5. ADRESIRANJE

5.1. Opšte osobine adresiranja

Instrukcije i podaci se smeštaju u memoriju računara. Fizička struktura ovakve memorije se može posmatrati na sledeći način: memoriju čine jedinice (reči ili bajtovi) pri čemu se svaki sastoji od fiksnog broja memorijskih elemenata (nazvanih bitovi), a svaka jedinica može da sadrži bilo instrukcije, bilo podatke. Svaka jedinica ima jedinstveno ime, koje se zove adresa, formirano pomoću kombinacije bitova. Broj adresnih bitova jednak $\log_2(N)$, gde je N maksimalni broj jedinica.

U daljem tekstu analiziraćemo neke opšte osobine adresiranja koje se odnose na adresnu rezoluciju, poravnanje, redosled bitova, bajtova, reči i adresnih prostora.

5.1.1. Adresna rezolucija i poravnanje podataka

Adresna rezolucija specificira najmanji mogući iznos informacije (u bitovima) koja se može od strane arhitekture direktno adresirati. Apsolutni minimum je jedan bit. Neke arhitekture kao što su iAPX 432 sposobne su da direktno adresiraju jedan bit. Adresna rezolucija od jednog bita se retko koristi, jer zahteva veliki broj adresnih bitova i dodatno vreme da se podaci tako poravnaju da su bitovi u korektnoj poziciji kada se podacima manipuliše od strane ALU ili oni se smeštaju/čitaju u/iz memorije.

Veliki broj savremenih arhitektura kodiraju instrukcije u grupama bitova koje imaju dužinu po 8 bitova (VAX-11), 16 bitova (MC68020, 80286, IBM/370) ili umnožak ovih brojeva. Ovo ima za posledicu da se dužina adrese smanji za tri ili četiri bita u odnosu na bit adresiranje, kao i da se smanji vreme poravnavanja.

Kod najvećeg broja klasičnih arhitektura adresna rezolucija za podatke i instrukcije je ista. U Tabeli 5.1 prikazana je adresna rezolucija za podatke i instrukcije kod nekoliko dobro poznatih arhitektura.

Tab. 5.1. Adresna rezolucija kod nekih postojećih mašina.

Rezolucija	MC68020	VAX-11	NS32000	IBM/370	B1700	B6700	iAPX432
Instrukcije	16	8	8	16	1	48	1
Podaci	8	8	8	8	1	48	8

B1700 i iAPX 432 omogućavaju da instrukcije počinju nad bilo kojoj bit granici i imaju dužine koje nisu umnožak od 8 ili 16 bitova. Prednost ovakvog pristupa je kompaktnije kodiranje, koje se postiže na račun većeg broja adresa po instrukciji, na primer tri do četiri bita više u odnosu na bajt adresiranje. Adresna rezolucija od 16 bitova kod instrukcija je rezultat činjenice da ove mašine imaju instrukcije koje su 16-bitne. Ovo znači da kod 16-bitne, rečno organizovane memorije, najveći broj instrukcija se pribavlja u toku jednog memorijskog ciklusa.

Primer 5.1:

Kod MC68020 postoje instrukcije za testiranje, brisanje, promenu ili postavljanje jedinstvenog bita u okviru odredišnog operanda. Broj bitova se može specificirati kao neposrdni operand ili može biti prisutan u registru za podatke. Tipični primeri korišćenja ovih instrukcija su

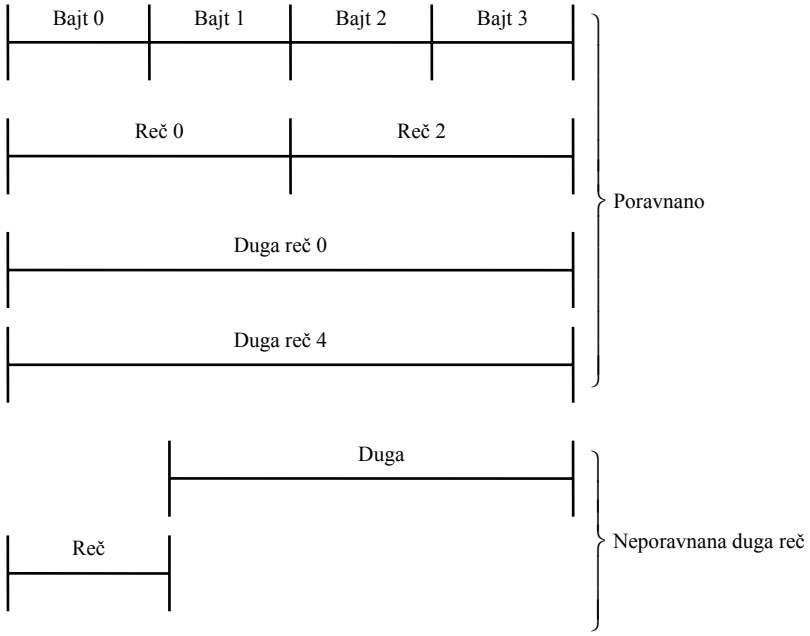
```
BTST.B    #3,Dst   ; testiraj treći bit bajta lociranog u Dst
BCLR.L    #0,D1    ; briši bit 0 u registru za podatke 1
BSET.L    D,D2     ; postavi bit 31 u registru za podatke D2
                   ; (usvajajući da D1 sadrži vrednost 31)
```

Ako je odredišni operand registar za podatke, tada je brojanje bitova po modulu 32, što obezbeđuje manipulaciju svim bitovima u registru za podatke. Ako je odredište memorijska lokacija, bit operacija se obavlja u jednom bajtu, koristeći broj bita po modulu 8. Bit broj 0 odnosi se na LS bit.

Poravnanje je zahtev kojim se traži da podaci i/ili instrukcije budu smeštene u memoriju na takav način (mada adresna rezolucija može biti 8-bitna) da su: adrese 16-bitnih reči na adresama koje su umnožak od dva (parno

poravnanje adrese); adrese 32-bitnih dugih reči na adresama koje su umnožak od četiri (poravnanje tipa duga reč). Ovakvim uređenjem postiže se veća brzina rada u toku izvršenja programa (slika 5.1).

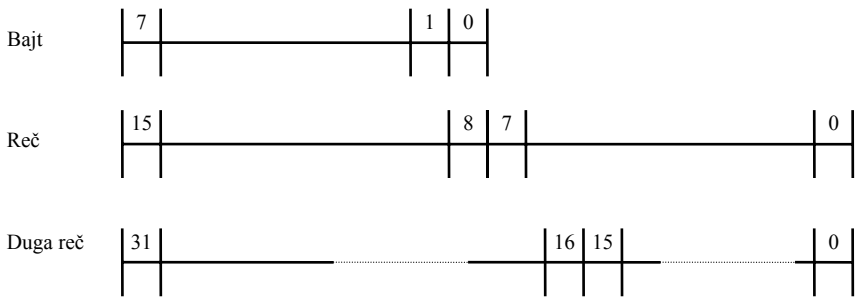
Ako 32-bitna duga reč ima adresu 0001, to znači da će bajtovi na adresama 0001, 0002, 0003 i 0004 formirati dugu reč (slika 5.1), ali će 32-bitnoj mašini biti potrebno da obavi dva memorijska ciklusa čitanja da bi se podatak prihvatio. Da je podatak bio poravnan, ukupni 32-bitni podatak bi bio pribavljen u toku jednog memorijskog ciklusa. U opštem slučaju poravnani podaci (čak i kada nije neophodno) su poželjni. To znači da reči budu u granicama koje odgovaraju rečima, a duge reči na granicama koje odgovaraju dugim rečima. Ovakav pristup dovodi da se postižu bolje performanse sa stanovišta brzine rada. Kod velikog broja arhitektura, kada su u pitanju podaci, neophodno je obaviti samo bajtovsko poravnanje, a kada se govori o magacinu uvek se manipiliše poravnanjem tipa reč.



