno za jmp  %i7+8
        restore %l0, 0, %o0         !rezultat prebačen u o0 programa
        .end
```

Potprogram zbraja tri vrijednosti x, y i konstantu –17. %sp je drugi naziv za spremnik %o6. Kazalo stoga se pomiče za 16 lokacija kako bi se napravilo mjesto za pohranu sadržaja spremnika ukoliko je to potrebno. Također za primijetiti je da su operacije koje slijede po logici prije ubačene iza naredbi za poziv procedure i povratak iz procedure. Poziv i povratak iz procedure nisu postavili zastavicu poništavanja što znači da se naredba koja slijedi obavezno izvodi.

3.4.5. Prekidi i zamke kod SPARC procesora

Posebna stanja kod SPARC procesora nazivaju se zamke (*traps*) ako se generiraju interno, odnosno prekidi (*interrupts*) ako se generiraju iz vani. Postoji zastavica dozvole zamki ili prekida u spremniku stanja procesora, kao i četiri bita za određivanje prioriteta prekida. Prekid će biti prihvaćen samo ako mu je prioritet veći od tekućeg. Zamke koriste prozore spremnika na način da ona neće biti prihvaćena ukoliko ne postoji slobodan prozor.

Sljedeći su koraci prilikom pojave zamke:

1. Prebacuje se na sljedeći prozor spremnika, kao naredba **save**.
2. Sprema se PC, nPC i PSR u lokalne spremnike novog prozora.
3. Zabranjuju se nove zamke.
4. Prebacuje se izvođenje na proceduru za obradu zamke (*trap handler routine*).

Adresa procedure za obradu zamke određena je bazom upisanom u bazni spremnik zamke, TBR, i osam bitovim tipom zamke generiran interno ili poslan od uređaja koji je postavi zahtjev za prekidom. Adresa se dobiva na način da tip zamke definira pomak u tablici vektora prekida. početna adresa tablice definirana je baznim spremnikom zamki. Procedura za obradu zamki ne smije mijenjati sadržaje ulaznih i izlaznih spremnika novog prozora iz razloga što ulazni spremnici novog prozora poklapaju se sa izlaznim spremnicima prekinute procedure, dok izlazni spremnici preklapaju se sa sljedećim prozorom. Povrat iz procedure za obradu zamki izvodi prethodna četiri koraka suprotnim redoslijedom kako bi se vratilo izvođenje na prekinuti program.

3.4.6. Cjevovodi kod procesora SPARC MB86900

Različiti procesori temeljni na SPARC arhitekturi koriste i različite razine cjevovoda. Fujisu MB86900 je bio prvi procesor temeljen na SPARC arhitekturi izrađen u CMOS tehnologiji. Maksimalna frekvencija takta je 16.6 MHz. Ovaj procesor ima cjevovod sa četiri razine, jedna za svaki korak u ciklusu izvođenja naredbe: dohvati, dekodiraj, izvedi i upiši.

Sljedeća slika prikazuje cjevovod za četiri naredbe. U prvom taktu dohvata se prva naredba iz memorije. U drugom taktu prva naredba se dokodira, dohvaćaju se operandi i prosljeđuju ivedbenoj jedinici, dok se paralelno dohvata druga naredba iz memorije. U

trećem taktu ALU izvodi prvu naredbu i rezultat upisuje u privremeni spremnik, druga naredba se dekodira, a treća dohvaća iz memorije. U četvrtom taktu rezultat prve naredbe se upisuje u spremnike, dok je druga naredba u izvođenju, treća u fazi dekodiranja, a četvrta u fazi dohvaćanja iz memorije.

Rješenje s odgođenim grananjem omogućava da se grananje neometano propagira kroz cjevovod ukoliko je moguće odrediti adresu grananja na vrijeme tako da naredba na adresi grananja bude sljedeća koja ulazi u cjevovod. Ukoliko se ispostavi da je ispunjen uvjet za grananje nastavlja se s interpretacijom i izvođenjem naredba, a u suprotnom ona se odbacuje i stavlja se u cjevovodu naredba koja slijedi naredbu za grananje.

