4. MAŠINSKI JEZIK

4.1. Frekvencija korišćenja instrukcija

Trenutno je najveći broj programa je napisan na HLL, prvenstveno zbog veće programerske produktivnosti, detekcije grešaka u toku kompilacije i lakšeg održavanja programa. No, nezavisno od toga, mašinski jezik se još uvek koristi u svim onim situacijama gde je potrebno postići bolje performanse (brže izvršenje programa) i direktno upravljanje resursima računarskog sistema. Da bi bolje razumeli zahteve za realizaciju skupa naredbi, neophodno je prvo analizirati HLL-ove.

Za analizu instrukcija važno je praviti razliku između *statičke* i *dinamičke frekventne analize*. Kod statičke frekventne analize vrši se brojanje instrukcija, tj. vrši se analiza koliko puta se one pojavljuju u programu. Rezultat ove analize je važan zbog optimizacije korišćenja memorije. Dinamička frekventna analiza, sa druge strane, za svaku instrukciju daje broj koji ukazuje na to koliki se broj puta određena instrukcija izvršava u toku rada programa. Ova analiza je važna zbog smanjenja vremena izvršenja programa; procesor se tako projektuje da instrukcije koje se najčešće izvršavaju imaju najkraće moguće vreme izvršenja.

4.1.1. Statistike HLL-a

U Tabeli 4.1 prikazana je statička frekventna distribucija iskaza na FORTRAN-u (Knuth 1971) i Pascalu (Shimasaki 1980). Naglasimo da je kod FORTRAN-a konstrukcija IF () STATEMENT brojana kao i IF i kao iskaz pa je zbog toga ukupni procenat veći od 100%. Naglasimo da je kod oba jezika najčešće korišćen iskaz "dodela" (assigment). Kod Pascala na drugom mestu je call prvenstveno zbog modernijeg strukturnog, modularnog pristupa programiranju. Kod FORTRAN-a je CALL na četvrtom mestu, nakon IF i GOTO iskaza.

Tab. 4.1. Raspodela frekvencija (%) FORTRAN i Pascal iskaza.

Konstrukcija jezika	Raspodela
FORTRAN	
Dodela	41.0
IF	14.5
GOTO	13.0
CALL	8.0
CONTINUE	5.0
WRITE	4.0
FORMAT	4.0
DO	4.0
DATA	2.0
RETURN	2.0
DIMENSION	2.5
Ostalo	8.5
Ukupno	108.0
Pascal	
Dodela	37.2
call	31.6
if	19.2
with	3.4
procedure	2.6
var	1.4
repeat	1.3
while	1.1
for	0.7
case	0.7
goto	0.2
Ostalo	0.6
Ukupno	100.0

U Tabeli 4.2 prikazana je analiza frekventne distribucije iskaza assigment (Tanenbaum 1978). Ona ukazuje da je najveći broj iskaza tipa assignent prost. Kada se projektuje računar veoma je važno da se ima u vidu koje se instrukcije najčešće koriste, kako bi se ove mogle najefikasnije izvršavati.

U Tabeli 4.3 prikazana je analiza (Tanenbaum 1978) tipova operanada koji se koriste kod jezika SAL (sličan je Pascalu). Evidentno je da su konstante najvažnije, jer je njihov broj 40% od svih operanada.

Hennessy (1982) je sačinio frekventnu distribuciju konstanti, prikazanu u Tabeli 4.4, koje se sreću kod programa napisanih na Pascalu.

Ova analiza pokazuje da se konstante "0" i "1" koriste veoma često (kao na primer if A>0 then ... ili A:=A+1 itd.). Najveći broj procesora u svom skupu instrukcija ima instrukcije koje koriste neposredno adresiranje pomoću kojih se specificiraju male konstante kao direktni operandi (na primer MOVEQ kod MC68020 specificira 8-bitnu neposrednu vrednost; ispitivanja su pokazala da se ovom instrukcijom pokriva 95,5% svih iskaza tipa MOVE constant koji se javljaju).

