1. PROGRAMSKI MODEL

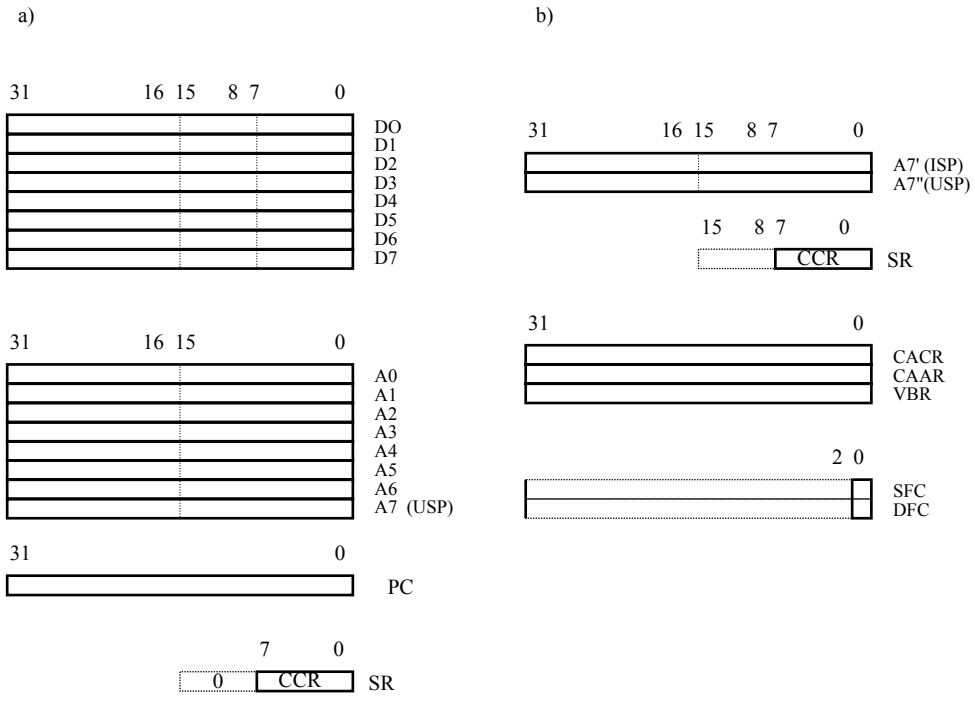
MC68020 je 32-bitni mikroprocesor. Magistrala podataka je 32-bitna. Adresna magistrala je 32-bitna i omogućava adresiranje do 4GB memorije. U samom čipu je ugrađeno 256 bajtova keš memorije za instrukcije i interfejs za spregu sa FP koprocesorom MC68881. MC68020 radi u dva načina rada: korisničkom načinu rada i supervizorskom načinu rada. Nivo privilegije supervizorskog načina rada je viši.

U oba načina rada MC68020 ima različite programske modele. Korisnički programski model čine četiri dela, dok supervizorski programski model proširuje programski za nekoliko dodatnih delova.

1.1. Korisnički programski model

- **Registri za podatke i adrese**

Na slici 1.1a) prikazani su registri opšte namene koji se mogu podeliti na dva skupa: osam registara za podatke (D0-D7) i osam adresnih registara (A0-A7). Svi ovi registri (D0-D7 i A0-A7) su 32-bitni.



Sl. 1.1. a) Registri opšte namene; b) specijalni registri.

Registri za podatke se koriste za manipulaciju podacima, a mogu se adresirati kao operandi tipa bajt, reč ili duga reč (8, 16, ili 32 bita, respektivno). Operacije koje manipulišu bajtovima ili rečima koriste samo LS bajt ili reč registra (na slici 1.1 je to prikazano isprekidanim linijama). Registri D0-D7 mogu se koristiti za adresiranje samo kao indeksni registri.

Adresni registri koriste se za čuvanje adresa operanada; operacije nad njima nemaju uticaj na stanje statusnog registra, što je u suprotnosti sa operacijama nad registrima za podatke. Bajtovske operacije od strane adresnih registara se ne podržavaju, a reči koje se kopiraju u adresne registre znakovno se proširuju na 32 bita (opet je u suprotnosti sa radom registara za podatke).

- **Pokazivač magacina**

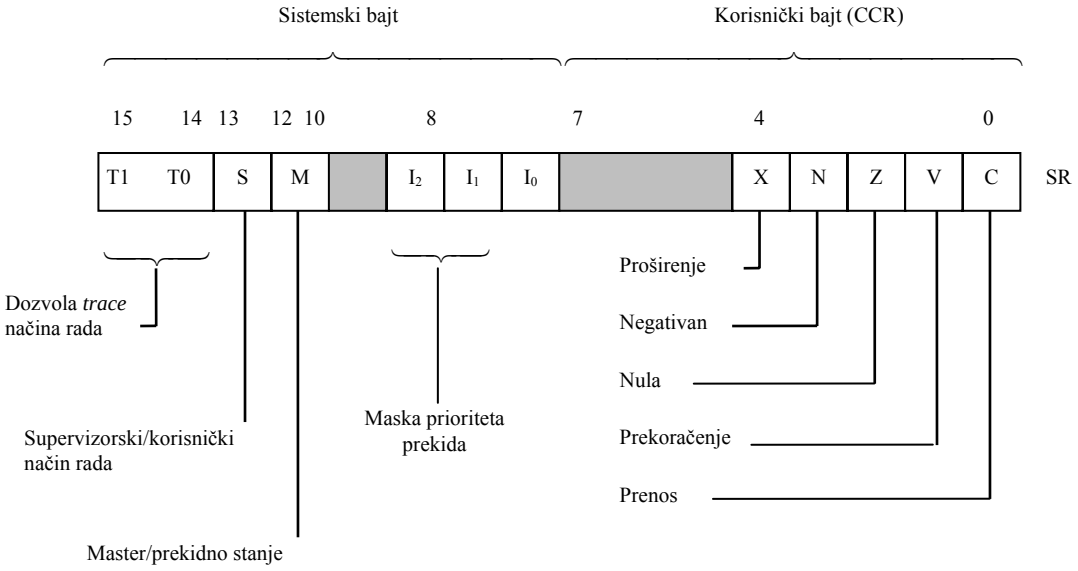
Registar A7 je implicitno određen da bude sistemski pokazivač magacina (SP) za pozive potprograma i prekida. SP se koristi samo u korisničkom načinu rada.

- **Programski brojač**

MC68020 ima 32 bitni programski brojač (PC) koji se koristi za pristup instrukciji koja se izvršava kao naredna.

- **CCR (Condition Code Register)**

Kod MC68020 postoji 16-bitni statusni registar (SR), koga čine sistemski bajt i korisnički bajt (slika 1.2). U korisničkom načinu rada, pristupa se samo korisničkom bajtu, dok se sistemski bajt uvek čita kao nula.



Sl. 1.2. Statusni registar mikroprocesora MC68020.

Korisnički bajt, LS bajt statusnog registra, sadrži markere koji ukazuju na uslov, pa se zbog toga ovaj registar zove CCR. CCR sadrži četiri uslovna markera; Zero (Z), Negative (N), Overflow (V) i Carry (C), koji odražavaju status procesora nakon operacije. Pored toga CCR sadrži bajt proširenja (X), koji se koristi kao operand (prenos) kod operacija sa proširenom tačnošću. On se postavlja na isti način kao i standardni marker procesora (C), ali se njegovo stanje ne menja kod operacija kopiranja podataka (data move), dok se (C) briše kod operacija MOVE. Ostala tri bita CCR-a su rezervisana i ne mogu se koristiti. Uslovni markeri (kodni bitovi) postavljaju se u saglasnosti sa sledećim pravilima:

- ◊ **N (Negative):** Postavlja se ako je MS bit rezultata postavljen, inače je obrisan (ovaj bit signalizira negativan rezultat, jer MC68020 koristi aritmetiku dvoičnog komplementa).
- ◊ **Z (Zero):** Postavlja se ako je rezultat nula, inače je obrisan.
- ◊ **V (Overflow):** Postavlja se ako se javi aritmetički premašaj; to znači da prenos u MS bit se razlikuje od prenosa iz MS bita. Ovo ukazuje da se rezultat ne može predstaviti u odredišnom operandu. U ostalim slučajevima je obrisan.
- ◊ **C (Carry):** Postavlja se ako se generiše prenos sa MS bita operanda u toku operacije sabiranja. Postavlja se takođe ako se generiše pozajmljivanje kod operacije oduzimanja. Inače je obrisan.
- ◊ **X (Extended):** Transparentan je za operacije premeštanja podataka (data movement). Kada se menja, ovaj bit se postavlja na isti način kao i bit (C).

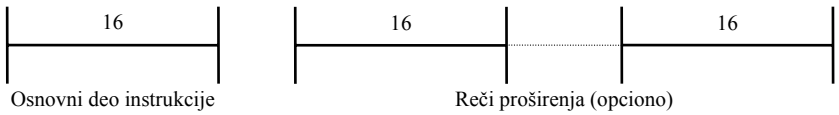
1.2. Supervizorski programski model

Kao što je prikazano na slici 1.1b) u supervizorskom načinu rada može se pristupiti dodatnim registrima. To su registri: CACR (Cashe Control Register) i CAAR (Cache Address Register) koji upravljaju radom "on-chip" keša instrukcija, VBR (Vector Base Register) ukazuje na početak vektor tabele izuzetaka, SFC (Source Function Code) registar i DFC (Destination Function Code) registar koji obezbeđuju identifikatore za izvorišni i odredišni prostor kod MOVE instrukcija.

- **Pokazivač magacina** - pored pokazivača magacina u korisničkom načinu rada A7 (USP - user stack pointer), kod supervizorskog programskog modela koriste se dva dodatna pokazivača magacina: A7' (ISP - interrupt stack pointer) i A7" (MSP - master stack pointer). Kom će se od ovih registara pristupati kao pokazivaču magacina određeno je od strane bita M statusnog registra (slika 1.2). Opšti naziv SSP (supervisor stack pointer) odnosi se na MSP ili ISP u zavisnosti od stanja ovog bita.
- **Statusni registar** - sistemski bajt, MS bajt SR-a (slika 1.2) može se menjati isključivo u supervizrskom načinu rada. Ovaj registar sadrži bitove koji se koriste za upravljanje radom procesora. Bitovi testiranja (T0,T1) koriste se da dozvole rad "on-line debugging". Koriste se tri načina trasiranja: nema trasiranja, trasiranje kod izvršenja instrukcije i trasiranje kod promene toka programa. "Interrupt mask" bitovi (I2,I1,I0) koriste se da ukažu na tekući nivo prioriteta prekida. Samo prekidi višeg nivoa u odnosu na tekući se prihvataju, a svi ostali se maskiraju (ne dozvoljavaju). Supervizorski bit (S) koristi se da ukaže na način rada (S=1, supervizorski način rada; S=0, korisnički način rada). Korisnički način rada je važan kod višekorisničkog (multiuser) okruženja (na primer za upravljanje memorijom i u zaštitne svrhe). Kada je S=0, USP je izabran kao SP, kada je S=1, M bit (Master bit) se koristi za izbor SP-a (M=1: MSP; M=0: ISP).

1.3. Formati instrukcija

MC68020 podržava instrukcije čija je dužina dva bajta (reč) ili veći broj bajtova; tj. adresna rezolucija instrukcija je reč. Osnovni deo instrukcije zahteva jednu reč, dok, u zavinosti od opkoda ili specifikacije operanda, ukupno može biti potrebno i do 10 reči proširenja (slika 1.3). Arhitektura MC68020 je tako projektovana da se najčešće za specifikaciju operanada koristi podatak tipa reč.