3.4.7. Unapređenja SPARC arhitekture

SPARC International je nedavno objavio specifikaciju za SPARC V9, 64 bitovnu implementaciju SPARC arhitekture. Definirano je kako postojeće naredbe nad 32 bitovnim brojevima koriste nove 64 bitovne spremnika, te su uvedene nove naredbe. Ispitivanje naredbi i njihovih formata pokazuje kako su mnoge naredbe ostale neimplementirane. Tako su ti formati iskorišteni za nove naredbe koje koriste prednosti 64 bitovne arhitekture, 64 bitovno množenje i dijeljenje, upis, čitanje, operacije nad 128 bitovnim brojevima prikazanim s pomičnim zarezom, programsko određivanje predviđanja grananja, grananje u ovisnosti o vrijednosti spremnika (eliminira se prethodna usporedba), uvjetno prebacivanje podatka kojim se eliminira sasvim potreba za nekim naredbama za grananja.

SPARC V9 uveo je posebne spremnike za obradu zamki i prekida i time je odvojio obradu zamki i prekida od izvođenja programa. Također povećan je broj spremnika opće namjene dostupnih korisniku. Mehanizam pokazivača tekućeg prozora kao i otkrivanje i servisiranja slučaja kada su svi prozori popunjeni, odnosno kada su svi prazni, značajno je poboljšan.

Rezultat zajedničkog projekta između Sun Microsystems i Texas Instrumentsa je UltraSPARC koji je implementacija SPARC V9. Ovo je superskalarni procesor s dvije jedinice za obradu cjelobrojnih veličina, pet jedinica za obradu brojeva prikazanih s pomičnim zarezom, posebna jedinica za obradu grananja i jedinica za pristup memoriji. Također postoji i pet jedinica brzu obradu slike i grafiku.

SPARC kao tipičan RISC procesor

- RISC ima znatno manje naredbi nego CISC. Naredbe su jednostavnije, a aritmetika koja zahtjeva više koraka prebacuje se na obradu u posebnu sklopovsku jedinicu.
- SPARC pristupa memoriji preko naredbi učitaj/piši. Aritmetičke operacije obavljaju se samo nad sadržajem spremnika.

- Postupak dekodiranja naredbi i operanada je jednostavan jer postoji samo nekoliko formata naredbi i ograničeni broj adresnih modova.
- Odgođeno grananje je tipično za svaki RISC i proizlazi iz postupka kako se obrađuju naredbe u cjevovodu.
- SPARC procesor ne koristi i odgođen dohvat iz memorije, što ima većina RISC procesora, ali koristi prozore spremnika što nema većina RISC procesora.

Zaključak

- Cijena i performanse računala su pokretačka snaga računalne industrije. Performanse računala mogu se mjeriti na više načina počevši od najjednostavnijeg mjerenja broja naredbi u sekundi koje računalo izvede, do sofisticiranijih procjena koje nastoje simulirati stvarno radno opterećenje računala, kao npr. Wheatstone i Dhrystone programi za ispitivanje. Nedavno SPEC koristi stvarne programe kako bi se što realnije procijenile performanse računala.
- Ekonomičnost kodiranja naredbi i adresnih modova karakteristike su CISC procesora. Rezultat ovakvog pristupa su naredbe koje se međusobno razlikuju u dužini i vremenu potrebnom za njihovo izvođenje, adresnim modovima i broju pristupa memoriji. Tadašnja cijena memorije razlogom je ovakvom rješenju. Međutim kada se sklopovskim rješenjima, cjevovodima i superskalarnom obradom pokušala povećati brzina obrade glavna prepreka bile su složene naredbe i adresni modovi.