Tab. 4.2. Raspodela frekvencije (%) za naredbu dodelu u FORTRANU.

Tip	Statički	Dinamički
Jednoelementni iskaz sa desne strane (A:=B)	80.0	66.3
Dvoelementni iskaz sa desne strane (A:=A op B, A:=B op C)	15.2	20.4
Ostalo	4.8	13.3

Tab. 4.3. Raspodela frekvencija (%) tipova operanada.

Tip	Statički	Dinamički
Konstante	40.0	32.8
Skalrne promenljive	35.6	41.9
Elementi polja	9.3	9.2
Polja strukture	7.1	11.1
Poziv funkcije	4.8	1.6
Bit poje	3.2	3.3

Tab. 4.4. Raspodela frekvencija konstanti.

Konstanta	Procenat	Kumulativni procenat
0	24.8	24.8
1	19.0	43.8
2	4.1	47.8
3-15	20.8	68.7
16-255	26.8	95.5
>255	4.5	100.0

4.1.2. Statistika mašinskog jezika

U Tabeli 4.5 prikazane su dinamička frekvencija i distribucija dinamičkog vremena izvršenja za instrukcije mikroprocesora MC68020 koje su generisane od strane aplikacionih programa napisanih na HLL a izvršavaju se pod operativnim sistemom UNIX (Mc Gregor 1985.). Instrukcija tipa MOVE se najviše koristi (odgovara HLL iskazu A:=B), zatim su Bcc (*Branch Conditionally*), DBcc (*Decrement and Branch Conditionally*) (Ove dve instrukcije se koriste za **if**, **repeat**, **while**, **for**, **case** i druge iskaze).

Tab. 4.5. Raspodela frekvencija (%) instrukcija mikroporcesora MC68000.

Instrukcija	Dinamička frekvencija	Dinamičko vreme izvršenja
MOVE	32.85	36.86
Bcc, DBcc	23.40	18.23
Aritmetičke/logičke	12.18	13.64
Compare	7.86	7.61
Shift/rotate	3.25	3.19
Test	3.09	2.92
Link/Unlink	2.76	3.15
Clear	2.69	3.93
Ostale	11.92	10.47
Ukupno	100.00	100.00

U Tabeli 4.6. prikazana je frekventna distribucija adresnih načina rada za MC68000 (McGregor 1985.). U ovoj tabeli je prikazana dinamička frekvencija načina rada koji se odnose na formiranje efektivne adrese (EA) za izvorni i odredišni operand. Najčešće korišćen adresni način rada za izvorni operand je bazno adresiranje (kod Motorole se zove bazni razmeštaj). Najčešće korišćen način adresiranja za odredišni operand je registarski direktni.

Tab. 4.6. Raspodela frekvencija (%) adresnih načina rada mikroporcesora MC68000.

Adresni način rada	Frekvencija izvorne EA	Frekvencija odredišne EA
Nijedan	35.10	17.28
Registarski direktni: Dn, An	12.50	49.48
Registarski indirektni: (An), (An)+, -(An)	8.24	14.50
Bazni razmeštaj (d8,An), (d8,An,Xn)	22.83	10.94
Apsolutni: xxx.W, xxx.L	0.54	0.00
PC relativni: (d8,PC), (D8,PC,Xn)	0.42	-
Neposredni: #xxx, Quick	7.77	-
Implicitno magacinski	3.08	4.93
Ukupno	90.48	97.13