Sl. 5.1.

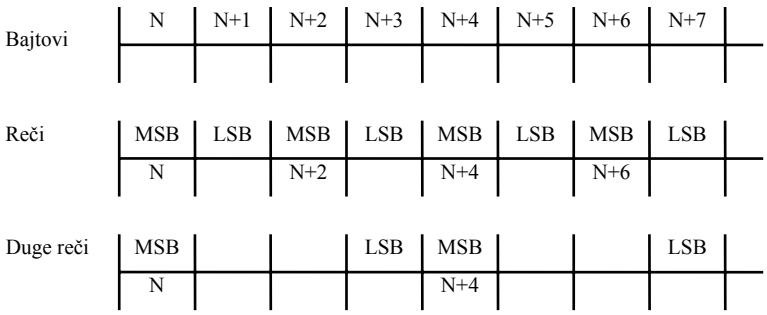
5.1.2. Redosled na nivou bita, bajta i reči

Na slici 5.2 prikazano je numerisanje bitova u bajtu, reči i dugoj reči za MC68020.



Sl. 5.2.

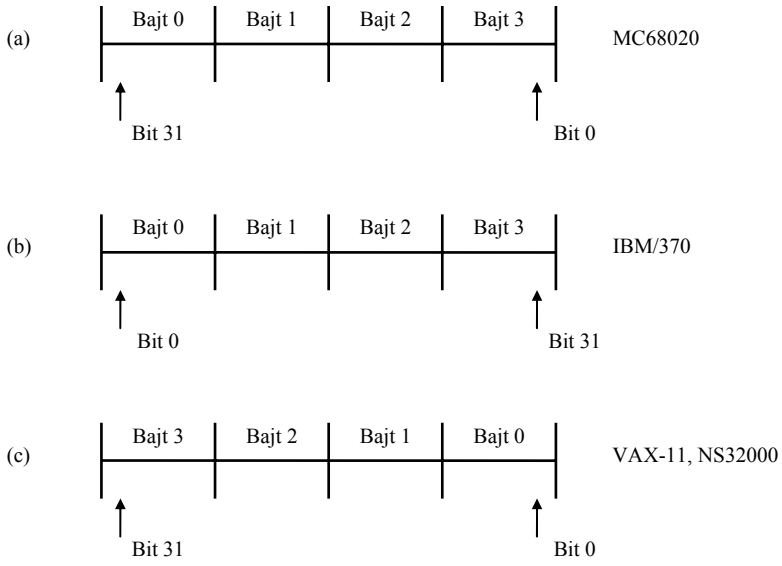
Numerisanje se izvodi automatski sa desna na levo, počev od LSB pozicije. Numerisanje bajtova u 16-bitnoj reči i 32-bitnoj duploj reči za MC68020 prikazano je na slici 5.3.



Sl. 5.3. Redosled bajtova kod MC68020.

U ovom slučaju numerisanje se izvodi sa leva na desno, a to je nekonzistentno sa numerisanjem bitova u bajtu. Na primer, 16-bitnu reč čine dva bajta, pri čemu levi bajt ima niži broj. Adresa reči ili duge reči je adresa čiji je numerički broj najmanji, ali je to MS bajt, kao što je prikazano na slici 5.3.

Sve arhitekture nemaju istu strukturu (u zavisnosti od redosleda bitova, bajtova ili reči) kao što je to kod MC68020 (slika 5.4a). Konzistentniju bit i bajtovsku šemu numerisanja možemo sresti kod IBM/370 (slika 5.4b), gde se bitovi i bajtovi numerišu sa leva na desno. Drugi pristup se može sresti kod VAX-11 i NS32000 (slika 5.4c), gde se bitovi i bajtovi numerišu sa desna na levo.



Sl. 5.4. Različite šeme označavanja bitova i bajtova.

Način na koji se vrši numerisanje bitova i bajtova je važan kada se vrši prenos programa na mašinskom jeziku sa jedne arhitekture na drugu.

5.1.3. Adresni prostori

Najveći broj računarskih arhitektura poseduje, logički posmatrano, veći broj različitih adresnih prostora. Razlog ovome je da se određeni prostori koriste za određenu namenu. Najvažniji prostori su sledeći:

1. **Radna memorija ili registarski prostor** je prostor u kome se nalaze registri opšte namene; na primer, MC68020 ima u ovom prostoru osam registara za podatke i osam za adrese. Prednost korišćenja ovih registara ogleda se u kompaktnoj operandskoj specifikaciji (tri ili četiri bita), što u se suštini svodi na favorizaciju dužine instrukcije. Ovim se štedi memorijski prostor i obezbeđuje brže izvršenje instrukcionog ciklusa pribavljanja (fetch). Vreme pristupa registru je znatno kraće od vremena pristupa memoriji.
2. **Prostor glavne memorije** - u ovom prostoru se čuvaju programi i podaci. Oblast glavne memorije (često se zove memorijska oblast) se ponekad logički na dalje može podeliti na prostor gde se čuvaju programske

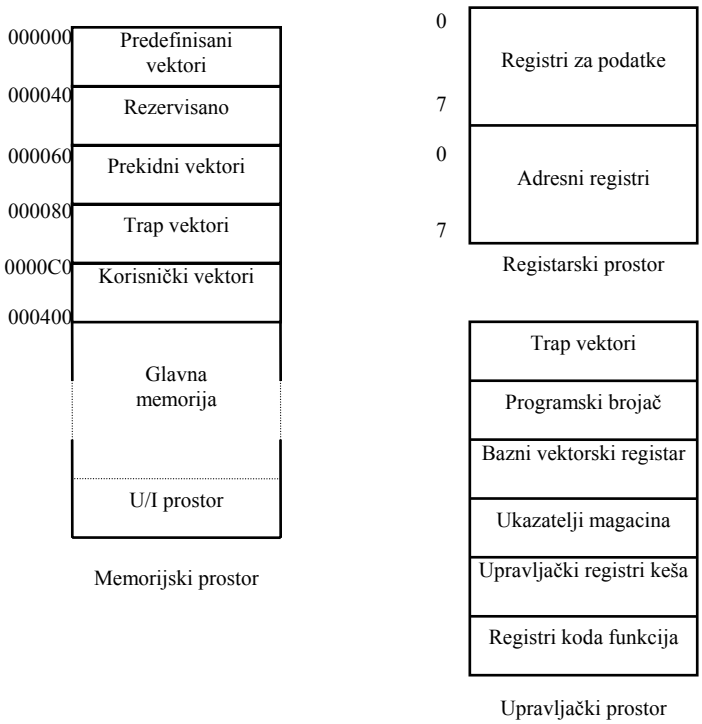
konstante; prostor za globalne podatke; prostor za lokalne podatke; prostor za prenos parametara između procedura itd.

3. **U/I prostor** - ovaj prostor se koristi za adresiranje periferala. Poželjno je uvesti ovaj prostor u arhitekturu kada U/I instrukcije zahtevaju različitu sinhronizaciju rada u odnosu na rad memorije, i kada se U/I ne poklapa ni sa jednim od pomenutih memorijskih prostora.
4. **Prostor magacina** - najveći broj arhitektura podržava mehanizam magacina na direktan način. To znači da je u magacinu moguće adresirati elemente podataka.
5. **Upravljački prostor** - ovaj prostor sadrži, na primer, reč stanja procesora (PSW) ili statusni registar (SR), registre za upravljanje memorijom, registre za upravljanje keš memorijom itd. U ovom prostoru radi obrade prekida i trapova sepecificiraju se i vektori.