Sl. 1.3. Struktura instrukcije mikroprocesora MC68020.

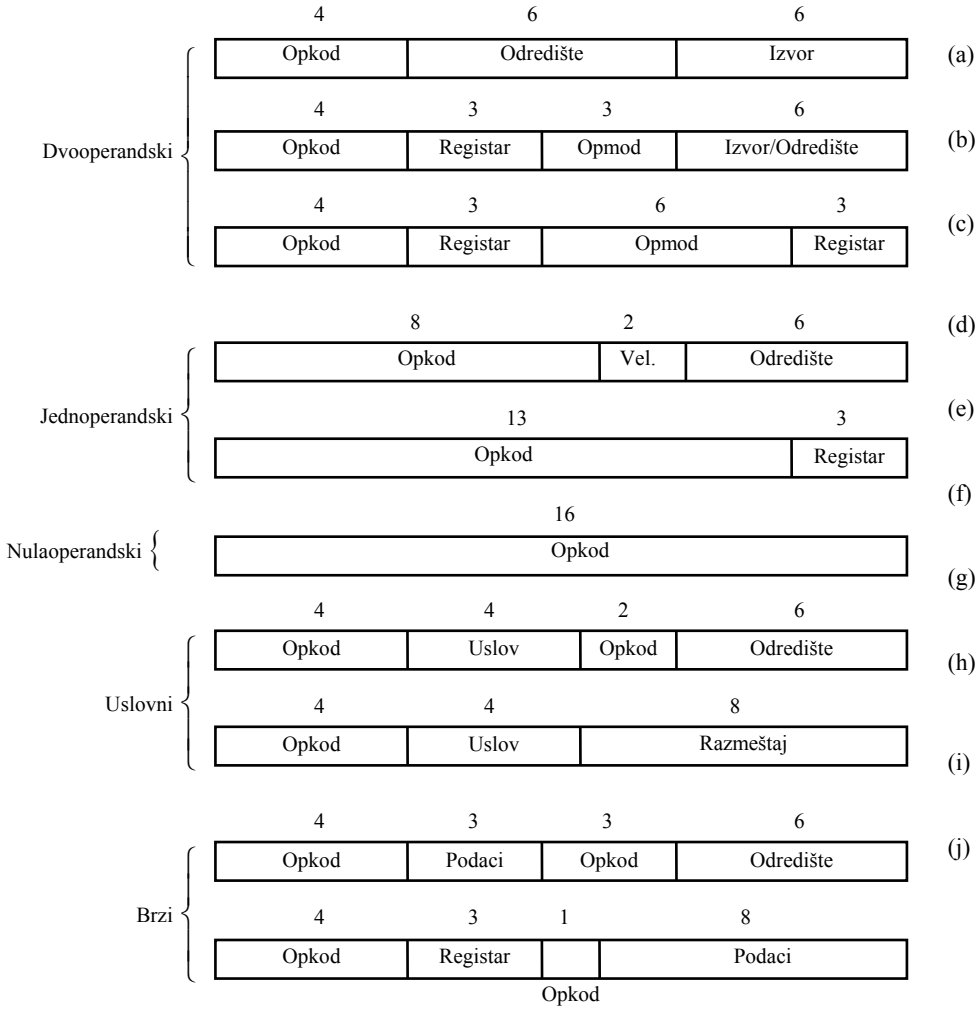
U daljem radu koristićemo sledeću notaciju:

- Src - izvorni operand
- Dst - odredišni operand
- Dn - registar za podatke
- An - adresni registar
- Xn - indeksni registar (registar podataka ili adresa)
- Rn - bilo koji registar za podatke ili adrese
- Data - neposredni podatak koji se sadrži u instrukcionom nizu
- Cnt - podatak koji koriste instrukcije rotiranja i pomeranja
- Bnum - specifikacija broja bita
- Disp - 8, 16 ili 32-bitni razmeštaj
- d_{16} - 16-bitni razmeštaj
- d_8 - 8-bitni razmeštaj
- List - lista izabranih registara
- N - obim operanda u bajtovima (1, 2, ili 4)
- ( ) - sadržaj od
- ea - efektivna adresa (EA)
- M - memorija
- # - neposredna vrednost
- * - iza zvezdice sledi komentar (često se koristi i ;)
- < > - bilo koji važeći simbol (simbol mora biti specificiran)
- 0x - vrednost izražena u heksadecimalnoj vrednosti (koristi se i $)
- ob - vrednost izražena u binarnoj notaciji (dvoični komplement)
- blanko - vrednost izvražena u decimalnoj notaciji
- [ ] - "subscripting", koristi se da ukaže i na dužinu, a može i da bude oznaka za opciono.

Primer 1.1:
(a) 61, 0x3D i 0B00111101 predstavljaju istu vrednost
(b) M[3]= memorijska lokacija 3
(c) M[(A0)]= memorijska lokacija specificirana sa (A0)
(d) D3[3]= bit 3 registra za podatke D3
(e) D3[10..7]= bitovi 10 do 7 registra za podatke D3
(f) M[3][2]= bit 2 od M[3]

1.3.1. Osnovni formati instrukcija

Skup instrukcija MC68020 sadrži 101 instrukciju koje koriste veći broj različitih formata instrukcija. Ovi formati mogu se kategorizovati u zavisnosti od broja operanada za svaku instrukciju (slika 1.4).



Sl. 1.4. Formati instrukcija kod MC6820.

Dvooperandski format instrukcije

Najveći broj dvooperandskih instrukcija mogu se svrstati u jedan od formata sa slike 1.4. Operandi mogu biti opšti operandi (označeni sa odredišni (Dst) ili izvorni (Src)). Opšti operandi dozvoljavaju specifikaciju memorijske lokacije ili registra. Adresni način rada za Src/Dst operand specificiran je preko specifikatora operanda. Specifikator operanda je sastavljen od dva trobitna polja, "mode" polja i polja za specifikaciju registra. "Mode" polje specificira adresni način rada, koji može biti, na primer, indeksno, apsolutno ili neposredno adresiranje, pa se u tom slučaju instrukcija proširuje rečima proširenja. Registarsko polje specificira broj korišćenog adresnog registra za podatke. Dvooperandski formati instrukcija mogu se klasifikovati na sledeći način:

1. Dst:=(Src) (dva opšta operanda)

Src/Dst dvooperandski format sa slici 1.4 ima opkod polje od četiri bita, koje je nedovoljno za specifikaciju velokog broja instrukcija čiji se rad može podržavati. Zbog toga ovaj format postoji samo kod MOVE instrukcija, koje se veoma često javljaju. Ostale dvooperandske instrukcije se podržavaju od strane ograničenih dvooperandskih formata sa sl. 1.4b i 1.4c.

2. Reg:=(Reg)operacija(Src) (jedan opšti operand, koji je izvorni)
   Dst:=(Dst)operacija(Reg) (jedan opšti operand, koji je i izvorni i odredišni)

Na slici 1.4b), trobitno "opmod" polje se koristi kao proširenje opkod polju. Ono specificira tip podatka (bajt, reč ili duga reč) kao i smer operacije - to je, koji se operand (registar ili opšti operand) koristi za čuvanje rezultata. U grupu instrukcija koje koriste ovaj oblik specifikacije operanda spadaju instrukcije ADD, SUB, AND i MUL.

3. Reg:=(Reg)operacija(Reg) (dvoregistarski operandi)

Kako gornje instrukcije ne nude dovoljno opkod prostora za dvooperandske instrukcije, često se ponekad koristi format sa slici 1.4c. Ovaj format koristi registre kako za izvorne, tako i za odredišne operande. Šestobitnim poljem opmod ne specificira se samo tip podatka operanda, nego i da li se koristi An, Dn ili oba registra. Instrukcije iz ove klase su EXG (promeni sadržaj dva registra) i SUBX (oduzmi sa proširenjem).

Jednooperandski format instrukcija

Za jednooperandske instrukcije, kao što je prikazano na slici 1.4d i 1.4e postoje dva formata. Prvi format se koristi kod instrukcija koje zahtevaju specifikaciju Dst operanda, kao što su CLR (Clear) i NEG (Negate). Drugi oblik se koristi kod jednooperandskih instrukcija koje koriste samo jedan Dn registar, kao što su EXT (Proširi znak sadržaju registra) i SWAP (promeni mesta MS i LS polovine registra).

Nultooperandski format instrukcije

Na slici 1.4f prikazan je format ovih instrukcija (nemaju operand). Primeri ovih instrukcija su NOP, RESET, RTS (return from subroutine).

Formati uslovnih instrukcija

Na slici 1.4g i 1.4h prikazani su formati nekih uslovnih instrukcija. Format instrukcije Scc (Set Conditionally) prikazan na slici 1.4g koristi Dst operand, a format instrukcije Bcc (Branch Conditionally) prikazan je na slici 1.4h koristi 8-bitno polje razmeštaja. Ako se kodiranje uslova posmatra kao deo opkoda, tada ove instrukcije koriste 8- ili 10-bitni opkod, a Scc instrukcija, na primer, može se klasifikovati kao jednooperandska instrukcija (sa Dst operandom).

"Quick" formati instrukcija

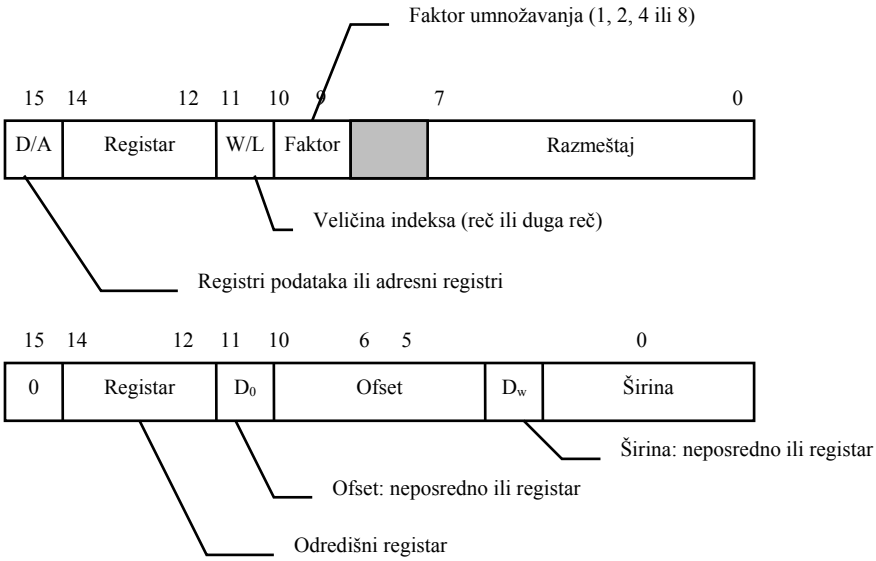
Specijalni formati instrukcija, prikazani na slici 1.4i i 1.4j, koriste se za instrukcije ADDQ, SUBQ i MOVEQ. Ove instrukcije koriste malu konstantu (neposredna vrdnost) koja je kodirana unutar osnovnog dela instrukcije, tako da reči proširenja nisu potrebne. Instrukcije ADDQ i SUBQ (slika 1.4i) sadrže 3-bitnu konstantu, koja se nalazi u opsegu od 1 do 8 (8 je kodirano kao 0; sabiranje nule nije korisno). Instrukcija MOVEQ koristi format sa slike 1.4j, a namenjena je za premeštanje (kopiranje) 8-bitnih konstanti u Dn registru. Podatak se znakovno proširuje na 32-bitni operand a svi bitovi (32) se prenose u Dn registar. Sa 8-bitnom konstantom omogućuje se specifikacija između -128 i +127.