4.2. Format mašinskih instrukcija

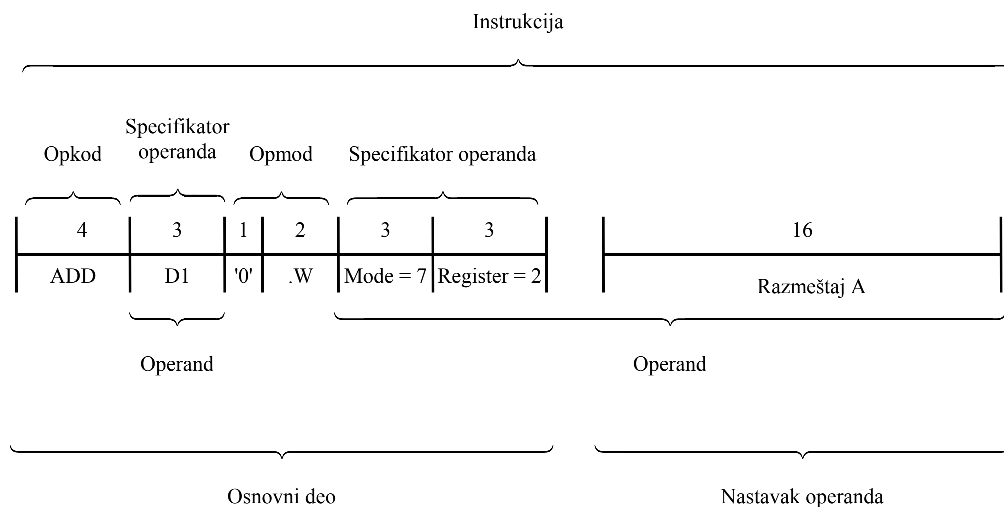
U odnosu na HLL iskaze, instrukcije koje se izvršavaju na hardverskom nivou ili mikroprogramski, obično imaju jednostavan format. Ovo se može objasniti na sledeći način:
Jednostavnost ukazuje na veću brzinu izvršenja, jer su putevi podataka manje komplikovani. Manje hardvera ukazuje na to da će biti: manje projektantskih grešaka; poboljšana pouzdanost; skraćeno vreme projektovanja.

- Analize HLL-a ukazuju da se jednostavniji iskazi češće koriste.
- Mašinske instrukcije visokog nivoa (složenije) ne mogu se jednostavno koristiti od strane HLL, jer se HLL jezici međusobno mnogo razlikuju.
- Kompleksne instrukcije čine kompilaciju veoma teškom zbog toga što se mnogo specijalnih slučajeva mora razmatrati.

Pre nego što produžimo sa daljom analizom neophodno je prvo upoznati se sa određenom terminologijom. Da bi objasnili ovu terminologiju, koristićemo instrukciju ADD koja pripada skupu naredbi mikroprocesora MC68020.

4.2.1. Terminologija

Instrukcija se sastoji od nekoliko delova koji zajedno specificiraju tip operacije i lokacije (ili vrednosti) operanada. Fraza "instrukcija" odnosi se na sve njene komponente. Na primer, na slici 4.1, celokupna instrukcija ima 32 bita, a sadrži i razmeštaj (displacement) A. Ako instrukcije imaju promenljive dužine, obično je prvi deo (fiksne dužine) taj koji specificira tip operacije i broj operanada (basic instruction part ili first word). Na slici 4.1 osnovni deo čini prvih 16 bitova; razmeštaj A nije deo osnovnog dela.



Sl. 4.1. Terminologija instrukcije: specifikacija instrukcije ADD.W A(PC),D1.

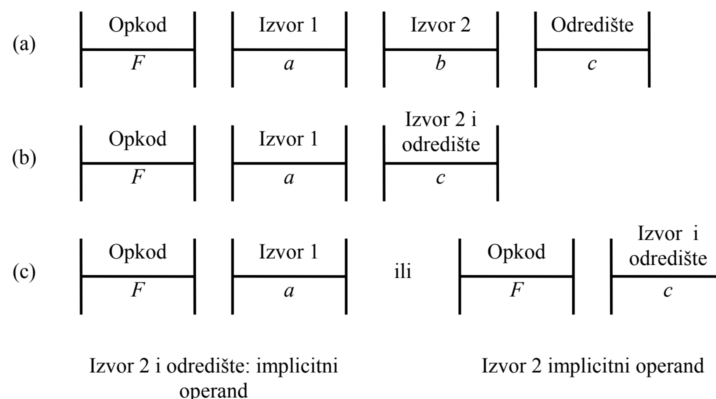
Operacija se obično specificira u polju opkod (*opcode*) koje je deo osnovnog polja. Kod računara kao što je MC68020, operacija se daje detaljnije specificira u polju *opmod* (*opmode*). Ovo polje specificira obim operanada (u ovom slučaju je .W, tj. reč) i smer prenosa podataka u operaciji (u ovom slučaju je "0" što znači da je prenos tipa memorija$\rightarrow$registar). Specifikaciju operanada čini nekoliko delova. Prvi deo je fiksne dužine i može specificirati: koji se adresni način rada koristi; koji se registar koristi; kolika je vrednost neposrednog podatka (konstanta male vrednosti). Ovaj deo se zove *specifikator operanda*. Specifikator operanda ponekad zahteva dodatne podatke (kao