Svi ovi prostori mogu egzistirati izdvojeno, kako logički tako i fizički:

- **Logički izdvojeni prostori** - ovo ukazuje da skup instrukcija mora biti sposoban da specificira korišćeni prostor; na primer, prostor magacina za PUSH instrukciju ili U/I prostor za instrukciju OUT. Zbog toga kažemo da su svi logički izdvojeni prostori vidljivi na arhitekturnom nivou.
- **Fizički izdvojeni prostori** - implementiraju se posebno pomoću specijalnog hardvera, a to se izvodi sa ciljem da se poboljšaju performanse; na primer, registarski prostor se često implementira na ovaj način, jer on mora biti takav da mu se pristupa brzo. Ovo je na arhitekturnom nivou transparentno, a vidljivo je samo na implementacionom nivou.

Tehnika preklapanja (*overlay*) omogućava preklapanje prostora. Na primer, magacin se može implementirati kao deo memorijskog prostora koji uslovljava da logički magacin i memorijski prostori dele isti fizički prostor.



Sl. 5.5. Logički adresni prostor kod MC68020.

Specifikacija željenog logičkog prostora se izvodi preko jednog od metoda (koje smo već razmatrali): u opkod prostoru (preko specijalnih instrukcija), ili preko operandskog prostora (specifikacijom registra, magacina ili memorije). Korišćenjem preklapanja, koje se ponekad zove *preslikavanje memorije* (memory mapping), jer se preslikavanja izvode u prostoru glavne memorije, moguće je, na primer, da se izvrši preklapanje prostora glavne memorije i U/I-a.

Na slici 5.5. za MC68020 prikazan je logički adresni prostor, koji se postavlja u tri, fizički izdvojena, prostora: registarski, upravljački i memorijski.

Danas se, skoro kod svih arhitektura, registri dodeljuju logički izdvojenim prostorima. Logičko preklapanje registara sa memorijom nije poželjno, jer je memorijski adresni prostor veći (zahteva veći broj bitova pa

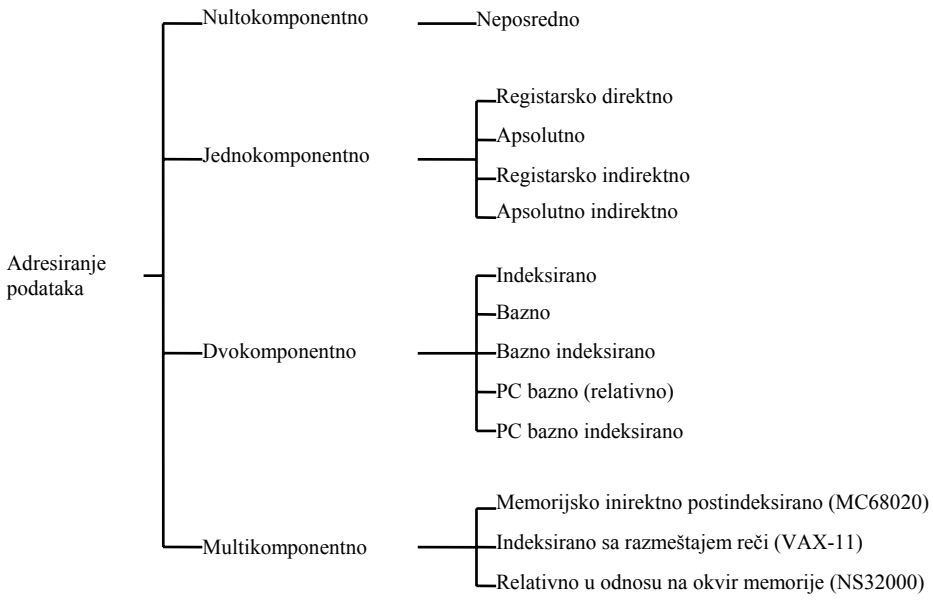
zbog toga i duže vreme dekodiranja) i zbog toga što je potrebno ugraditi dodatni hardver, da bi se realizovao mehanizam preklapanja.

Svi ostali logički prostori kod MC68020 formiraju logičko preklapanje sa memorijskim prostorom. Sa izuzetkom upravljačkog i registarskog prostora, svi prostori se takođe fizički implemetiraju kao deo prostora glavne memorije. Granice U/I prostora u memoriji, kada je u pitanju arhitektura MC68020, nisu fiksirane ali se implementiraju u zavisnosti od uslova u kojima radi i tipa aplikacije koji se izvodi.

Prednost logičkog preklapanja sastoji se u tome što nema potrebe za posebnim instrukcijama (opkodovima ili operandima) pomoću kojih se ostvaruje komunikacija između prostora, tako da je moć ukupnog skupa naredbi svima dostupna. Ovo čini da je arhitektura jednostavna i dobro organizovana. No, ne smemo zaboraviti da se preklapanje izvodi na račun prostora koji se "krade" od glavne memorije. Na slici 5.5 prikazano je da ova preklapanja zauzimaju ograničeni memorijski prostor. Kod arhitektura koje imaju veliki adresni prostor (32-bitno adresiranje) tehnika preklapanja se obavezno koristi.

5.2. Adresni načini rada

O načinu predstavljanja skalara i strukturnih tipova podataka smo već govorili. Analizirajmo sada kako se objekti skalarnog podatka mogu adresirati kada se oni deklarišu kao izdvojeni ili kada su oni deo strukturnog tipa podataka. Metod adresiranja podataka se može klasifikovati u saglasnosti sa brojem adresnih komponenata potrebnih za specifikaciju vrednosti operanada ili lokacije operanada (slika 5.6).



Sl. 5.6. Klasifikacija adresnih načina rada.

Multikomponentne adresne klase sadrže sve načine rada koji se mogu formirati od nulto-, jedno- ili dvokomponentnih adresnih klasa. Nije korisno (niti moguće) da se analiziraju sve nego samo nekoliko od ovih klasa. U Tabeli 5.2. prikazani su adresni načini rada za MC68020 koje ćemo u daljem tekstu detaljnije analizirati.

Tab. 5.2. Adresni načini rada kod MC68020.

Mod	Registar	Naziv kod MC68020	Sintaksa	Izračunavanje efektivne adrese
0	reg#	Direktno registarsko za podatke	Dn	ea = Dn
1	reg#	Direktno adresno registarsko	An	ea = An
2	reg#	Indirektno adresno registarsko	(An)	ea = (An)
3	reg#	Indirektno adresno registarsko sa postinkrementom	(An)+	ea = (An), An = (An) + N
4	reg#	Indirektno adresno registarsko sa predekrementom	-(An)	An = An - N, ea = (An)
5	reg#	Indirektno adresno registarsko sa razmeštajem	(d_{16},An)	ea = d_{16} + (An)
6	reg#	Indirektno adresno registarsko 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) + bd + (Xn)]) + od
7	0	Apsolutno kratko	xxx.W	ea = (sledeća reč)
7	1	Apsolutno dugo	xxx.L	ea = (sledeće 2 reči)
7	2	PC relativno	(d_{16},PC)	ea = d_{16} + (PC)
7	3	PC indeksirano	(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 pretindeksiranjem	([bd,PC,Xn],od)	ea = (M[(PC) + bd + (Xn)]) + od
7	4	Neposredno	#xxx	podatak = (sledeća reč (reči))

Napomena: Xn označava kompletan indeksni operand Xn.SIZE*SCALE, gde je Xn neki od adresnih registara ili registara za podatke, SIZE specificira obim indeksa (reč ili duga reč) a SCALE omgućava množenje indeksnog registra sa 1 (nema umnožavanja), 2, 4 ili 8.