- Temelj RISC procesora su jednostavnije naredbe, jednake dužine te pristup memoriji pomoću naredbi piši/čitaj. Projektanti su ustanovili da se jednostavne naredbe, jednake dužine bolje uklapaju u sklopovska rješenja cjevovoda i superskalarne obrade. Ujedno ovakav pristup dozvoljava veću frekvenciju takta. Istina je da ovakav pristup zahtijeva veću memoriju te učestaliji pristup memoriji, ali brža memorija kao i tehnike preddohvata naredbi praktički su donijeli prevagu ovom konceptu koji je dodatno unaprijeđen cjevovodom i superskalarnom obradom.

Zadaci

1. Program se sastoji od sljedećih naredbi:
   - 60% naredbe čitaj/piši s vremenom izvođenja naredbe od 1.2 μs
   - 10% ALU naredbe s vremenom izvođenja naredbe od 0.8 μs
   - 30% naredbi za grananja s vremenom izvođenja naredbe od 1 μs

   a) Ako je trajanje takta 0.2 μs izračunajte prosječno izvođenje naredbe, CPI.
   b) Koliki je prosječan broj naredbi u jedinici vremena, MIPS, ovog programa.
2. Koja je vrijednost u odredištu nakon sljedećih naredbi MC68000? Koji se adresni modovi u njima koriste?

   A0 = 2000H, M[2000H] = AAAAH, M[1FFFH] = 5555H

   a) MOVE.W #'BE', D0
   b) MOVE.L #FFE07DCAH, (A0)
   c) MOVE.W A0, D0
   d) MOVE.W -(A0), D0

3. Koja vrijednost će biti u spremnicima A0, D0, D1, i D2 procesora MC68000 nakon sljedećeg programskog odsječka? Početne vrijednosti su:

   a) D0 = 0, D1 = 2H, D2 = 10H

```
ORG   $2000
      ADD     D1, D0
      DBLE    D2, $FFFA
```

   b) D0 = 14H, D1 = -2, D2 = 10H

```
ORG   $2000
      ADD     D1, D0
      DBLE    D2, $FFFA
```

4. Napišite RTN opis za MC68000 DBcc naredbu.

5. Generirajte heksadecimalni strojni kod za sljedeće naredbe MC68000.

   a) MOVE.L (A2), D4
   b) MOVE.L 2AH(A3, D6), D1
   c) ADD.L A5, D6
   d) ROL #3, D4
   e) BGT -24

6. Objasnite funkciju sljedećeg programskog odsječka i kodirajte ga u heksadecimalnom obliku.

```
PORT    EQU     $280
ZNAK    EQU     55H
        ORG     $1000
BUFFER  DS.B    100
        MOVEA.L #BUFFER, A0
UPIS    MOVE.B  PORT, D0
        CMPI.B  #ZNAK, D0
        BEQ     KRAJ
        MOVE.B  D0, (A0)+
        BRA     UPIS
KRAJ    RTS
```

7. Objasnite razlog zbog kojeg nije dozvoljeno prekidati rad procesora za vrijeme procesa interpretacije i izvođenja naredbe.

8. Generirajte heksadecimalni strojni kod za sljedeće naredbe SPARC procesora.

   a) addcc %r2, %r3, %o7
   b) sethi 3FF250H, %g1
   c) ldub [%i2 + 7FFH], %r23

9. Napišite RTN opis naredbi za posmak SPARC procesora sll, srl i sra.

10. Sintetizirajte naredbe za rotiranje sadržaja spremnika ROL, ROR, ROLC od ostalih naredbi SPARC procesora. Pretpostavite da je broj n koji određuje broj mjesta za rotiranje implicitno zadan preko .equ direktive.

11. Napišite i usporedite za SPARC i MC68000 procesore programe koji upisuje brojeve 0, 2, 4, 6, 8, 10, 12, 14, 16, 18 u memoriju počevši od adrese 5000H.

12. Napišite SPARC naredbe ili niz naredbe koje simuliraju svih četrnaest adresnih modova MC68000. Neka naredbe upisuje sadržaj jednog spremnika u memoriju.