Prošireni formati

Kao što je već bilo naglašeno, dužina osnovnog dela instrukcije je uvek 16 bita, sa maksimalno 6 bitova po specifikatoru operanda. Kako 6 bitova nije uvek dovoljno da se egzaktno specificira željeni operand ili adresa operanda, ponekad kada je neophodno koriste se reči proširenja. U zavisnosti od informacije u reči proširenja, moguće je razlikovati sledeće formate reči proširenja:

1. Razmeštaj kod MC68020 je 16- ili 32-bitna celobrojna vrednost u dvoičnom komplementu, koja se smešta u jednu ili dve reči proširenja, i znakovno se proširuje i sabira sa PC-om da bi se formirala odredišna adresa. Reči proširenja su opcione kod instrukcije grananja, gde je samo jedna reč prisutna ako je 8-bitni razmeštaj u okviru osnovnog dela instrukcije 0, ili su prisutne dve reči ako je 8-bitni razmeštaj jednak 0xFF.
2. Neposredni podatak se može smatrati konstantom, ako je specificiran unutar dela instrukcije. On može biti dužine 8, 16 ili 32 bita, ali, kako kod MC68020 postoje samo instrukcije koje su umnožak od 16 bitova, 8-bitne konstante se čuvaju u LS bajtu reči, dok MS bajt sadrži nule.

3. Ako se koristi apsolutni adresni način rada, iza instrukcije sledi jedna ili dve reči proširenja, koje sadrže apsolutnu memorijsku adresu.
4. Kod indeksnog adresnog načina rada, MC68020 izračunava (svoju) adresu operanda sabiranjem sadržaja baznog (adresnog) registra sa sadržajem indeksnog registra i 8-bitnim razmeštajem. Da bi se specificirale sve komponente, potrebna je jedna reč proširenja (slika 1.5a). Kod memorijsko indirektnih adresnih načina rada neophodan je veći broj reči proširenja.



Sl. 1.5. Format reči proširenja za indeksni adresni način rada.

Specifikator operanda sadrži broj baznog registra, a taj registar je uvek adresni registar. Reč proširenja ukazuje kada će se registar (adresni ili za podatke) koristiti kao: indeks registar, broj registra, veličina indeksne vrednosti (znakovno proširena LS reč) i 8-bitni razmeštaj (znakovno proširen).

5. Instrukcijom MOVEM puni se ili se memoriše lista od 16 registara iz ili ka specificirane memorijske adrese. Da bi se specificiralo koji se registri prenose, koristi se maskirajuća lista koja ima oblik reči proširenja. Ako je bit u masci postavljen, prenosi se odgovarajući registar.
6. Instrukcije koje manipulišu bit poljima koriste reč proširenja (slika 1.5b) da bi se specificirao opcioni odredišni registar, polje Ofset (bilo da je neposredno ili se nalazi u Dn registru) i obim polja (bilo da je neposredno ili se sadrži u Dn registru).

Prisustvo reči proširenja može se specificirati opkodom (za slučaj instrukcije MOVEM), specifikatorom operanda (adresni način rada koji koristi reč proširenja), ili vrednošću operanda (kod instrukcija grananja). Analizirajući specifikaciju reči proširenja, evidentno je da je prisustvo registarske liste za maskiranje određeno od strane opkoda, a da je prisustvo reči proširenja za apsolutne adrese ili indekse određeno od strane specifikatora operanda. Prisustvo reči proširenja koje sadrže neposredne podatke, ili razmeštaje, može se odrediti bilo od strane opkoda bilo od strane specifikatora operanda, dok vrednost operanda može ponekad da odredi potrebu reči proširenja razmeštaja, kao što je slučaj sa razmeštajem kod grananja.

1.4. Tipovi podataka

MC68020 podržava rad sa sledećim tipovima podataka:

1. **Logički (Booleans)** - da bi podržao rad Boolean promenljivih, koje se koriste od strane HLL, u skupu instrukcija ovog mikroprocesora postoji instrukcija Scc koja može koristiti iste uslove kao i instrukcija uslovnog grananja, i koja dodeljuje vrednost TRUE (sve jedinice) ili FALSE (sve nule) odredišnom operandu, u zavisnosti od specificiranog uslova.
2. **Celobrojni** - dele se na celobrojne vrednosti fiksne i promenljive dužine. Celobrojne vrednosti fiksne dužine imaju dužinu 8, 16 ili 32 bita (bajt (B), reč (W) i duga reč (L), respektivno), i mogu se posmatrati u prezentaciji dvoičnog komplementa. Šta više, instrukcije MUL i DIV koriste 64-bitne ("gurad word" (Q)) celobrojne vrednosti, koje se smeštaju u dva Dn registra, sa ciljem da čuvaju proizvod dvostruke dužine (proizvod dve

duge reči celobrojnog tipa). Ako se sa 32 bita ne nudi dovoljna preciznost, programer može koristiti celobrojne vrednosti sa proširenom preciznošću (promenljive dužine), koje se podržavaju pomoću specijalnog bita u CCR-u (extended flag X) i specijalnih instrukcija kao što su ADDX (Add with Extended), NEGX (Negate with Extended) i MULU (Multiply Unsigned). MC68020 ne podržava rad sa znacima na poseban način, tj. oni se obrađuju kao 8-bitne celobrojne vrednosti.

3. **Decimalni** - MC68020 podržava rad sa BCD brojevima na određeni način. U jednom bajtu ima po dve BCD cifre. Sve BCD instrukcije (ABCD (Add BCD), SBCD (Subtract BCD), i NBCD (Negate BCD)) operišu samo sa jednobajtnim operandima. Sa X bitom (kao što je slučaj kod višestruke preciznosti sa celobrojnim vrednostima), program može koristiti decimalne brojeve promenljive dužine, ali se dužina može kontrolisati softverski. Kod MC68020 postoje instrukcije PACK i UNPACK za konverziju u ili iz nepakovanih BCD brojeva.

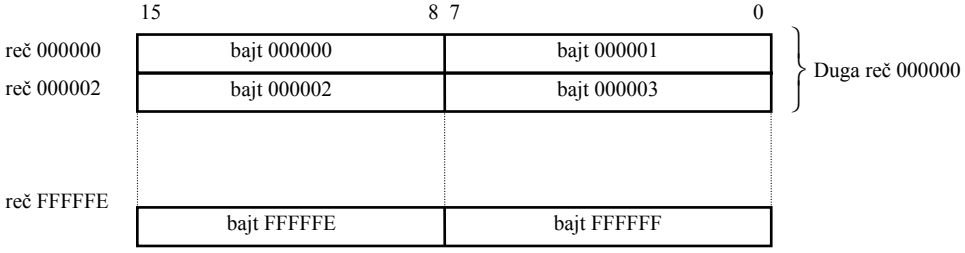
I pored toga što nisu kompletni tipovi podataka, jedinstvenim bitovima se može manipulisati pomoću četiri specijalne instrukcije koje brišu, postavljaju, komplementiraju ili prosto testiraju bit. Ne može se smatrati da jedinstveni bit predstavlja u potpunosti Boolean tip podatka zbog nemogućnosti da podržava operacije kao što su AND i OR.

Rad sa bit poljima se podržava, tamo gde bit polje može biti dužine od 1 do 32 bita i može startovati na bilo kojoj bit lokaciji u memoriji. Bit polje se specificira preko efektivne adrese operanada, bit ofset (pomeraj) u odnosu na baznu adresu i veličinu polja. Aktivnost instrukcija koje podržavaju rad sa bit poljima odnosi se na brisanje bit-polja, postavljanje, komplementiranje ili testiranje (slično kao i jednobitne instrukcije), pored instrukcije za izbacivanje (odstranjivanje), ubacivanje (insertovanje) ili pretraživanje bit polja.

Adrese se mogu smatrati specijalnim tipom celobrojnih vrednosti (neoznačene 32-bitne celobrojne vredmosti). Rad sa adresama se podržava od strane nekoliko instrukcija i to onih koje operišu sa onim adresama koje se smeštaju u registre An, kao što su MOVEA (Move Address) i CMPA (Compare Address).

1.5. Organizacija podataka u memoriji

Organizacija memorije kod MC68020 prikazana je na slici 1.6.



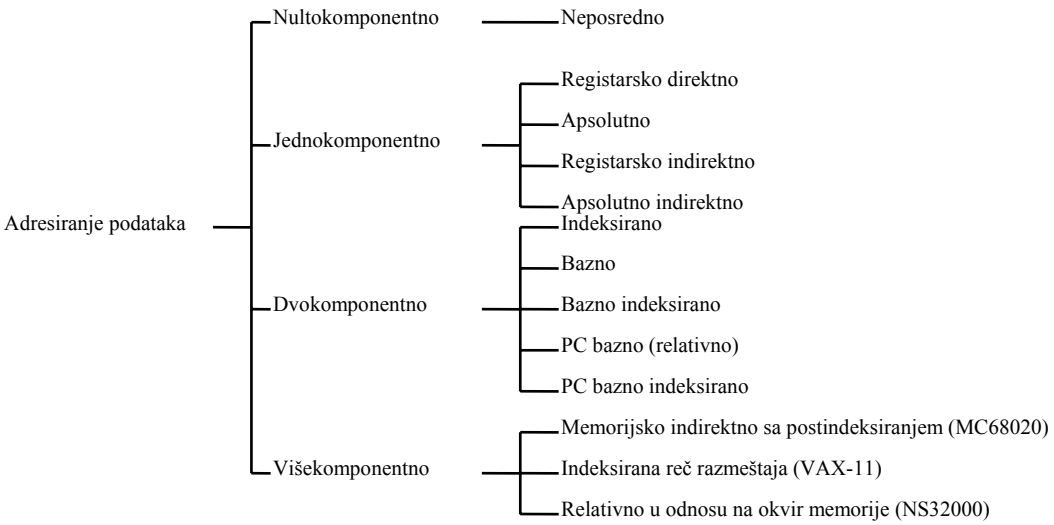
Sl. 1.6. Organizacija memorije kod mikroporcesora MC68020.

Osnovna adresibilna jedinica podatka je bajt: svaki bajt se može adresirati posebno. Reč čine dva bajta a adresa reči je određena MS bajtom. Adresa duge reči takođe odgovara adresi MS bajta. Instrukcijama se pristupa samo kao podacima tipa reč (parna bajt adresa).

Što se podataka tiče adresna rezolucija je bajtovska. Reči i duge reči ne moraju biti poravnane na parnim bajt adresama, ali najefikasniji prenos podataka se vrši ako su podaci poravnani. Problem se javlja kada bajtovske operande treba smestiti ili izbaviti iz magacina koristeći se SP-om. Da bi se održao magacin poravnan, zbog maksimalne efikasnosti, magacin se inkrementira ili dekrementira za dva, a ne za jedan u slučaju kada se u magacin smešta ili izbavlja podatak tipa bajt.

1.6. Adresni načini rada

Opšti specifikator operanda kod MC68020 čini trobitno "mode" polje i trobitno "register" polje, pomoću kojih se, saglasno Tabeli 1.1, omogućava specifikacija 12-različitih adresnih načina rada. Za načine rada (mode) 0 do 6 registarsko polje specificira broj korišćenog registra. Kod "Mode 0" koristi se Dn registar, a načini rada od 1 do 6 uvek koriste An registar. Kod "Mode 7" registarsko polje se koristi za precizniju specifikaciju adresnog načina rada.



Sl. 1.7. Klasifikacija adresnih načina rada.

Tab. 1.1. Adresni načini rada kod MC68020.