5.2.1. Nultokomponentni adresni način rada

Ovaj način rada se zove neposredno adresiranje (immediate addressing) i predstavlja specijalni oblik adresiranja, jer se ne koriste adresne komponente. Vrednost operanada je direktno specificirana u instrukcionom nizu. Važnost neposrednog adresiranja se može sagledati na osnovu frekventnosti korišćenja (ranije smo ukazali da konstante čine 40% svih operanada). Najveći broj konstanti u programu se specificira korišćenjem neposrednog adresiranja, što ima prednost, jer su konstante obično mali brojevi (0 i 1 se javljaju najčešće). Shodno tome, efikasnije je specificirati konstantu umesto adrese memorijske lokacije. [ta više, za konstante nema potrebe da se menjaju, tako da se one mogu ugraditi u instrukcioni niz.

Postoje dva, u osnovi različita, mesta u kojima je moguće čuvati neposredne podatke:

1. Unutar osnovnog dela instrukcije: Ovo se često zove "short" ili "quick" neposredno adresiranje. Konstante mogu biti samo male (moraju se smestiti u okviru prve reči instrukcije ili bajta), ali one efikasno koriste memoriju.
2. Unutar proširenja instrukcije - Ovaj metod koristi konstante čija je dužina jednaka ili multipl osnovne dužine instrukcije. Dužina neposrednog podatka se specificira unutar osnovnog dela instrukcije, a neposredni podatak sledi osnovni deo (na osnovu ovoga se izvodi termin neposredno adresiranje).

Primer 5.2:

Kod MC68020 moguće je specificirati neposredni podatak u opkodu ili u prostoru operanada.

(a) Opkod prostor - postoji nekoliko opkodova koji ukazuju da je jedan operand neposredni podatak, a drugi operand se specificira na standardni način. Dalja podela se može učiniti u saglasnosti sa mestom na koje se neposredni podatak smešta:

- U osnovnom instrukcionom delu - instrukcije kao što su ADDQ i MOVEQ mogu rezervisati prostor u okviru osnovnog instrukcionog dela za specifikaciju malih konstanti: na primer,
  ADDQ: $1 \le C \le 8$
  MOVEQ: $-128 \le C \le 127$.
- U reči proširenja - druge instrukcije koje koriste neposredni način adresiranja smeštaju neposredni podatak u jednu (za podatak tipa bajt ili reč) ili dve (za dugu reč) reči proširenja. Instrukcije ovog tipa su, na primer ANDI (AND Immediate), CMPI i STOP.

(b) Prostor operanada - instrukcije koje imaju jedan ili veći broj opštih načina adresiranja mogu koristiti izvorni operand za specifikaciju neposrednog adresiranja (Tabela 5.2, mod 7.4).

Neposredni podaci se smeštaju u jednu ili dve reči proširenja. Primeri su

```
ASL.B     #2,D0;
```

```
BTST.L      #3,D1;
MULS.W      #12,D5.
```

5.2.2. Jednokomponentni adresni način rada

Ovi jednostavni načini rada specificiraju efektivnu adresu operanda koristeći se jednom komponentom u adresnom specifikatoru. Ova komponenta može biti broj registra ili apsolutna adresa operanda. Obe komponente se mogu direktno ili indirektno koristiti.

Registarsko direktno adresiranje

Jedan od najčešće korišćenih adresnih načina rada za pristup operandu je registarski direktni način rada. U ovom načinu rada, broj registra na jedinstveni način identifikuje registar koji sadrži vrednost operanda koju smo specificirali. Ovaj način rada se često koristi pre svega zbog efikasnosti i brzine. Pošto je u okviru CPU-a samo nekoliko registara dostupno (tipično 8 ili 16), može se postići kompaktna specifikacija operanada koja koristi 3 ili 4 bita. [ta više, pošto je registarsko polje normalno ugrađeno u čipu, vreme pristupa ovom polju u odnosu na glavnu i keš memoriju je znatno kraće. Sve savremene arhitekture imaju više od jenog registra koji podržava ovaj način adresiranja. Kod arhitektura sa samo jednim registrom, ovaj registar se često specificira implicitno.

Kod arhitekture MC68020 registarski skup je podeljen na dve grupe:

(a) registri za podatke - specificirani preko mod=0 u Tabeli 5.2, prvenstveno se koriste kao izvorište ili odredište za aritmetičko/logičke operacije.
(b) adresni registri - specificiraju se preko mod=1 u Tabeli 5.2, prvenstveno se koriste za specifikaciju adresa operanada.

Apsolutno adresiranje

Kod ovog načina adresiranja, adresa sespecificira samom naredbom, tj. ne postoji potreba za adresnim izračunavanjima. Apsolutno adresiranje se koristi kod mnogih arhitektura za specifikaciju vektora, U/I statusnih registarskih adresa i dr.

Primer 5.3:

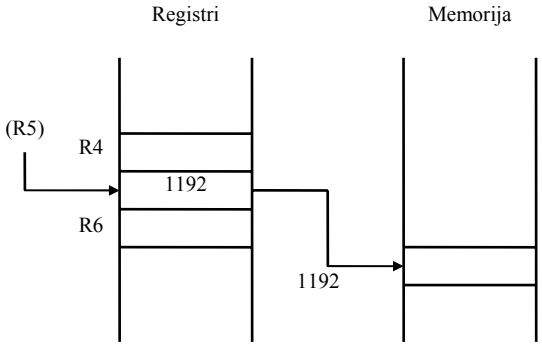
MC68020 podržava dva načina rada apsolutnog adresiranja. Adrese mogu biti kratke (16 bitne, mod 7.0 Tabela 5.2) ili dugačke (32-bitne, mode 7.1, Tab. 5.2). Kratke adrese se smeštaju u jednoj reči proširenja i znakovno se proširuju da bi se formirala 32-bitna adresa (ovo omogućava adresiranje dva memorijska bloka veličine 32 kB od kojih je jedan na kraju (top) a drugi na početku (bottom) fizičke memorije), a duga adresa je smeštena u dve reči proširenja, koje se povezuju sa ciljem da se formira kompletna adresa. Primeri su:

```
MOVE.L      0x80B932C6.L,D0     ; 32-bitna (.L) apsolutna adresa
ADD.W       0xFE30.W,D7         ; 16-bitna (.W) apsolutna adresa
ASL.B       0xD9CD.W            ; 16-bitna (.W) apsolutna adresa
```

Indirektno adresiranje

Indirekcija znači da specificirana lokacija, umesto da sadrži vrednost operanda, sadrži adresu vrednosti operanda (efektivna adresa). Registarsko indirektno adresiranje se veoma često koristi. Ono je korisno (i neophodno) kada adrese strukture podataka nisu poznate sve dok se program ne izvršava, jer se adrese mogu programski izračunavati i smeštati u registar, nakon čega se operandu može pristupiti preko ovog registra. Na slici 5.7 prikazan je način kako se izvodi registarsko indirektno adresiranje. Koristi se mod (R5), jer se u R5 nalazi memorijska adresa 1192. Notacija za ovaj oblik adresiranja je:

memorijska adresa=M[(R5)].



Sl. 5.7. Registarsko indirektno adresiranje.

Indirektno adresiranje sa registarskom modifikacijom

U principu, nekoliko varijacija osnovnog registarskog indirektnog adresiranja je moguće. Veći broj arhitektura podržava adresne načine rada kod kojih se registar koji sadrži adresu operanda modifikuje pre ili nakon korišćenja. Nekoliko načina je moguće, u zavisnosti od smera modifikacije (inkrementiranje ili dekrementiranje) i vremenskog trenutka modifikacije (pre- ili postmodifikacija). Objasnićemo predekrementirajući adresni način rada.