što je razmeštaj) koji se smeštaju posle osnovnog dela. Ovaj podatak je poznat kao *reč proširenja operanda* (operand extension word). Specifikator operanda i proširenje operanda formiraju *operand*. Operand može biti adresa, specifikacija registra ili neposredni podatak. Ako operand specificira registar ili memorijsku lokaciju, on se zove *adresni operand* (address operand). Na kraju, vrednost neposrednog podatka ili vrednost koja se nalazi u specificiranoj memorijskoj lokaciji ili registru, zove se *vrednost operanda* (operand value).

4.2.2. Klasifikacija skupa instrukcija

Ukažimo na opšti oblik instrukcije koji je još na povoju računarstva bio korišćen od strane projektanata za izražavanje aritmetičkih operacija. Oznaka c=F(a,b) se može razmatrati kao jednostavan matematički oblik funkcije F, koja, kada se primeni na dve promenljive a i b (operandi), daje vrednost c. Kod računarskih instrukcija, lokacije a, b i c se obično specificiraju preko posebnih polja. Ova polja se normalno zovu *adrese*, i pored toga što to nije u potpunosti tačno, jer ova polja mogu da sadrže, na primer, memorijsku adresu, broj registra ili neposredni podatak. Ograničimo se dalje na konstataciju da ova polja specificiraju adrese.

Zamenom vrednosti za a, b i c dobijamo troadresnu instrukciju (slika 4.2a). U trenutku kada su memorije bile skupe, projektanti su pokušavali da smanje broj bitova po instrukciji. Jedan od načina da se to učini je da se specificiraju registri (potreban je manji broj bitova), umesto kompletne adrese. Alternativni način je da se smesti odredišni operand u istu lokaciju kao i izvorni operand, pa su u tom slučaju potrebne samo dve adrese, kao c=F(a,c), (slika 4.2b). U opštem slučaju, ovo i nije tako nepovoljno, jer rezultat operacije često zamenjuje jedan od operanada, kao na primer inkrementiranje brojača petlje (A:=A+1). Nedostatak je što se jedan od izvornih operanada menja, preko stare vrednosti se upisuje nova (destruktivna operacija).

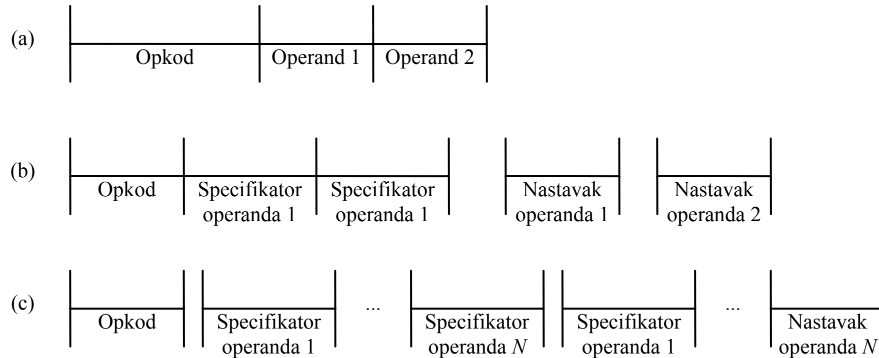


Sl. 4.2. Formati instrukcije: (a) troadresna instrukcija; (b) dvoadresna instrukcija; (c) jednoadresna instrukcija.