Mod	Registar	Naziv	Sintaksa	Izračunavanje efektivne adrese
0	reg#	Direktno registarsko za podatke	Dn	ea = Dn
1	reg#	Direktno adresno registarsko	An	ea = An
2	reg#	Adresno registarsko indirektno	(An)	ea = (An)
3	reg#	Adresno registarsko indirektno sa postinkrementom	(An)+	ea =(An), An = *An) + N
4	reg#	Adresno registarsko indirektno sa predekrementom	-(An)	An = An - N, ea = (An)
5	reg#	Adresno registarsko indirektno sa razmeštajem	(d_{16},An)	ea = d_{16} + (An)
6	reg#	Adresno registarsko indirektno sa indeksiranjem	(d_8,An,Xn)	ea = d_8 + (An) + (Xn)
6	reg#	Memorijsko indirektno sa postindeksiranjem	([bd,An],Xn,od)	ea = (M[(An) + bd]) + (Xn) + od
6	reg#	Memorijsko indirektno sa preindeksiranjem	([bd,An,Xn],od	ea = (M[(An) + (Xn) + bd]) + od
7	0	Apsolutno kratko	xxx.W	ea = (sledeća reč)
7	1	Apsolutno dugo	xxx.L	ea = (sledeće dve reči)
7	2	Relativno u odnosu na programski brojač	(d_{16},PC)	ea = d_{16} + (PC)
7	3	Relativno u odnosu na programski brojač sa indeksiranjem	(d_8.PC,Xn)	ea = d_8 + (PC) + (Xn)
7	3	PC memorijsko indirektno sa postindeksiranjem	([bd,PC],Xn,od)	ea = (M[(PC) + bd]) + (Xn) + od
7	3	PC memorijsko indirektno sa preindeksiranjem	([bd,PC,Xn],od	ea = (M[(PC) + bd + (Xn)]) + od
7	4	Neposredno	#xxx	podatak = (sledeća(e) reč(i))

Napomena: ovde se pod Xn podrazumeva kompletan indeksni operand Xn.<l>*SCALE, gde je Xn proizvoljni registar za podatke ili adresni registar, <l>=BđWđL, a SCALE je faktor umnožavanja (1, 2, 4 ili 8).

Adresni načini rada mikroporocesora MC68020 prikazani su u tabeli 1.1, zajedno sa imenom (definisanim od strane firme Motorola), asemblerskom sintaksom i načinom na koji se izračunava efektivna adresa (ea). Adresni načini rada uobičajeno se klasifikuju shodno broju komponenata, pri čemu komponenta može biti registar, razmeštaj (displacement) ili apsolutna adresa (slika 1.7).

1.7. Jednokomponentni adresni načini rada

Kao i kod najvećeg broja arhitektura i kod MC68020 ovaj način adresiranja se zove neposredno adresiranje. Kod neposrednog adresiranje (mod=7.4 Tab. 1.1) vrednost operanda se sadrži u jednoj ili dve reči proširenja, u zavisnosti od obima operanda. Na primer, kod instrukcije ADD.W #0x7AE2,D2 izvorni operand specificira neposrednu vrednost 0x7AE2, kod instrukcije ADD.L #0x700AB123,D2 izvorni operand specificira neposrednu vrednost 0x700AB123.

Primer 1.2:

Neposredno adresiranje se koristi za specificiranje konstanti obima 8, 16 ili 32 bita. Konstanta je deo instrukcije, i smeštena je zajedno sa instrukcijom u programskoj memoriji. Na primer

```
ADD     #5,D1
```

dodaje vrednost 5 u (D1)[15:0].

Instrukcija

```
MOVE.B  #'A',(A1)
```

kopira u memoriju ASCII vrednost 'A' u bajt adresiran od strane A1.

Instrukcija

```
MOVE.L  #'1234',D1
```

uslovljava da se D1 zameni sa ASCII ekvivalentom od 1234 ili heksadecimalnom vrednošću 31323334.

Instrukcija

```
MOVE.W  #$F0,D1
```

ima efekat (D1)[15:0]JF0 .

Instrukcija kod koje je neposredni operand odredište, kao što je

```
MOVE.B  1000,#1000
```

biće ilegalna i neće se asemblirati.

1.7.1. Adresni način rada koji koristi jedan registar

An i Dn registrima se može direktno pristupati, ali kod indirektnih načina rada može se koristiti samo An.

1.7.2. Direktno adresiranje preko registra za podatke

Kod ovog načina adresiranja (mod=0 Tab.1.1) operand se nalazi u Dn. Ako se operacija obavlja nad operandom tipa bajt ili reč, MS bitovi Dn-a ostaju nepromenjeni.

Primer 1.3:

Instrukcijom

```
ADD.W D0,D1
```

obavlja se sledeća aktivnost: D1[15:0]:=(D1[15:0])+(D0[15:0]); LS reč registra D0 sabira se sa LS rečju registra D1.

Instrukcijom

```
MOVE.W  D1,D2
```

kopira se (D1)[15:0] u (D2)[15:0].

Instrukcijom

```
CLR.W   D2
```

brišu se LS 16 bitova registra D2.

1.7.3. Direktno adresiranje preko adresnog registra

Kod ovog načina adresiranja (mod=1 Tab. 1.1) operand se nalazi u adresnom registru. Ako je odredište LS reč An registra, menja se celokupni sadržaj An (svih 32 bita), jer se vrši znakovno proširenje do 32 bita (bit 15 se kopira na bit pozicijama od 16 do 31).

Primer 1.4:

Instrukcijom

```
MOVEA.L A1,A2
```

vrši se 32-bitni prenos iz A1 u A2.

Instrukcijom

```
MOVE.W  A1,D1
```

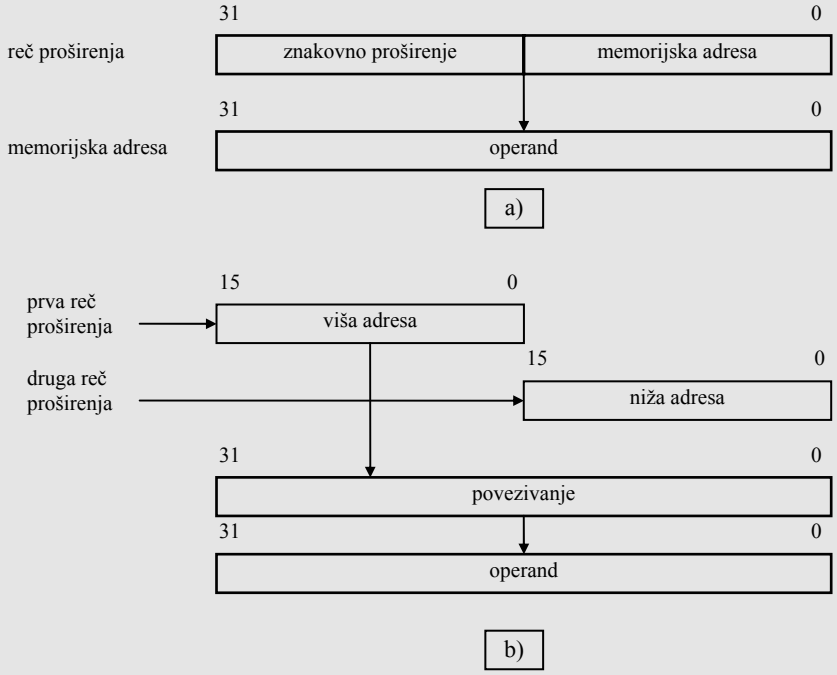
vrši se prenos 16-bitne adrese iz A1 u D1.

1.7.4. Apsolutno adresiranje

Kod apsolutnog adresiranja (mode=7.0 i 7.1 iz Tab. 1.1) memorijska adresa se sadrži u jednoj ili dve reči proširenja koje slede posle instrukcije. Kod MC68020 postoje dva apsolutna adresna načina rada: kratko i dugo. Kod "apsolutnog kratkog", posle instrukcije sledi jedna 16-bitna reč proširenja koja se znakovno proširuje da bi se formirala apsolutna memorijska adresa. Ovo ima za rezultat da se dobije adresa koja se nalazi na početku 32kB (MSB=0) ili na kraju 32kB (MSB=1) adresnog prostora od 16MB. Kod "apsolutnog dugog" načina adresiranja koriste se dve reči proširenja (MS reč adrese prvo), koje se povezuju da bi se formirala apsolutna memorijska adresa. Na primer, kod instrukcije ADD.W 0x76F1.W,D2 izvorni operand specificira kratku apsolutnu adresu 0x76F1 na početku memorijskog prostora, dok se instrukcijom ADD.W 0xA6F1.L,D2 apsolutna kratka adresa 0xA6F1 specificira na memorijskoj lokaciji 0xFFFFA6F1.

Primer 1.5:

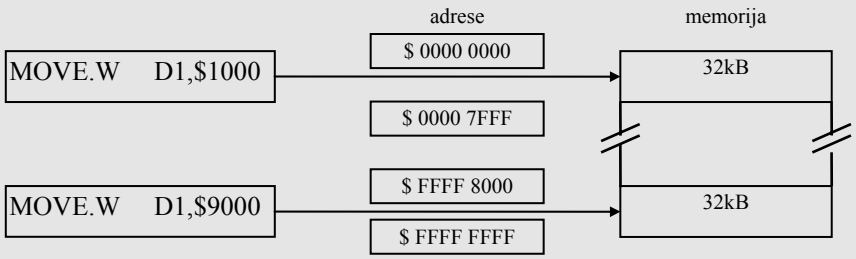
Način formiranja apsolutno kratke i duge adrese prikazan je na slici 1.8a) i b).



Sl. 1.8. Način formiranja apsolutne kratke i duge adrese.

Primer 1.6:

Na slici 1.9 prikazan je efekat kod korišćenja kratkog apsolutnog načina adresiranja. Ako je heksadecimalna adresa između 0 i $7FFF, adresiraju se niža 32kB memorije. Kratke adrese, $8000 i iznad, znakovno se proširuju na 32 bita, što rezultira memorijskim adresama između $FFFF8000 i $FFFFFFFF, a to je viših 32kB adresibilnog memorijskog prostora mikroprocesora MC68020. Na ovaj način projektanti skupa instrukcija tretiraju ova dva memorijska segmenta na neki način kao ekstreme. Uobičajeno, projektanti sistema specificiraju niži memorijski segment kao mesto gde smeštaju sistemske parametre. U suštini, MC68020 koristi prvih 1024 bajtova za svoje vektore, koji definišu početne adrese za trap i prekidne rutine. Viši segment kod najvećeg broja sistema se rezerviše za U/I interfejs. Apsolutni kratki način adresiranja omogućava efikasan pristup fiksnim lokacijama u jednom od ova dva segmenta .



Sl. 1.9.

Primer 1.7:

Kod instrukcije

```
MOVE.W    $12000,D1
```

zahteva se apsolutno dugo adresiranje za izvorni operand jer je adresa veća od 16 bitova.

Kod instrukcije

```
MOVE.B    $3FFF,$12000
```

specificiraju se dva apsolutna adresna načina rada. Izvorna adresa biće $0000 3FFF što će usloviti da asembler koristi dugi način rada.