Kod predekrementirajućeg adresnog načina rada, registar koji sadrži adresu se dekrementira pre korišćenja, tako da on ukazuje na naredni (ili prethodni) element podatka. Iznos dekrementiranja zavisi od obima podataka. Adresni način rada sa predekremetiranjem se često koristi u kombinaciji sa adresnim načinom rada postinkrementiranje. Sa ova dva načina rada moguće je implementirati magacin preko pokazivača magacina (registar koji ukazuje na element koji se nalazi na vrhu magacina). Ako se element izbavlja iz magacina, pokazivač magacina mora da se inkrementira da bi ukazao na naredni element (usvajamo da magacin u memoriji raste naniže, startujući od najviše adrese) koristeći adresni način rada sa postinkremetiranjem. Element se može smestiti u magacin preko predekrementirajućeg adresnog načina rada (prvo se dekrementira pokazivač magacina na adresu naredne slobodne lokacije, a zatim se on koristi kao pokazivač na adresu operanda).

Primer:

MC68020 podržava različite oblike indirektnog načina adresiranja:

Osnovno adresno registarsko indirektno (mod=2, Tabela 5.2), koje se može kombinovati sa predekrementiranjem ili postinkrementiranjem (mod=4 i mod=3, Tabela 5.2, respektivno). Primeri nekih instrukcija koje koriste indirektno adresiranje su:

```
AND.L     #4,(A4)      ; registarsko indirektno
DIVS.W    (A3)+,D0     ; postinkrementiranje, A3 se inkrementira za 2 (podatak tipa
                       ; reč) pa se posle toga koristi
```

Primeri operacije sa magacinom koje koriste adresne načine rada sa predekrementiranjem, indirektno i postinkrementiranje su:

```
MOVE.W    X,-(A1)      ; Push X u magacin na koji pokazuje A1
MOVE.W    Y,-(A1)      ; Push Y u magacin na koji pokazuje A1
ADD.W     (A1)+,(A1)   ; Saberi X i Y i smesti rezultat u magacin
MOVE.W    (A1)+,Z      ; Pop rezultat iz magacina na koji ukazuje A1 i smesti ga u Z
```

5.2.3. Dvokomponentni adresni način rada

Moguće je kombinovati dve komponente da bi formirali dvokomponentne adresne načine rada. Jedna komponenta se obično zove baza, a druga razmeštaj (displacement). Obično, razmeštaj (ili ofset) predstavlja rastojanje između bazne adrese i adrese operanada. Bazna adresa i razmeštaj se mogu fiksirati (biti poznati u toku kompilacije) ili da se menjaju (izračunavaju u toku izvršenja programa). Kombinacija fiksne bazne adrese sa fiksnim razmeštajem "nije interesantna" (predstavlja oblik apsolutnog adresiranja što smo već analizirali). Za dalju analizu, kao što je prikazano na slici 5.8, ostaju još tri moguća adresna načina rada.

		Razmeštaj	
		Fiksni	Promenljivi
Bazno	Fiksni	---	Indeksiran
	Promenljivi	Bazno	Bazno indeksirano

Sl. 5.8.

Indeksno adresiranje

Kod ovog adresiranja, fiksna bazna adresa se specificira u instrukcionoj povorci (uporedljiva sa apsolutnom adresom). Da bi se formirala adresa operanda, promenljivi pomeraj (offset), koji se čuva u registru nazvanom indeks, se sabira sa baznom adresom. Ovaj način adresiranja se koristi kada je bazna adresa strukture podataka, kao što je vektor, poznata u toku kompilacije, ali se tačna pozicija elementa može odrediti samo u toku izvršenja programa. Da bi prilagodili obim operanda na nekoliko dužina, indeks se često množi faktorom umnožavanja. Ako vektor A[0..100] ima elemente koji zauzimaju po četiri bajta, faktor skaliranja (umnožavanja) je četiri, usvajajući da je adresna rezolucija jedan bajt. Ako bazna adresa vektora A[0..100] iznosi 1000, tada se element A[3] nalazi na adresi 1000+3*4=1012. Nivo indirekcije kod ovog adresiranja je takođe moguć. Postoje dve varijante indirektnog načina: indirektno sa preindeksiranjem i indirektno sa postindeksiranjem. Ako M[X] označava "sadržaj memorijske lokacije X", formule za izračunavanje adrese operanada su:

Pre_indexed: address=M[base+index]
Post_indexed: address=(M[base])+index

Kod preindeksiranja, ofset se dodaje baznoj adresi i generiše međuadresa. Sadržaj ove međuadrese sa nakon toga koristi kao adresa operanda. Kod postindeksiranja, bazna adresa se koristi kao međuadresa. Nakon toga dodaje se ofset sadržaju međuadrese da bi se dobila adresa operanda.

Bazno adresiranje

Ovo adresiranje je suprotno indeksnom adresiranju po tome što instrukcija sadrži fiksni razmeštaj a promenljiva bazna adresa se nalazi u registru. Glavna razlika je u tome što bazna adresa kod indeksnog adresiranja mora uvek biti potpuna adresa (full-lenght address), dok razmeštaj kod baznog adresiranja može biti kratki ili dugački. Ako ofset i bazna adresa imaju istu dužinu, ne može se praviti razlika između baznog i indeksnog adresiranja.

Bazno adresiranje se koristi kada bazna adresa bloka podataka nije poznata u toku izvršenja programa, a relativna pozicija elementa (stavke) u toku kompilacije je poznata. Neke moguće bazne adrese su:

- Početna adresa zapisa koji se aktivira (poziva): to je početna adresa memorijskog bloka koji se alocira posle aktiviranja programskog bloka, procedura ili funkcija. Za pristup lokalnim varijablama i parametrima može se koristiti fiksni razmeštaj.
- Početna adresa parametarskog bloka ili zapisa: to je pokazivač koji pokazuje na početak skupa parametara ili zapisa. Ponovo, fiksni razmeštaj se koristi za pristup adresi željenog parametra ili elementa zapisa.
- Specifični pokazivač - ovo je važno kod konstrukcija i adresiranja dinamičkih struktura, kao što su povezane liste i stabla.

Bazno indeksno adresiranje

Kod ovog adresiranja, kako bazna adresa tako i ofset su promenljivi i obe vrednosti se izračunavaju u toku izvršenja programa. Ovo omogućava pristup proizvoljnom elementu u strukturi podataka sa početnom adresom koja nije poznata u toku kompilacije.

PC-relativno adresiranje

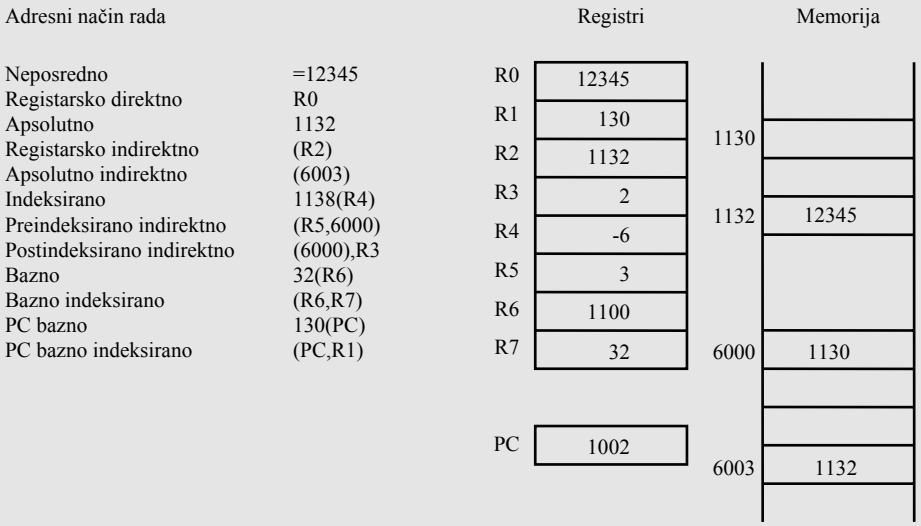
Specijalni slučaj dvokomponentnog adresiranja je ono koje koristi prigramski brojač (PC) kao registar u kome se nalazi bazna adresa. Kod arhitektura gde je PC jedan od registara opšte namene, standardno registarsko adresiranje se može koristiti za pristup podacima koji se nalaze u prostoru gde je smešten program. Na primer, VAX-11 može koristi mod (R15)+ za pristup neposrednim operandima, jer je R15 programski brojač. Ali, ako je PC specijalan registar, mora da postoji dodatni adresni način rada pomoću koga će biti moguće adresirati podatke relativno, u odnosu na tekuću vrednost PC-a. Kako je ova vrednost (bazna adresa) promenljiva, razlikujemo dva moguća adresna načina rada: "PC based" i "PC based indexed" (slika 5.8). "PC based" adresni način rada se često zove *relativno adresiranje* i standardno može koristiti promenljivi obim razmeštaja. Imajući u vidu da je najveći broj podataka i programskih stavki blizu mesta odakle se njima vrši obraćanje, kratak (jednobajtni) razmeštaj je

često dovoljan. "PC based indexed" ili *relativno indeksno adresiranje* ima efekat kao i standardno bazno adresiranje, sa izuzetkom što se PC koristi kao bazni registar.