Često je jedan od dva operanda lociran u registru, a ne u memoriji, tako da je potrebna specifikacija jedne memorijske adrese i (mali) broj registra. Ove instrukcije su obično poznate kao *jedna-i-po-adresna instrukcija* ($1\frac{1}{2}$, one-and-a-half address instructions).

U prethodnim primerima se podrazumeva da je broj adresa specificiran od strane instrukcije jednak broju operanada koji se koristi u toku izvršenja instrukcije. Dalje pojednostavljenje broja adresa se može postići ako se usvoji da je jedan operand uvek isti (obično je registar). Ovaj operand se zove implicitni operand. Drugi operand se i dalje specificira eksplicitno. Starije mašine, kao što su UNIVAC I, pa i mikroprocesor MOSTEK 6502 koristile su jednoadresne naredbe (slika 4.2c), gde se podrazumeva da je implicitni operand u akumulatorskom registru ili u magacinu, tj. AC=F(a,AC), ili c=F(AC,c).

Koristeći usvojenu terminologiju, moguće je klasifikovati arhitekture u saglasnosti sa formatom instrukcija. Pregled tri klase prikazan je na slici 4.3. Prva klasifikacija se zasniva na tome da je broj operanada fiksiran, (slika 4.3a), ili promenljiv, (slika 4.3c). Skupovi instrukcija sa fiksnim brojem operanada mogu se klasifikovati na one sa fiksnom dužinom, (slika 4.3a), i one sa promenljivom dužinom, (sl 4.3b).



Sl. 4.3. Klasifikacija skupova instrukcija.

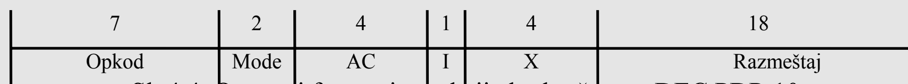
Fiksni broj operanada

Ovaj format instrukcija zbog svoje jednostavnosti se najčešće sreće.

Instrukcije fiksne dužine - skupovi instrukcija sa fiksnim brojem operanada i fiksnom dužinom (slika 4.3a). Ove instrukcije su popularne zbog jednostavnosti i brzine izvršenja a koristile su se u starijim mašinama kao što je PDP-10. U novije vreme se često koriste kod arhitektura gde je brzina izvršenja primarni cilj, kao što su RISC mašine, jer se ove instrukcije brže dekodiraju, a u ove mašine je lakše implementirati akceleracione mehanizme kao što je jedinica za pribavljanje operanada unapred.

Primer 4.1:

Na slici 4.4 prikazan je format instrukcije mašine PDP-10. Sve instrukcije su dužine 36 bitova, a 7 bitova se koristi za specifikaciju opkoda. Instrukcije specificiraju dva operanda, pri čemu je prvi operand uvek jedan od 16 internih registara. Drugi operand se određuje na sledeći način: koristi se I-polje da specificira indirekciju, a X-polje da specificira mogući indeksni registar:



Sl. 4.4. Osnovni format instrukcije kod računara DEC PDP-10.

Tmp := if I=0 and X=0 then Displ
:= if I=0 and X>0 then (X)+Displ
:= if I=1 and X=0 then MŠDisplĆ
:= if I=1 and X>0 then MŠ(X)+DisplĆ

Operand2 := if Mode=0 then Tmp
:= if Mode>0 then MŠTmpĆ

gde MŠAĆ označava sadržaj memorijske lokacije čija je adresa A. Polje Mode određuje kako se međuvrednost Tmp interpretira i gde se smešta rezultat operacije:

Mode=0 : Tmp je neposredna vrednost. Rezultat se upisuje u Operand1 (akumulator).
Mode=1 : Tmp je adresa vrednosti operanda. Rezultat se upisuje u Operand1 (akumulator).
Mode=2 : Tmp je adresa vrednosti operanda. Rezultat se upisuje u Operand2.
Mode=3 : Tmp je adresa vrednosti operanda. Rezultat se upisuje u Operand1 i Operand2.