Primer 1.8:

Sledeća programska sekvenca prikazuje korišćenje adresnih načina rada.

```
MOVE.L    D6,D2       ; (D2)=(D6)
CLR.W     D0          ; (D0)[15:0]=0
MOVEA.L   D0,A3       ; (A3)=(D0)
MOVE.B    $2001,D0    ; (D0)[7:0]=($2001)
MOVE.L    $214AA,$24  ; ($24)=($214AA)
MOVE.L    $0534,$0528 ; ($0528)=($0534)
```

1.7.5. Adresno registarski indirektni način rada

Kod ovog načina adresiranja, adresa operanda se specificira adresnim registrom (mode=2, 3 i 4 u Tab. 1.1). Kod ovog načina adresiranja podržavaju se dve varijante: predekrementiranje i postinkrementiranje. Kada se koristi način rada sa predekrementiranjem, An se prvo dekrementira za veličinu (u bajtovima) operanda (1 za bajt, 2 za reč, a 4 za dugu reč), a zatim se An koristi kao pokazivač operanda. Kod načina rada sa postinkrementiranjem, An se inkrementira nakon korišćenja. Postinkrementiranje i predekrementiranje se koriste kada se pristupa elementima polja, nizovima i drugim strukturama podataka, kao i za implementaciju magacina. Na primer, kod instrukcije

```
ADD.W (A3)+,D2
```

izvorni operand specificira adresno registarski indirektni način rada. Nakon izvršenja instrukcije A3 se inkrementira za dva (jer se operacijom specificira operand tipa reč).

Primer 1.9:

Punjenje registra A1 obavlja se sledećom instrukcijom

```
MOVEA.L   #<addr>,A1
```

kojom se vrši prenos adrese <addr> u A1. Obraćanje (A1) se vrši sledećom instrukcijom

```
MOVE.W    (A1),D1
```

kojom se 16-bitna vrednost koja je upisana na lokaciji <addr> smešta u D1.

Primer 1.10:

Adresa koja se čuva u adresnom registru može se modifikovati u toku izvršenja programa na veći broj načina. Add Address ima format

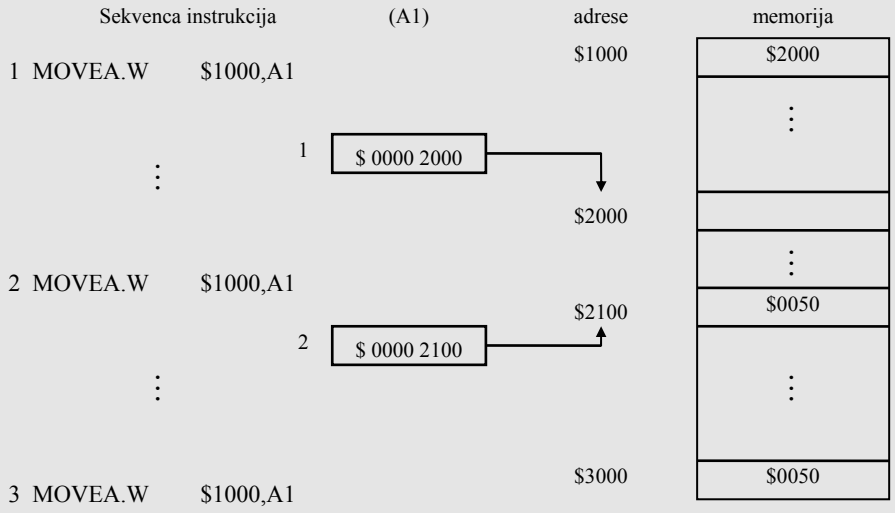
```
ADDA.<X>    <EA>,<An>
```

za sabiranje vrednosti lokacije <EA> sa vrednošću u An. Dužina je ograničena na W ili L za 16 i 32-bitne vrednosti, respektivno. Adresa

<An>←<An>+<EA>

se generiše u toku izvršenja instrukcije ADDA. Operand kome se obraćamo koristeći se registrom An u toku naredne instrukcije usloviće da se adresira nova lokacija. Na slici 1.10 prikazana je jedna moguća sekvenca.

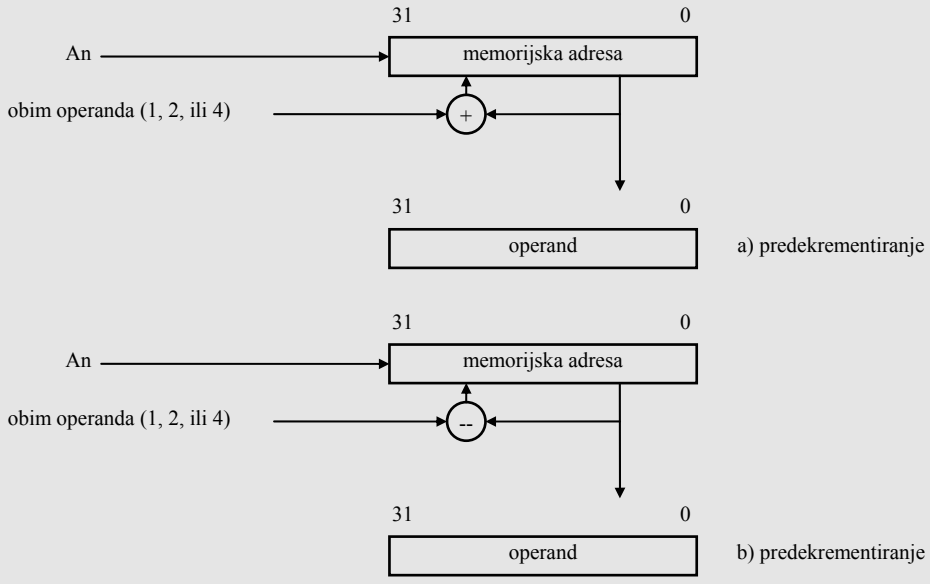
Vrednost $2000, koja predstavlja sadržaj lokacije $1000, prvo se kopira u A1. Ova adresa, $2000, može ukazivati na prvu reč strukture podatka u memoriji. Sabiranjem konstante $1000 koristeći se instrukcijom ADDA (varijacija od ADD), uslovljava se da A1 pokazuje na lokaciju $2100 (pomeraj od 100 lokacija (heksadecimalno) u odnosu na početak strukture podataka). Koristeći A1 u MOVE instrukciji, vrši se prenos vrednosti iz lokacije $2100 u odredišnu lokaciju $3000.



Sl. 1.10.

Primer 1.11:

Princip rada sa postinkrementiranjem i predekrementiranjem je prikazan na slici 1.11a i 1.11.b.



Sl. 1.11. Formiranje efektivne adrese kod postinkrementiranja i predekrementiranja.

Primer 1.12:

Instrukcijom

MOVE.W -(A1),(A2)+

kopira se reč iz lokacije (A1)-2 u lokaciju (A2). Ako se instrukcija izvrši ponovo, izvorna adresa je za jednu reč niže u memoriji, a odredišna za jednu više, u odnosu na početnu. Ova instrukcija se tipično koristi kod programskih petlji kod kojih se autoindeksirana instrukcija repetitivno izvršava, sve dok uslov za izlazak iz petlje nije ispunjen.

Kopiranje blokova podataka iz jednog memorijskog segmenta u drugi se vrši instrukcijom

```
MOVE.W    (A1)+,(A2)+
```

U ovom slučaju (A1) ukazuje na prvi blok a (A2) na drugi. Ovaj tip adresiranja ekvivalentan je sa sledećom sekvencom

```
MOVE.W    (A1),(A2)
ADD.L     #2,A1
ADD.L     #2,A2
```

gde se izvorna i odredišna adresa inkrementiraju za dva nakon prenosa pošto su operandi obima reč.

Primer 1.13:

Sledećom programskom sekvencom kopira se 32-bajtni blok podataka sa lokacije $20000 u lokaciju $30000, ali u obrnutom redosledu. Redosled podataka je obrnut, startujući sa prenosom od prvog bajta izvornog bloka (koristeći postinkrementirajuće adresiranje) ka zadnjem bajtu odredišnog bloka (koristeći adresiranje sa predekrementiranjem). Bajt sa lokacije $20000 se kopira u $3001F, bajt sa lokacije $20001 se kopira u $3001E itd. Registar D1 se koristi kao brojač u petlji, a u registrima A1 i A2 se smeštaju adrese blokova.

```
        MOVE.B    #32,D1  ; postavi brojač na 32
        MOVEA.L   #20000,A1       ; postavi adrese za
        MOVEA.L   #32020,A2       ; prenos
LOOP:   MOVE.B    (A1)+,-(A2)     ; kopiraj naredni bajt
        SUBI      #1,D1           ; dekrementiraj brojač
        BNE       LOOP            ; brojač=0
```

1.8. Dvokomponentni adresni načini rada

U ovu grupu spadaju sledeća dva načina adresiranja:

1. Adresno registarsko indirektno sa razmeštajem - Ovaj način adresiranja (mode=5 Tab. 1.1) uobičajeno se zove bazno ili bazno relativno, a zahteva jednu reč proširenja za specifikaciju 16-bitnog razmeštaja. Razmeštaj (znakovno proširen) se sabira sa sadržajem adresnog registra, pa se na taj način formira efektivna adresa koja se koristi za pribavljanje operanda. Razmeštaj se može smatrati da je označen tako da se njegov opseg kreće od -32768 do +32767. Tipičan primer instrukcije koja koristi ovaj način adresiranja je

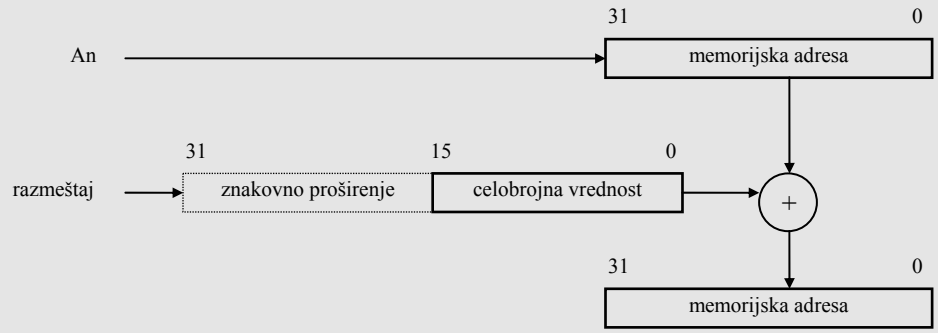
   ```
   ADD.W 16(A3),D2
   ```

   kod koje se adresa izvornog operanda izračunava kao ea=(A3)+16.

   Efektivna adresa kod ovog načina adresiranja se izračunava kao (ea)=(An)+<d16>, gde je <d16> 16-bitna celobrojna označena vrednost. Razmeštaj se smešta kao reč proširenja (u mašinskoj instrukciji). Asemblerska notacija je MOVE.W <d16>(An),<ea > gde je <d16> specificiran kao heksadecimalna ($XXXX) ili decimalna (XXXX) konstanta. Na primer, MOVE.W $20(A1),D1 vrši prenos 16 bitova u registar D1 sa lokacije koja je 32 bajta viša od adrese koja je smeštena u A1. Forma instrukcije MOVE.W ($20,A1),D1 je ekvivalentna. Simbol za neposrednu vrenost "#" nije potreban jer asembler prepoznaje <d16> kao pomeraj, a ne kao adresu.

Primer 1.14:

Na slici 1.12 prikazan je način realizacije adresno registarsko indirektnog adresiranja sa razmeštajem.



Sl. 1.12. Formiranje efektivne adrese kod indireknog adresno registarskog adresiranja sa razmeštajem.

2. PC indirektno sa razmeštajem - Kod ovog načina adresiranja (mode=7.2 Tab. 1.11), uobičajeno se zove bazno ili relativno bazno, efektivna adresa se formira kada se PC-u koji predstavlja bazni registar doda 16-bitni znakovno proširen razmeštaj (ea=(PC)+<offset>). Ovaj način se razlikuje od bazno adresnog (koristi An registar) u tome što se PC adresni način rada ne može koristiti za specifikaciju odredišnog operanda, jer to može uzrokovati samomodifikujući kod. Ovaj način adresiranja je koristan za generisanje relokatibilnog koda, jer je odredište, da kažemo od, "jump" instrukcije nezavisno od pozicije programa u memoriji. Instrukcija koja koristi ovaj način adresiranja ima oblik ADD.W 16(PC),D2.