Najčešće se PC relativno adresiranje koristi kod instrukcija grananja koje su esencijalne za implementaciju HLL upravljačkih struktura. Korišćenje relativnog umesto apsolutnog adresiranja automatski obezbeđuje relokatibilnost programskih modula koji se mogu smeštati bilo gde u memoriji (oni su poziciono nezavisni). Adresiranje podataka, relativno u odnosu na programski kod, je takođe važno kada su "read-only" podaci obavezni deo programa. Ovi podaci se mogu puniti istovremeno kao i kod i sastoje se od translacionih tabela (EBCIDIC i ASCII) ili tabela sa inicijalnim vrednostima koje se koriste za iterativna izračunavanja.

Primer 5.5:

Na slici 5.9 prikazana je specifikacija vrednosti operanda 12345 (lociran u registru R0 i u memorijskoj lokaciji 1132) za nulto-, jedno- i dvokomponentni adresni način rada. Na primer, kod preindeksiranog indirektnog adresiranja, dodavanje 6000 sadržaju R5 (=3) daje vrednost (6003), gde je smeštena adresa operanda (1132).



Sl. 5.9.

5.2.4. Multikomponentni adresni način rada

Kao što je naglašeno, svi prethodni načini adresiranja se mogu kombinovati sa ciljem da formiraju tipove ezoteričnih (nejasnih) i nekih korisnih načina adresiranja. U detaljnu analizu ovih načina adresiranja se nećemo upuštati.

5.3. Adresiranje struktuiranih podataka

Većim brojem tipova podataka o kojima smo do sada govorili se može lako pristupiti koristeći se nulto- ili jednoadresnim načinima rada. Za strukturne tipove podataka, kao što su polja, zapisi i magacin, potreban je nešto složeniji adresni mehanizam. Ukažimo sada na adresiranje elemenata unutar ovih struktura.

5.3.1. Polja

Za adresiranje elemenata polja važni su sledeći aspekti:

- **provera indeksa** - indeks se mora nalaziti unutar unapred definisanih granica polja. Naglasimo da se ova provera ne mora uvek vršiti u toku izvršenja programa, jer je kompilator sposoban da oceni kada je indeks u okviru granica - na primer, kada je indeks konstanta.
- **umnožavanje** - predstavlja množenje indeksa veličinom submatrice ili tipom podatka koji se koristi. Na primer, ako se koristi 32-bitni FP broj kod bajt-adresibilne mešine, faktor umnožavanja je četiri.

- **izbor elementa** - izbor željenog vektor elementa se direktno podržava od strane većeg broja arhitektura pomoću, na primer, indeksnog adresiranja. Za izbor elemenata u polju, često postoje posebne instrukcije koje obavljaju indeksno izračunavanje.

Adresa elementa A[I,J] u polju A[1..M,1..N] se može izračunati na sledeći način (usvajamo da se svaki element smešta u samo jednu memorijsku lokaciju, ako ne treba uvesti skaliranje, i "row-major ordering" se koristi):

$$\begin{aligned} \text{addressA}[I,J] &= \text{addressA}[1,1]+(I-1)*N+(J-1) \\ &= \text{addressA}[1,1]+I*N+(J-1) \\ &= \text{constantA0}+I*N+J \end{aligned} \quad (1)$$

gde je A0=addressA[1,1]-N-1.

Adresno izračunavanje elementa u 2D polju zahteva jedno množenje i dva sabiranja (za slučaj da je A0 određeno u toku kompilacije). Nedostatak množenja (ova operacija traje dugo) se može eliminisati u velikom broju slučajeva, ako kompilatori zamene operaciju množenja repetitivnim sabiranjem ili oduzimanjem u okviru spoljne petlje. Ovo se zove "strength reduction", tj. zamena kompleksnih operacija sa jednostavnim u spoljnoj petlji, a prikazano je sledećim programskim fragmentom. Paskalska **for** petlja za inicijalizaciju osam kolona polja A se može prevesti na asemblerski jezik za MC68020 (koristeći jednakost 1), gde registar A0 sadrži konstantu A0, a D0 se koristi za čuvanje promenljive I.

```
var A:array [1..M,1..N] of integer;
for I:=M downto 1 do
        A[I,8]:=0;
```

Verzija na asemblerskom jeziku ima oblik:

```
        MOVE.L      #A0,A0          ; A0 sadrži početnu adresu A
        MOVE.L      #M,D0           ; D0 sadrži inicijalnu vrednost za I
Loop:   MOVE.W      D0,D1
        MULS.W      #N,D1           ; D1:=I*N
        CLR.L       8(A0,D1.W)      ; J8, a address=A0+I*N+8
        SUBQ.W      #1,D0           ; I=I-1
        BNE Loop                    ; sve dok (until) I=0
```

Kada se množenje zameni repetitivnim oduzimanjem, rezultat je sledeći (A0 ponovo sadrži konstantu A0 iz jednakosti (1), a D0 se sada koristi kao indeks registar):

```
        MOVE.L      #M*N,D0         ; D0:=I*N
Loop:   CLR.L       8(A0,D0.W)      ; address=A0+I*N+8
        SUB.W       #N,D0           ; I=I-1
        BNE Loop                    ; until I=O
```

Analiziraćemo oblik adrese elementa A[S1,S2,...SN] iz polja A[L1..U1,...,LN..UN], pri čemu usvajamo da je A "row-major ordered". Da bi se pojednostavila računica, uvešćemo promenljive d_i (i=1..N-1), gde $d_i = U_{i+1} - L_{i+1} + 1$, tj. da di predstavlja broj indeksne vrednosti za dimenziju i+1. Vrednost d_n se može posmatrati kao veličina polja elemenata (=E adresibilnih jedinica), a početna adresa polja je A. Adresa elemenata polja se može sada izračunati na sledeći način:

- Za polje A[L..U], adresa elementa A[S] je:

$$\text{address}=A+(S-L)*E$$

- Za polje $A[L_1..U_1,L_2..U_2]$, adresa elementa $A[S_1,S_2]$ je:

$$\text{address}=A+((S_1-L_1)*d+(S_2-L_2))*E \quad (2)$$

U opštem slučaju za polje $A[L_1..U_1,...,L_N..U_N]$, adresa elementa $A[S_1,S_2,...,S_N]$ je:

$$\text{address}=A+(...((S_1-L_1)*d_1+(S_2-L_2))*d_2+...+(S_N-L_N))*d_N$$

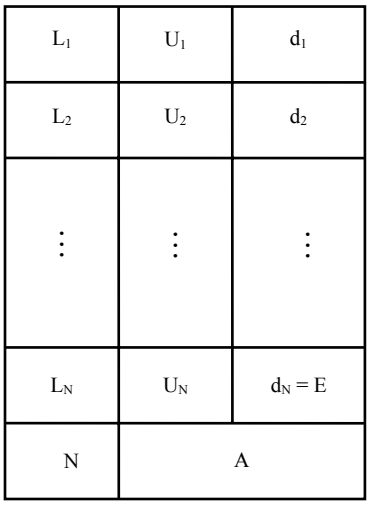
U skladu sa jednačinom (2) mora se vršiti provera da bi bili sigurni da se svaki indeks nalazi u granicama njegovog opsega, tj. da li važi uslov $L_i \le S_i \le U_i$. Da bi pojednostavili izračunavanje, potrebno je iskoristiti deskriptor koga zovemo "dope vector". Za svaku dimenziju, vrednosti L_i, U_i i d_i se pamte. [ta više, broj dimenzije N, početna adesa A i veličina elelmenta E se pamte (obim elementa se pamti na mesto d_N). Obim elementa se može smatrati konačnim multiplikacionim faktorom da bi se ostvarilo skaliranje, koje je zavisno od obima elementa.