Operacije registar→registar su takođe moguće, jer se registri adresiraju kada se koriste memorijske adrese 0-15. Ovo zahteva da 23-bitni specifikator operanda specificira akumulator kao operand. Kod PDP-10 za specifikaciju 16 akumulatora potrebna su četiri bita, pa se na taj način uštedelo 19 bitova.

Višestruki nivoi indirekcije su takođe mogući ako je I=1. Tada se 23 LS bita adresne reči interpretira kao 23 LS bita instrukcije, što rezultira da se specificira adresno izračunavanje za drugi operand.

Instrukcije sa promenljivom dužinom - potreba za instrukcijama sa promenljivom dužinom i fiksnim brojem operanada (slika 4.3b) javila se onda kada je postalo ekonomski isplativo praviti računare sa većim brojem registara (tipično 8 do 16 tzv. registara opšte namene) umesto sa jednim akumulatorom. Kod ovakvih arhitektura, jednoadresni instrukcioni format mašine, sa jednim akumulatorom, postao je neadekvatan za specifikaciju dva operanda, dok je klasični dvooperandski format instrukcije zauzimao suviše mnogo memorije kada memorijski operand, kome se vrši obraćanje, mora da specificira registar. Zbog toga je bilo neophodno uvesti specijalni format operanada, koji zahteva samo nekoliko bitova za specifikaciju registra i nešto veći broj bitova za specifikaciju memorijsko-zasnovanog operanda, jednu-i-po ($1\frac{1}{2}$) adresnu instrukciju. Dvoadresne instrukcije ipak nisu potpuno izbačene, one se koriste kod instrukcija sa promenljivom dužinom.

Veliki broj modernih arhitektura imaju skupove instrukcija sa fiksnim brojem operanada i promenljive dužine. Primeri su IBM/370, DEC PDP-11, Intel 80x86, Motorola MC68000, National Semiconductors NS32000 i dr.

Promenljivi broj operanada

Arhitekture (slika 4.3c)) kod kojih je dozvoljen promenljivi broj operanada za datu operaciju su retke. Tipičan primer je VAX-11 koji omogućava da se za najveći broj operacija specificira dva ili tri operanda na sledeći način:

```
ADD2   A,B      ; B:=(B)+(A)
ADD3   A,B,C    ; C:=(B)+(A)
```

Kako se može specificirati samo jedan od operatora (kao što su +, -, * ili /) po instrukciji, maksimalan broj operanada je tri (dva izvorna i jedan odredišni).

Primer 4.2:

Primer VAX instrukcije sa tri operanda je polinomska instrukcija:

```
POLY   argument,degree,table_address
```

Definicija ove instrukcije je data sledećim primerom:

$$P(X)=C_0+C_1*X+C_2*X$$

gde je $C_0=1.0$, $C_1=0.5$ i $C_2=0.25$. Instrukcija za izračunavanje polinoma P(X) je

```
        POLY    X,2,Ptable
                .
                .
                .
Ptable: .Float  .25          ;C2
        .Float  .5           ;C1
        .Float  1.0          ;C0
```

Na ovaj način se mogu implemetirati kompleksne instrukcije sa proizvoljnim brojem operanada.

4.3. Specifikacija komponenata instrukcije

Tri elementa koja su uključna u izvršenje instrukcije su: operacija, lokacije operanada i vrednost operanada. Drugim rečima u svakoj instrukciji su definisana tri različita prostora, a to su:

1. **opkod prostor**, skup svih mogućih operacija;
2. **operandski prostor**, skup svih mogućih imena (adresa) operanada;
3. **prostor podataka**, skup svih mogućih vrednosti operanada.

Osobine instrukcija i operanada se mogu specificirati u svakom od pomenutih prostora. Ove osobine su: opkod, tip podatka, oblast podataka, lokacija podataka i vrednost podataka. Specifikacija ovih osobina i njihov odnos, u odnosu na tri pomenuta prostora, je od interesa za razmatranje, pa zbog toga ukažimo na ove probleme detaljnije.