Primer 1.15:

Instrukcijom

MOVE.L (DATA,PC),D1

kopiraju se 32 bita sa lokacije adresirane kao (PC)+(DATA) u D1, a rezultat je (D1)=((PC)+Data). Vrednost DATA može biti 8-, 16-, ili 32-bitni označeni razmeštaj.

1.9. Višekomponentni adresni načni rada

MC68020 nudi neznatno prošireni oblik baznog indeksnog adresnog načina rada za sliku 1.7, zbog toga što se 8-bitni razmeštaj sabira sa baznim (adresnim) registrom i indeks registrom. Drugim rečima, postoje oba načina adresiranja, adresno-registarsko-indirektno i PC-indeksirano, a razlika je u tome što se PC-indeksirano adresiranje ne može koristiti za specifikaciju adrese odredišnog operanda. Specijalni način, koji nije opisan na slici 1.7, koristi se za adresiranje jedinstvenog bita.

1.9.1. Adresno registarsko sa adresiranje indeksom i razmeštajem

Kod ovog načina rada (mode=6 Tab. 1.1), reč proširenja (slika 1.5a)) specificira indeksni registar (Dn ili An registar), obim indeksnog registra (znakovno proširenje, LS reč ili ceo registar), faktor umnožavanja i 8-bitni razmeštaj. Umesto 8-bitnog razmeštaja, moguće je koristiti 16- ili 32-bitni razmeštaj ali su tom slučaju jedna ili dve reči proširenja potrebne. Instrukcije imaju oblik

D.L (A3,D1.W*4),D2

gde je A3 bazni a D1 indeksni registar (W specificira da se koristi LS reč, faktor umnožavanja 4 specificira operand tipa duga reč), a sve ovo rezultira da se formira sledeće efektivna adresa izvornog operanda

ea=8+(A3)+4*(D1).

Primer 1.16:

Efektivna adresa kod ovog načina adresiranja izračunava se na sledeći način

ea=(An)+(Xi).[S]*SCALE+<disp>

gde je Xi indeks registar, može biti An ili Dn. Notacija Xi.[S] obezbeđuje [S]=W ili L označavanje, respektivno. Bilo koja 16-bitna indeksna vrednost, kada se koristi za izračunavanje adrese, uvek se znakovno proširuje na 32 bita. Ako je vrednost indeks registra heksadecimalan broj od $8000 do $FFFF, vrednost se smatra negativnom. Faktor množenja, SCALE, omogućuje umnožavanja za 1 (nema umnožavanja), 2, 4 ili 8. Sa vrednošću SCALE=n indeksni registar se efektivno indeksira sa n bajtova. Vrednost SCALE=2 ukazuje na adresiranje reči, a SCALE=4

ukazuje na adresiranje dugih reči. Vrednost razmeštaja može biti 8-bitna kao što je prikazano na slika 1.13a) i definiše efektivnu adresu koja se određuje na sledeći način:

$$ea=(An)+Xi.[S]*SCALE+<d_8>$$

gde je $<d_8>$ 8-bitna označena celobrojna vrednost.

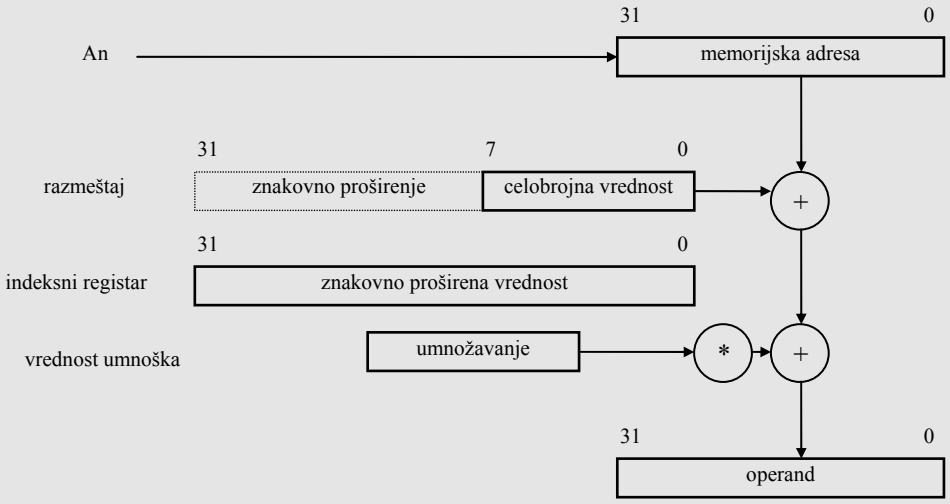
Asemblerskom instrukcijom

MOVE.W 2(A1,D1.W),D2

kopiraju se 16 bitova sa adrese

$$ea=(A1)+D1[15:0]+2$$

u registar (D2)[15:0]. Ovaj način rada zahteva jednu reč proširenja instrukcije u memoriji.



Sl. 1.13. (a) Adresno registarski indirektni sa indeksiranjem (8-bitova).

Na slici 1.13b prikazano je izračunavanje efektivne adrese kod registarko indirektnog adresiranja sa indeksnim i baznim razmeštajem. Ovaj način se razlikuje od prethodnog u tome što je bazni razmeštaj 16-bitna (znakovno proširena) ili 32-bitna vrednost. Takođe, sva tri operanda specificiranog sumatora su opciona.

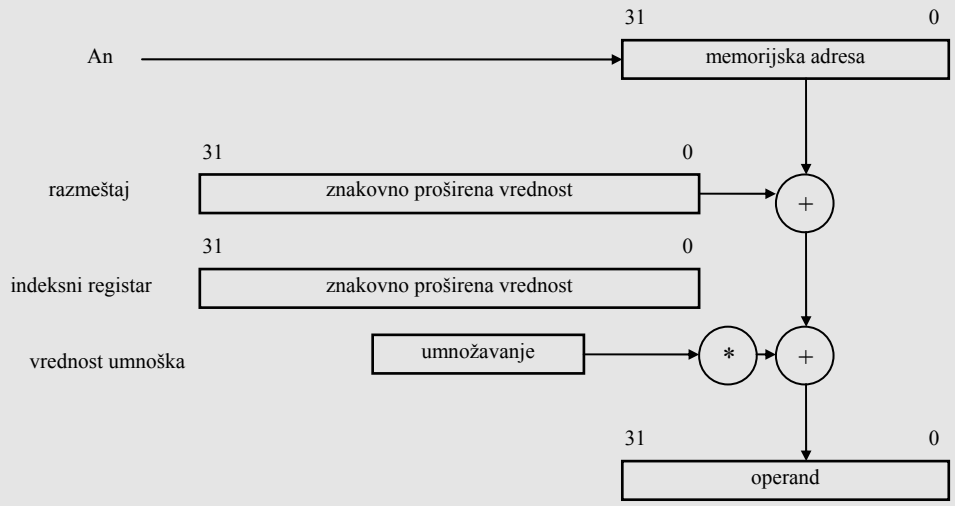
Kod instrukcije

MOVE.L ($900,A3,D2.W*4),D1

sa (A3)=$0002 0000 i (D2)=$0000 1000 imaćemo da je efektivna adresa

20000	baza
4000	indeks, umnožen
900	bazni razmeštaj
249000	

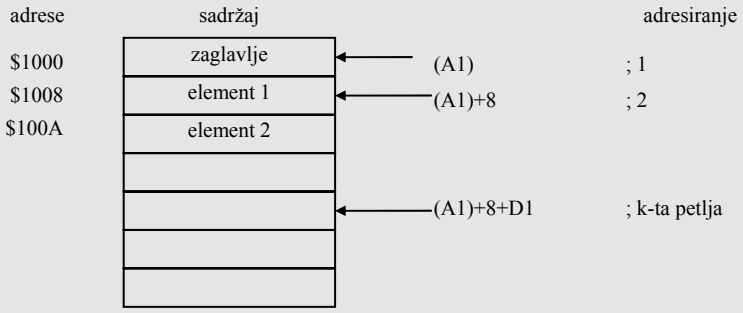
Za dati primer su potrebne dve reči proširenja.



Sl. 1.13. (b) Adresno registarsko indirektno sa indeksom i baznim razmeštajem.

Primer 1.17:

Usvojimo da je lista reči smeštena u memoriji počev od adrese $10000. Ako se prva četiri 16-bitna podatka u listi koriste za određivanje dužine liste i slične informacije, prvom elementu podataka se može pristupiti dodavanjem razmeštaja od 8 bajtova početnoj adresi. Uzastopnim elementima liste može se pristupiti indeksiranjem i umnožavanjem. Ova šema je prikazana na slici 1.14, a koristi A1 kao bazno adresni, a D1 kao indeksni registar.



Sl. 1.14. Adresiranje pomoću baznog i indeksnog registra.

Programska sekvenca ima sledeći oblik

```
        MOVE.L      $10000,A1       ; početak liste
        CLR.L       D1              ; indeks=0
LOOP:   MOVE.W      (8,A1,D1.L*2)   ; kopiraj elemenat
                                    ; u D2 pomnoži
                                    ; indeks za reč
        ADD.W       #1,D1           ; sledeća reč (granaj se na LOOP ako nije kraj)
```

1.9.2. PC indirektno adresiranje sa indeksom i razmeštajem

Ovaj način adresiranja (mod=7.3 Tab. 1.1) analogan je adresno registarskom indirektnom adresiranju sa razmeštajem, a koristi se za pristup tabelama u koje su upisane konstante, ili za skok na određenu lokaciju. Razmeštaj specificira rastojanje od tekuće instrukcije do željene tabele, a indeksni registar se koristi za izbor elementa u tabeli. Moguće je koristiti "kratak" ili "dugi" indeks. Kod instrukcije

ADD.W 8(PC,A3.L*2),D0

efektivna adresa izvornog operanda se izračunava na sledeći način:

ea=8+(PC)+2*(A3)

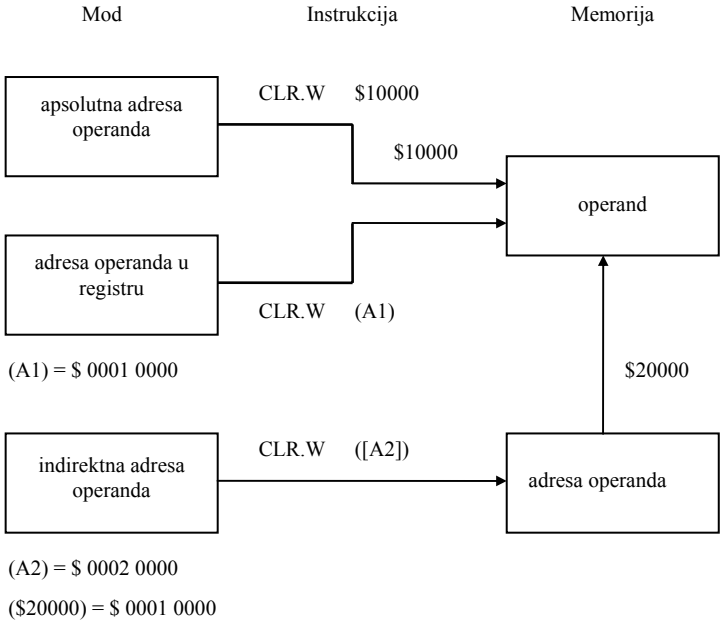
pri čemu se koristi LS i MS reč indeksnog registra A3.