Koristeći "dope vector" (slika 5.11), adresa elementa se može sada dobiti na sledeći način:

```
address:=0;
for I:=1 to N do
        begin
                if (S[I]<L[I] or (S[I]>U[I]) then trap;
                address:=(address+S[I]-L[I])*d[I];
        end;
address:=address+A;
```

"Dope vector" može takođe da sadrži informaciju kao što je tip podatka elementa i/ili ukupnu veličinu polja (mada je ovo redundantno, jer se to može izračunati na osnovu drugih dostupnih podataka). Početna adresa A se može zameniti adresom A_v = A[0,0,...,0], koja predstavlja virtuelnu početnu adresu polja A. Korišćenjem virtuelnog početka pojednostavljuje se adresno izračunavanje, jer ne postoji operacija oduzimanja da bi se dobila donja granica. U tom slučaju jednačina (2) dobija sledeći oblik:

$$address = A_v + (...(S_1*d_1+S_2)*d_2+...+S_N)*d_N \quad (3)$$



Sl. 5.11. "Dope vector" za *N*-dimenzionalno polje.

Dodela memorije polju ima veliki uticaj u zavisnosti od toga da li se granice polja mogu ili ne mogu menjati u toku izvršenja programa. Statičke granice ukazuju da sve vrednosti za L_i i U_i moraju biti poznate u toku procesa kompilacije. FORTRAN i Pascal koriste statičke granice sa ciljem da olakšaju alokaciju polja ili (vektora), tako da se ukupna veličina može odrediti u toku procesa kompilacije. Kod FORTRAN-a, početna adresa polja se može usvojiti da je konstantna u toku izvršenja programa, dok kod Pascala ofset početne adrese polja u odnosu na start aktivacionog zapisa je konstantan. Dinamičke granice se koriste kod jezika kao što je ALGOL i neki od dijalekata Pascala. Ovi jezici ne zahtevaju da sve (ili bilo koje) vrednosti L_i i U_i budu poznate u toku kompilacije. Ovo ukazuje da se ukupna veličina može odrediti jedino u toku izvršenja, a da se jedino do tog trenutka obezbedi rezervacija memorijskog prostora.

Adresiranje zasnovano na deskriptorima

Deskriptori se mogu koristiti za opisivanje strukture struktuiranih podataka. "Dope vector" sa slike 5.11. se može koristiti kao deskriptor N-dimenzionalnog polja. Arhitekture koje koriste deskriptore obično ne obezbeđuju (omogućavaju) takve kompleksnosti zbog toga što hardver koji ih podržava nije tako složen. Jedan deskriptor tipično opisuje samo jednu dimenziju polja.

5.3.2. Zapisi

Za adresiranje zapisa i elemenata zapisa, najvažniji aspekt je da sva polja imaju konstantni ofset, relativan u odnosu na početak zapisa. Elementima se može lako pristupati koristeći se adresiranjem tipa bazni razmeštaj. Kada su dozvoljeni različiti zapisi, polja mogu imati promenljivi ofset relativan u odnosu na početak zapisa. Da bi se adresiranje učinilo lakšim za kompilatore (i za one koji pišu kompilatore), jezici koji dozvoljavaju različite zapise često diktiraju da se prvo mora deklarisati fiksni deo, tako da polja u fiksnom delu i dalje imaju fiksni ofset, relativan u odnosu na početak zapisa (njima i dalje može pristupati koristeći se baznim adresiranjem).

5.3.3. Magacin

Magacin je važan kod HLL-ova kao memorijski medijum za smeštaj zapisa, prenos parametara između procedura, i čuvanje međurezultata izračunavanja. Kod magacinsko orijentisanih (stack-oriented) arhitektura sve (operacije sa izuzetkom Load i Store) implicitno koriste magacin. Instrukcija kao što je MUL prihvata oba operanda sa vrha magacina a zatim smešta rezultat ponovo u magacin. Pored standardnih aritmetičkih instrukcija, magacinsko-orijentisane mašine često koriste specijalne instrukcije kao što je SWAP (promeni uzajamno sadržaj dva elementa sa vrha magacina), RSUB (obrnuto oduzimanje, ako standardna SUB oduzima drugi element od elementa koji je na vrhu magacina, tada RSUB oduzima element koji je na vrhu magacina od drugog). Druge instrukcije koje se standardno koriste su JSR i RTS. JSR pre nego što obavi skok smešta prvo stanje programskog brojača u magacin. RTS izbavlja povratnu adresu iz magacina i smešta je u programski brojač. Kod ostalih arhitektura, potrebno je eksplicitno specificirati da operacija koristi operand iz magacina. Ako je to specificirano u OPCODE delu, tada su to instrukcije PUSH i POP, pomoću kojih se smeštaju i izbavljaju podaci u/iz magacina, respektivno.

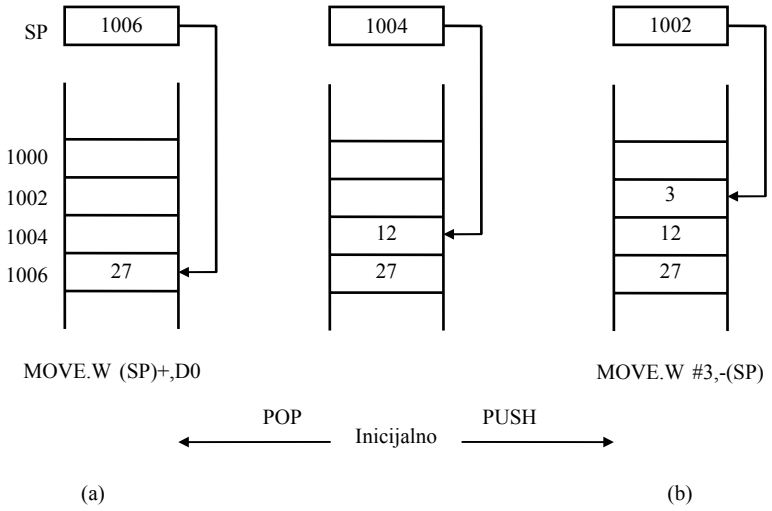
Podrazumeva se da se jedan od operandskih specifikatora tako kodira da predstavlja element sa vrha magacina (TOS). Implementacija ovog načina adresiranja mora automatski da obavi funkciju koja je u uskoj vezi sa radom magacina kao drugi zadatak. Na primer arhitektura NS32000 sadrži jedan takav specifikator operanda, tako da su instrukcije oblika:

```
MOV   A,TOS      ; Push A u magacin
MOV   TOS,A      ; Pop A
ADD   TOS,TOS    ; Saberi dva elementa sa vrha magacina
ADD   Src,TOS    ; Saberi Src sa vrhom magacina
ADD   TOS,Dst    ; Saberi vrh magacina sa Dst
```

Ako je TOS jedini izvorni operand, pokazivač magacina (SP) se automatski ažurira posle obavljene operacije (inkrementira za obim operanda). Kada se TOS operand javlja samo kao odredišni operand, rezultat se smešta u magacin na taj način što se prvo dekrementira SP (za dužinu operanda) a zatim smešta rezultat. Ako se TOS operand koristi kako za izvorište tako i za odredište, SP ostaje nepromenjen a koristi se kao adresa operanda (operand je elemenat sa vrha magacina).