4.3.1. Specifikacija opkoda

Opkod specificira operaciju koja treba da se izvrši. U najvećem broju slučajeva opkod specificira jednostavne operacije kao što su ADD, MOV ili MUL, ali može specificirati i kompleksne operacije kao što je VAX POLY instrukcija koju čini niz operacija množenja i sabiranja.

4.3.2. Tipovi podataka i specifikacija strukture podataka

U sva tri moguća prostora (slika 4.5) moguće je specificirati tipove podataka kao što su celobrojne reči (integer word W) ili FP (floating point F).

Specifikacija tipova podataka u opkod prostoru. Najčešći metod za specifikaciju tipa podatka, a javlja se kada opkod pored tipa operacije specificira i tip podatka (slika 4.5a).

Ovo normalno zahteva da se za sve operande navede tip podatka i operacija koja se izvršava. Tipični primeri su:

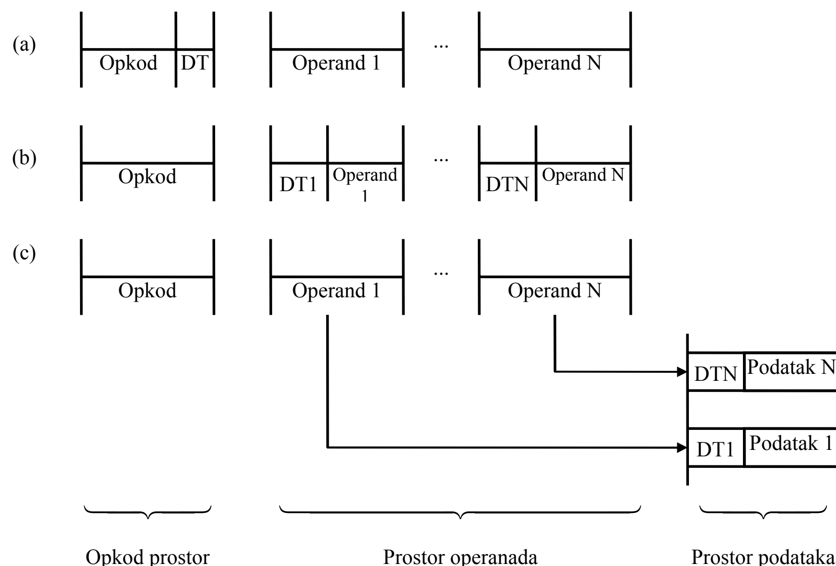
```
ADD.W R1,R2   ; saberi dve celobrojne reči
ADD.F         R3,R4   ; saberi dve FP reči
```

Operacije nad operandima različitog tipa podatka se ne mogu izvršavati direktno. Sa ciljem da se ovo izvede uvode se specijalne instrukcije za konverziju. Kod nekih arhitektura ove konverzije se izvode pomoću potprograma. Tipična arhitektura koja podržava instrukcije za konverziju je VAX-11. Ona poseduje instrukcije za konverziju između svih tipova podataka (bajt, reč, dupla reč, pokretni zarez, pokretni zarez dupla preciznost). Na primer, iskaz:

Float1:=Float2+Word1

se može prevesti u

```
CVTWF Word1,R1       ; konvertuj reč u FP
ADD.F       Float2,R2       ; saberi FP
MOV.F       R1,Float1       ; smesti rezultat
```



Sl. 4.5. Komponente instrukcije.

Ovaj metod specifikacije podataka ukazuje da tipovi podataka moraju biti poznati u fazi kompilacije i iskazi sa operacijama nad podacima različitog tipa ne treba da se javljaju često. Najveći broj jezika, kao što je ALGOL1, FORTRAN i Pascal zahtevaju da se za sve identifikatore eksplicitno specificira tip podatka. Na primer, kod Pascala

var X,Pi,Epsilon : **real**;

Kod FORTRAN-a se svi identifikatori koji počinju slovima I-N smatraju celobrojnim, a ostali se smatraju realnim, osim ako nije drugačije specificirano.