1.10. Memorijsko indirektno adresiranje

Kod apsolutnog adresnog načina rada, adresa specificira koja lokacija sadrži operand. Registarsko indirektno adresiranje omogućava da adresa, koja se nalazi u An, specificira memorijsku lokaciju operanda. An registar, modifikovan sa indeksom i sabranim razmeštajem, pokazuje na operand. Kod memorijsko indirektnog načina adresiranja omogućeno je da adresa koja se čuva u memoriji ukazuje na operand. Ova adresa koja je smeštena u memoriji predstavlja 32-bitna vrednost a koristi se kao deo u procesu određivanja efektivne adrese. Upoređenje različitih adresnih načina rada prikazano je na slici 1.15. Na ovoj slici izabran je najjednostavniji memorijsko indirektni način adresiranja, a za primer je uzeta instrukcija

CLR.W ([A2])

gde [A2] ukazuje da je adresa u memoriji. Prema tome [A2] specificira sadržaj sadržaja registra A2 kao adresu operanda. efekat instrukcije ogleda se u brisanju lokacije na adrsi ((A2)), tj. sadržaj lokacije briše se od strane svake instrukcije.



Sl. 1.15. Formiranje efektivne adrese kod različitih adresnih načina rada.

Primer 1.18:

Sekvencom instrukcija koja koristi registarsko indirektno adresiranje briše se sadržaj lokacije $10000

```
MOVE.L  $20000,A1   ; (A1)←(lokacija $20000)
CLR.W   (A1)
```

Ovom sekvencom briše se lokacija na koju ukazuje adresa u lokaciji $20000. Ako je ($20000)=$10000, vrednost reči na lokaciji $10000 je nula. Isti efekat se dobija instrukcijom

```
CLR.W   [(A2)]
```

ako A2 sadrži $20000. Jedna od očiglednih prednosti memorijsko indirektnog načina rada je da je broj indirektnih adresa (čuvaju se u memoriji) praktično neograničen, ako se A2 menja da ukaže na različite memorijske lokacije. Ovo se lako izvodi korišćenjem indeksiranja.

Dva memorijsko indirektna načina adresiranja koja koriste adresne registre su moguća kod MC68020. Prvi se zove postindeksiranje a drugi preindeksiranje. Na slici 1.16 prikazano je izračunavanje efektivne adrese kod indirektnog načina rada sa postindeksiranjem. Izračunavanje ima oblik

ea=[<bd>+An]+Xn.[S]*SCALE+<od>

gde se memorijska lokacija u kojoj se čuva indirektna adresa određuje sabiranjem baznog razmeštaja sa (An). Ova vrednost se koristi kao bazna adresa za memorijski operand, koji se locira sabiranjem indeksne vredosti sa dodatnim razmeštajem. Kao i kod drugih načina adresiranja, bazni razmeštaj <bd> i dodatni razmeštaj <od> su 16-bitne (znakovno proširene) ili 32-bitne vrednosti. Indeksni registar se tretira kao znakovno proširena reč ([S]=W) ili kao 32-bitna vrednost ([S]=L), i skalira se za 1 (nema skaliranja), 2, 4 ili 8.

Tipična minimalna specifikacija, sadrži adresni registar, kao u sledećoj instrukciji

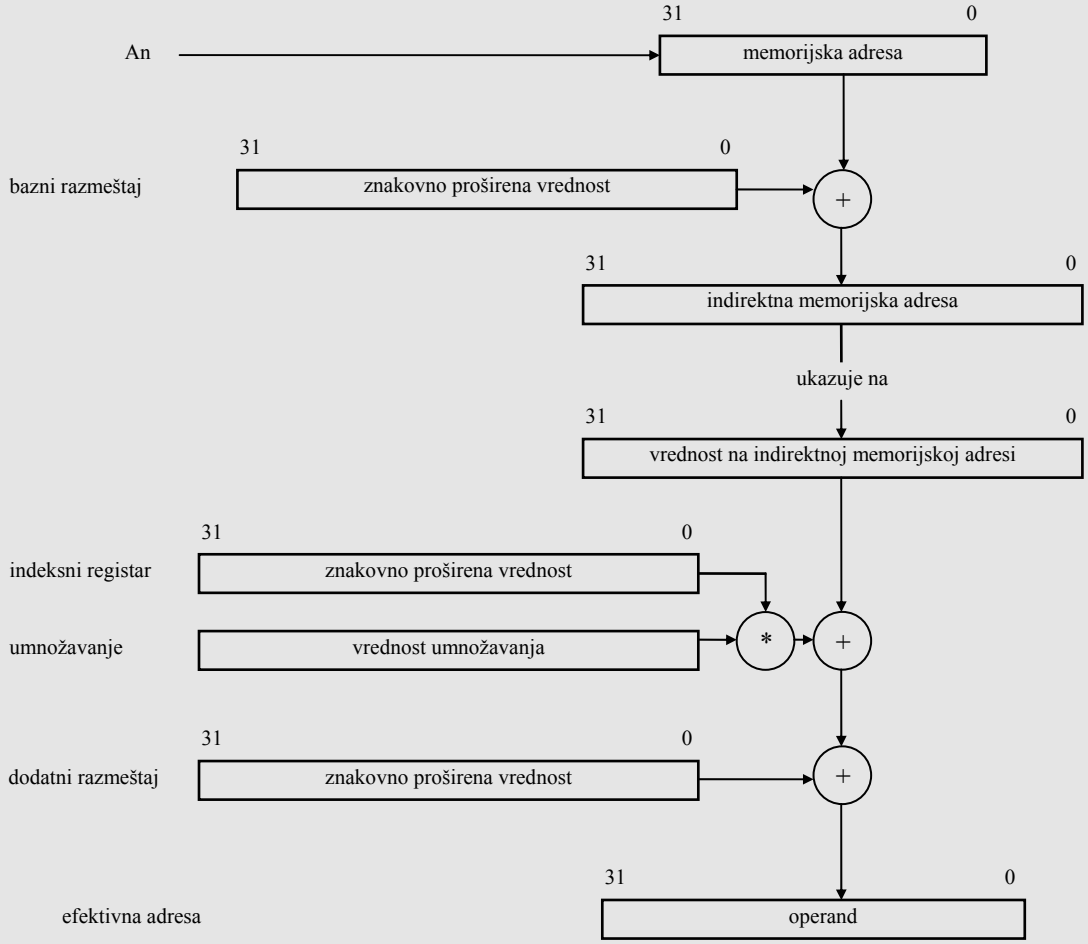
JMP ([A1])

koji uslovljava prenos upravljanja na adresu specificiranu lokacijom na koju ukazuje A1 (koristi se kod "jump table" ili "dispatch table").

Drugi tipičan primer

JMP ([A1],D1.L*4)

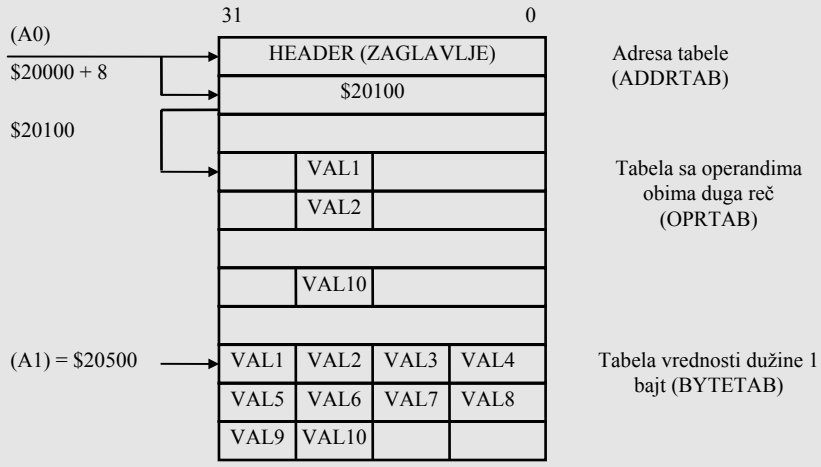
gde [A1] ukazuje na start neke rutine, a D1 bira korektnu startnu adresu u okviru rutine. Ona se množi sa četiri da ukaže na 32-bitnu adresu. Na primer, ako je [A1]=$10000, a (D1)=4, početna adresa četvrtog elementa treba da je $10010, a to je četiri duge reči ili 16 bajtova od početne adrese prvog elementa.



Sl. 1.16. Izračunavanje efektivne adrese kod indirektnog adresiranja sa postindeksiranjem.

Primer 1.19:

Posmatrajmo strukturu podataka na slici 1.17 koja prikazuje tabelu čija je početna adresa ADDRTAB=$20000.



Sl. 1.17.

Adresa nakon HEADER informacije ukazuje na operandski elemenat u drugoj tabeli sa adresom OPRTAB=$20100. Usvojimo da tabela na adresi ADDRTAB ima 8-bajtno zaglavlje iza koje sledi adresa tabele operanda. Prikazanom programskom sekvencom vrši se prenos drugog bajta od svakog od 10 četvorobajtnih operanada u tabeli definisanoj sa BYTETAB=$20500. Nakon inicijalizacije registara, u toku izvršenja programa, A0 ukazuje
na početnu adresu tabele, a A1 pokazuje na tabelu bajtova.

Adresa tabele 32-bitnih operanda je data sa (A0)+8 kod MOVE.B instrukcije na labeli LOOP. Svaki bajt koji se prenosi iz tabele koja počinje na OPTAB je pomeren za +1 od početka svakog 32-bitnog opernada u ovoj tabeli. Ovi 32-bitni operandi se adresiraju u sekvenci inkrementiranjem registra D0, umnoženog za četiri, svaki put kada se izvrši instrukcija kopiranja. Adresa bajt tabele BYTETAB za prenete bajtove, čuva se u A1, i postinkrementira se za jedan nakon svakog prenosa. D1 se dekrementira od 10 do nule sa ciljem da se upravlja brojem prenosa. Kada je (D1)=0, izvršeno je 10 petlji i program se završava.

Instrukcija MOVE.B koristi memorijsko indirektno adresiranje sa postindeksiranjem da bi se adresirali bajtovi u 32-bitnim operandima.

U ovom slučaju adrese različitih tabela definisane su EQU direktivama. Dužina tabele operanda je fiksirana na 10. Ove vrednosti se ne mogu menjati a da se ne izvrši ponovno asembliranje programa nakon modifikacije vrednosti. 32-bitni operandi moraju se smestiti u tabelu startujući od OPRTAB pre izvršenja programa. Informacija o "HEADER"-u ne koristi se u ovom primeru.

```
                        LLEN        100             ; dužina programa
                        ORG         $10000          ; početna adresa programa
                ADDRTAB:EQU         $20000          ; adresa tabele
                OPRTAB: EQU         $20100          ; tabela operanda
                TLENGHT:EQU         10              ; dužina tabele
                BYTETAB:EQU         $20500          ; tabela bajtova
početna adresa programa
        10000           MOVE.L      #TLENGTH,D1             ; postavi brojač na 10
                        MOVEA.L     #ADDRTAB,A0             ; adresa tabele
                        MOVEA.L     #BYTETAB,A1             ; bajt tabela
                        CLR.L D0                            ; indeks za operande
                LOOP:   MOVE.B      ([8,A0],D0.L*4,1),(A1)+ ;kopiraj sledeći bajt
                        ADDI.L      #1,D0                   ; povećaj indeks
                        SUBI.L  #1,D1               ; smanji brojač
                        BNE         LOOP            ; produži sve dok brojač=0
                        JMP         KRAJ            ; skok na labelu kraj
                                                    ; kraj nije prikazan
```

Na slici 1.18 prikazano je izračnavanje efektivne adrese kod memorijskog indirektnog adresiranja sa preindeksiranjem. Efektivna adresa se izračunava na sledeći način

ea=[<bd>+An+(Xn).[S]*SCALE]+<od>

gde je [S]=W ili L, SCALE=1 (nema umnožavanja), 2, 4 ili 8, a <bd> i <od> su 16-bitni (znakovno prošireni) ili 32-bitni razmeštaji.