Veliki broj arhitektura ne podržava direktno TOS operand, ali mnoge imaju registarsko-indirektni način adresiranja sa proširenjem tipa postinkrementiranje ili predekremetiranje, koje su pogodne za implementaciju magacinske memorije. I pored toga što je svim registrima dozvoljeno da funkcionišu kao SP, najčešće se samo jedan registar specijalno predodređuje da bude SP (registar A7 kod MC68020).

Korišćenjem načina adresiranja sa postinkrementiranjem i predekrementiranjem obezbeđuje se implementacija dva tip magacina: jedan kod koga SP pokazuje na elemenat sa vrha magacina (magacin se povećava prema numerički nižim memorijskim lokacijama), a drugi sa SP koji pokazuje prvu slobodnu lokaciju (ovaj magacin se povećava naviše). Prva implementacija (slika 5.14) se najčešće koristi, s obzirom da je standardnije da se ukazuje na nešto (vrednost) a ne na slobodnu lokaciju (nema vrednosti).



Sl. 5.14.

Filozofski je problem implementiranje praznog magacina (gde mora SP da ukazuje?). Prva implementacija je efikasnija kada se operandi čitaju i upisuju u toku iste operacije (kod ovog načina, SP ostaje nepromenjen). Da bi

smestili vrednost u magacin, koristi se adresiranje tipa predekrementiranje (slika 5.14a), dok kod izbavljanja vrednosti iz magacina se koristi postinkrementiranje (slika 5.14b). Sa ovakvom implementacijom prethodno pomenute instrukcije, koje koriste TOS operande, imaće sledeći oblik:

```
MOV   A,-(SP)      ; Push A u magacin
MOV   (SP)+,A      ; Pop A
ADD   (SP)+,(SP)   ; Saberi dva elementa sa vrha magacina
ADD   Src,(SP)     ; Saberi Src sa vrhom magacina
ADD   (SP),Dst     ; Saberi vrh magacina sa Dst
```

5.4. Kompaktna specifikacija operanada

Specifikacija operanada se često javlja. U Tabeli 5.3 prikazana je frekventna distribucija broja operanada po instrukciji za mašinu VAX-11, koja ukazuje da se u proseku koristi 1,8 operand po instrukciji. Zbog ovoga, kompaktna specifikacija operanada je neophodna sa ciljem da se postigne ušteda memorijskog prostora i propusnosti memorije. Postoji nekoliko tehnika koje su dostupne za specifikaciju operanada.

Tab. 5.3. Raspodela frekvencije (%) broja operanada po instrukciji kod VAX-11.

Broj operanada	Frekvencija
0	3.3
1	33.1
2	52.4
3	7.2
4	3.6
5	0.2
6	0.2

5.4.1. Implicitni operandi

Ovaj metod je često bio korišćen kod starijih arhitektura, kod kojih je bio dostupan samo jedan registar (tzv. akumulator). Zbog toga ove arhitekture koriste jednoadresni format, pri čemu je akumulator uvek drugi operand. Sa povećanjem broja registara opšte namene, ovaj metod se nije više koristio. Arhitekture zasnovane na magacinu još koriste ovaj metod, zbog toga što samo LOAD i STORE operacije zahtevaju eksplicitne operande, ali sve aritmetičke i logičke operacije implicitno koriste magacin kao izvorni i/ili odredišni operand.

5.4.2. Registarski operandi

One promenljive koje se često koriste (ili pokazivači na promenljive) čuvaju se u registrima. Simulacije su pokazale da su 32-bitni registri skoro uvek dovoljni. Da bi smanjili broj potrebnih bitova za specifikaciju registara, najveći broj registarsko-orijentisanih arhitektura ima 8 ili 16 registara. Direktni ili indirektni registarski operandi se zatim mogu specificirati na kompaktan način, a to zahteva tri do četiri bita.

5.4.3. Relativno adresiranje

Ovaj metod specifikacije se odnosi na onaj slučaj kada su operandi jedan za drugim. Ovakav zaključak važi kada su u pitanju zapisi, gde su lokalne promenljive alocirane u neprekidnim memorijskim blokovima. Zbog ove neprekidnosti, obično postoji mali razmeštaj (reda bajt) kod najvećeg broja slučajeva. U Tabeli 5.4 prikazano je da se bajt razmeštaj često koristi kako za adresiranje podataka tako i instrukcija.

Tab. 5.4. Frekvencija specifikatora operanda (%) kod VAX-11.

Adresni način rada	BASIC	COBOL	FORTRAN	Pascal
0-3 Literal	9.8	18.1	18.2	15.5
4 Indeks	4.7	2.1	5.6	7.6
5 Registarsko	34.9	37.9	41.0	45.1
6 Registarsko odloženo	9.0	15.3	9.1	2.0
7 Autodekrementno	0.5	1.0	0.5	0.7
8 Autoinkrementno	17.4	4.8	4.1	2.6
9 Autoinkrementno odloženo	0.1	0.2	0.3	0.0
10 Bajtovski razmeštaj	16.3	10.8	13.6	16.6
11 Odloženo sa bajtovskim razmeštajem	0.5	1.3	0.6	0.4
12 Reč razmeštaja	0.8	0.9	1.5	6.5
13 Odloženo sa razmeštajem dužine reč	-	-	0.0	-
14 Razmeštaj dužine duga reč	6.1	7.3	5.5	3.0
15 Odloženo sa razmeštajem dužine duga reč	-	0.3	0.0	-

Iz Tabele 5.5 vidi se da je najveći broj razmeštaja kod instrukcija grananja pozitivan (i da su rastojanja mala), jer se ove instrukcije uglavnom generišu kao rezultat **if**...**then**.(...**else**) iskaza. Iz Tabele 5.5 se vidi da se više od 90% od svih razmeštaja grananja može kodirati samo jednim bajtom.

Tab. 5.5. Frekvencija uslovnog grnanja sa razmeštajem (%) kod VAX-11.

Opseg bajtova	BASIC	COBOL	FORTRAN	Pascal
-128 do -17	0.4	0.5	4.5	0.0
-16 do -1	29.9	0.2	0.8	0.0
Bez grananja	51.6	43.6	34.4	49.7
1-4	3.1	13.1	13.0	11.0
5-8	1.6	7.8	14.1	2.7
9-16	5.9	11.0	11.1	10.2
17-32	3.5	14.3	8.7	10.5
33-64	1.5	5.5	6.1	4.0
65-127	1.1	1.7	2.3	3.5
Ostalo	1.4	2.3	5.0	8.4

5.4.4. Metod kružne analize (pooling)

Ponekad je moguće da kompilator skupi često korišćene operande koji su razređeni po memoriji na jedno mesto (pool). Adresiranje sa baznim razmeštajem (malo polje za razmeštaj) se tada uobičajeno koristi za specifikaciju operanada.

Specijalni oblik "Pooling"-a se koristi za procedure i funkcije. U ovom slučaju, "pool" čini tabela pokazivača na procedure i funkcije. Poziv specifične procedure ostvaruje se korišćenjem jednonivovske indirekciju, pa se na taj način smanjuje broj operandskih bitova na $\log_2(N)$, za slučaj da postoji N procedura. Da bi se pristupilo proceduri potreban je samo broj procedure.