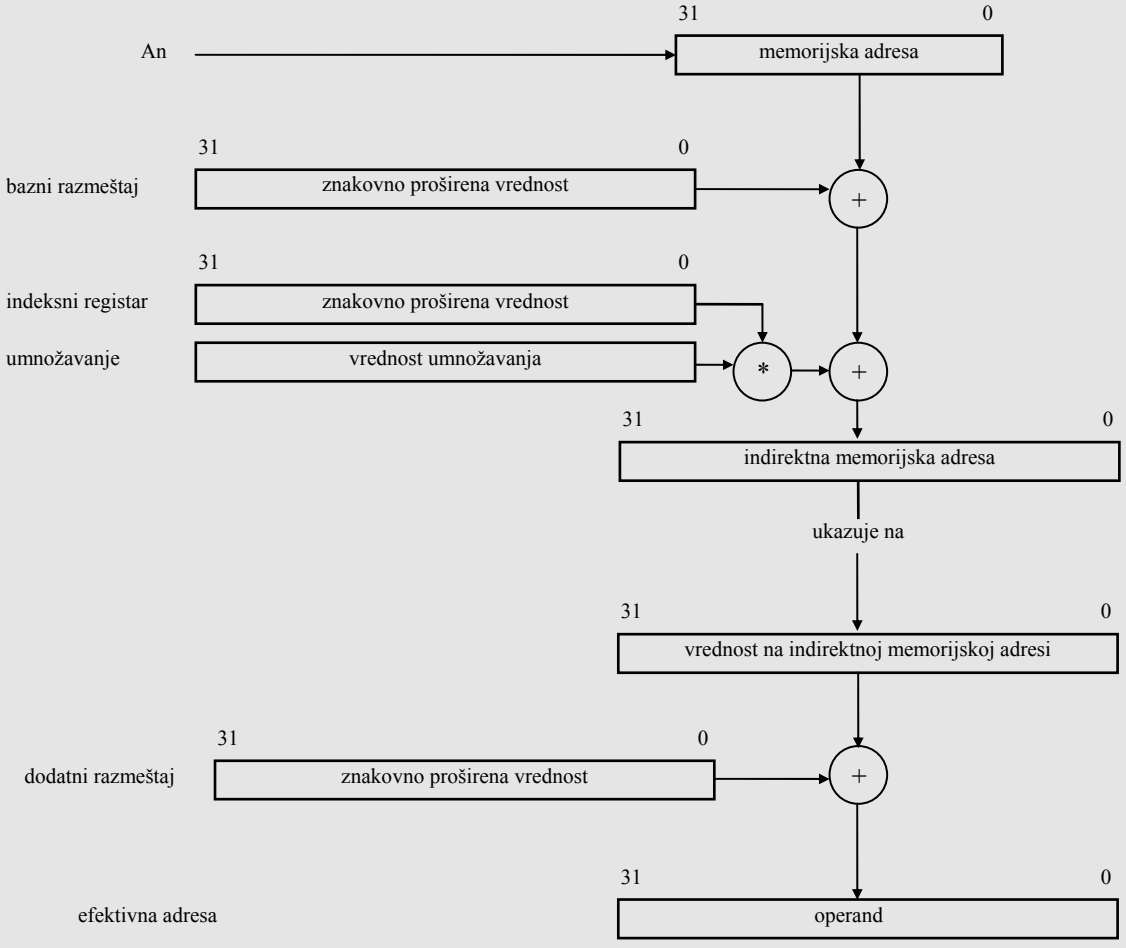
Sledećom instrukcijom

MOVE.L ([$900,A3,D2.W*4],$100),D1

sa (A3)=$20000 i (D2)[15:0]=$20 specificira se indirektno adresiranje izvornog operanda prvo na adresi

```
0002 0000   početna adresa tabele
     0900   razmeštaj
     0080
     ____
0002 0980
```

zatim se dodaje pomeraj od $100 sadržaju lokacije $20980 da bi se locirala adresa 32-bitnog operanda. Ako je ($20980)=$1F000, operand je na lokaciji $1F100. Naravno CPU ne interpretira značenja različitih komponenata adrese nego samo vrši izračunavanje adrese operanda. Nakon izvršenja instrukcije (D1)=($0001F100) tj. dobija se 32-bitni rezultat.



Sl. 1.18.

Primer 1.20:

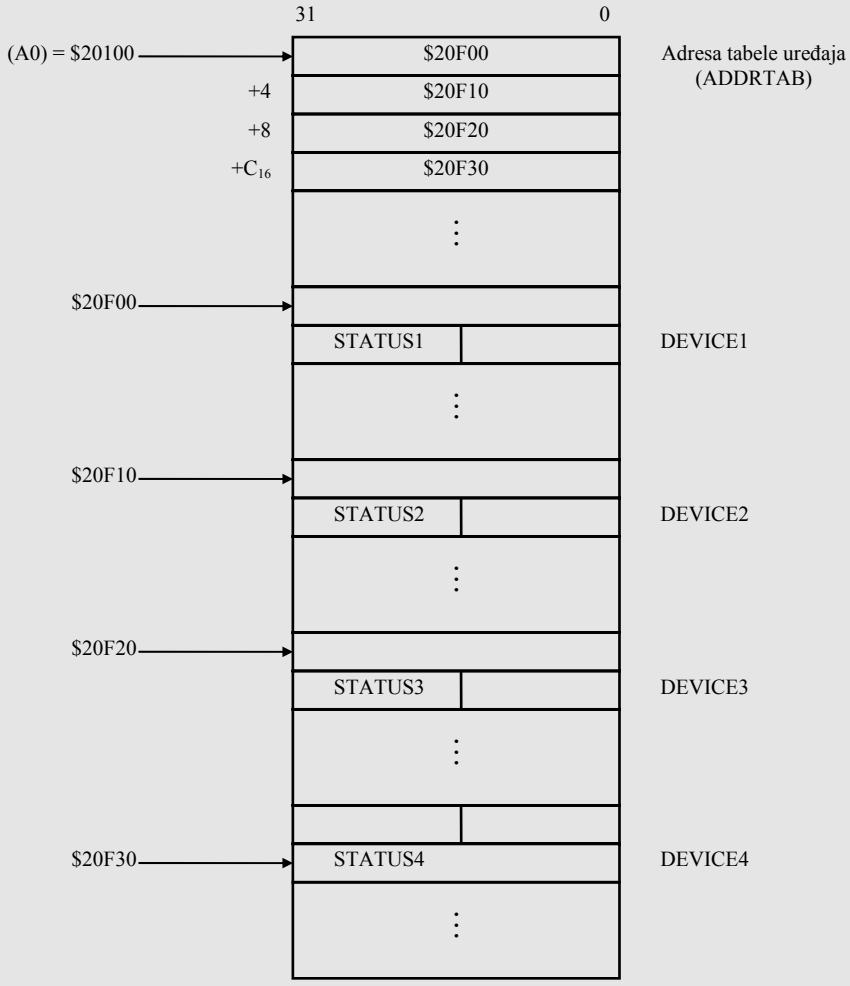
Na slici 1.19 prikazana je struktura podataka u memoriji za tabelu adresa koje startuje sa lokacije $20100. Svaki element u tabeli ukazuje na drugu tabelu, u kojoj se čuvaju informacije o U/I uređajima sistema. Status svakog od četiri uređaja se čuva u 16-bitnoj reči. Takođe se za svaki uređaj čuva i 32-bitni pomeraj od početka tabele. Program koji sledi kopira status svakog uređaja iz tabele i smešta ga u sistemski magacin. Početna adresa tabele svakog uređaja se adresira sa memorijsko inirektnim adresiranjem sa preindeksiranjem pri čemu se A0 koristi kao bazni registar, a D0 kao indeksni registar. Vrednost "STATUS" predstavlja dodatni pomeraj plus četiri bajta od

startne adrese. Petlja se izvršava četiri puta da bi se izvršio prenos 16-bitnih statusnih vrednosti u magacin. Sa te tačke gledišta, drugi program (nije prikazan u ovom primeru) treba da obradi svaku vrednost i odredi status uređaja. Tipično, uređaj je bilo zazuzet (busy) ili je spreman (ready) za prenos podataka.

Instrukcija kao što je

```
MOVE.W          (A7)+,D1
```

treba da se koristi za prenos statusa iz magacina u D1 radi dalje obrade. Svaka adresa tabele uređaja i broj uređaja (DEVCNT) u datom programu je fiksiran. Drugi program mora da smešta informaciju o uređaju u četiri tabele pre izvršenja datog programa.



Sl. 1.19.

```
            LLEN    100
DEVICE1     EQU     $20F00          ;definisanje adresa
DEVICE2     EQU     $20F10
DEVICE3     EQU     $20F20
DEVICE4     EQU     $20F30
ADDRTAB     EQU     $20100          ;tabela adresa uređaja
DEVCNT      EQU     4               ;broj uređaja
```

```
        STATUS  EQU     4                                   ;pomeraj za status
                ORG     $10000                              ;početna adresa programa
10000           CLR.L   D0                                  ;brojač petlje
                MOVEA.L #ADDRTAB,A0                         ;bazna adresa u A0
        LOOP    MOVE.W  ([0,A0,D0.L*4],STATUS),-(A7)
                ADDI.L  #1,D0                               ;sledeći uređaj
                CMPI.L  #DEVCNT,D0                          ; DEVCNT petlji
                BNE     LOOP
                JMP     KRAJ                                ;skok na kraj programa
20100           ORG     ADDRTAB
20100           DC.L    DEVICE1,DEVICE2,DEVICE3,DEVICE4
```

1.11. Bit adresiranje

Kod instrukcije za manipulaciju sa bitovima, operand (memorijski bajt ili podatak u memoriji) se adresira, koristeći se opštim operandom. Individualni bit se adresira njegovim bit brojem u tom bajtu ili registru, gde bit nula se uvek odnosi na LS bit. Broj bita se može specificirati statički kao neposredni podatak ili dinamički u Dn registru. Ipak postoje neka ograničenja. Ako opšti operand specificira Dn registar, usvaja se da je to 32-bitni podatak i da je numeracija bitova po modulu 32. Ako opšti operand specificira memorijsku lokaciju usvaja se da je to bajt lokacija i da je numerisanje po modulu 8.

Instrukcija

```
                BSET    D1,D0
```

postavlja bit D0[(D1)modulo32], dok instrukcija

```
                BSET    #3,D0
```

postavlja bit D0[3].

1.11.1. Adresiranje bit-polja

Bit polje se specificira pomoću bazne adrese (izborom memorijskog bajta, bit ofset poljem (ukazuje na bit koji je zadnji levi u bit polju u odnosu na MS bit baznog bajta), i obimom bit polja (određeno brojem bitova u bit polju). Pomeraj može biti u opsegu od -2 do 2 -1, a obim bit polja može da se menja između 1 i 32. Ne postoje ograničenja kako su bit polja poravnana u memoriji; na primer bit polje od 26 bitova može da zauzme 5 memorijskih bajtova. Procesor je u stanju da prepozna uslov sa ciljem da pribavi odgovarajuće bajtove.