Zahtevi za specifikaciju tipova podataka obezbeđuju se proverom tipova podataka još u toku kompilacije. Naime, iznos memorije koja se dodeljuje ovim promenljivama, ili tip podatka nad kojim se vrši konverzija statički se određuju u toku kompilacije. Ovo "rano" povezivanje tipova podataka sa identifikatorima, poboljšava efikasnost izvršenja programa. Najveći broj HLL-ova podržava rano povezivanje, dok najveći broj arhitektura specificira tip podatka u opkod prostoru. IBM/370 ima 15 različitih ADD instrukcija što znači da je isti broj opkodova potreban.

Specifikacija tipa podatka u prostoru operanda

Ovaj metod zahteva da svaki operand specifikator prati specifikacija tipa podatka (slika 4.5b). Mašine zasnovane na ovom metodu skoro i da ne postoje.

Specifikacija tipa podataka u prostoru za podatke

Ovaj način specifikacije (slika 4.5c) izvodi se uvođenjem *tagova*. To znači da se svaka memorijska reč proširuje jednim poljem (tag polje) koje specificira tip podatka za tu reč. Arhitekture koje koriste ovaj metod se zovu *tagged arhitekture* (tipične mašine su Burroughs B5000, B5500, B6700, i dr.). Kod ove arhitekture svaka memorijska reč (koja je 48 bitova) proširuje se sa trobitnim tagom. Osim FP duple preciznosti za koji treba dve memorijske reči, za sve ostale reči dovoljan je po jedan tag.

Prednost tagova je u tome što se tip podatka (tj. osobina podatka) smešta zajedno sa podatkom. To omogućava da se specificira operacija nezavisno od tipa podatka, i implicitno izvrši konverzija podataka. Ovo je posebno važno kod jezika APL i SNOBOL gde se tip podataka promenljivih menja dinamički u toku izvršenja svakog programskog iskaza.

Druga prednost tagova je što je moguće vršiti proveru operanada. Na primer, kada instrukcija, kao rezultat hardverske greške, pribavi operand koji je označen kao instrukcija, detektovaće se greška.

Nedostatak tagova je što svaka memorijska reč mora da poseduje svoj tag, što zahteva dodatnu memoriju. Prekoračenje postaje izrazito kada je memorijska lokacija tipa bajt. U tom slučaju tri do četiri bita znače prekoračenje od 50%.

Drugi nedostatak tagova ogleda se u manjoj brzini, jer obim operanada se ne zna sve dok se ne pribave (izdvaja se iz podatka koji se pribavlja).

Treći nedostatak je što arhitekture koje koriste tagove imaju dodatno prekoračenje u prostoru za instrukcije i vremenu, jer zahtevaju specijalne instrukcije za inicijalizaciju tagova.

Primer 4.3:

Kod B6700 i B7700, svaka 48-bitna reč se proširuje sa tri tag bita, što ukupno iznosi 51 bit. Interpretacija ovih tagova je prikazana na slici 4.6.

3	48
Tag	Mašinska reč

Tag	Značenje
0 0 0	Operand u jednostrukoj preciznosti
0 0 1	Indirektno referencirana reč (IRW, SIRW)
0 1 0	Operand u dvostrukoj preciznosti
0 1 1	Markiraj upravljačku reč magacina (MSCW)
1 0 0	Reč koraka indeksa (SIW)
1 0 1	Deskriptor podatka ili segmenta (DD, SD)
1 1 0	Koristi se od strane softvera (neinicijalizovano)
1 1 1	Programska reč ili reč povratka (PCW, RCW)

Sl. 4.6.

Operandi duple i jednostruke preciznosti razlikuju se na osnovu vrednosti taga 0 i 2 respektivno, dok se celobrojne vrednosti definišu kao prirodan podskup jednostruke preciznosti, pa se za FP ne zahteva poseban tag. Jednostavne promenljive tipa Boolean ili znakovi se pamte kao 48-bitne vrednosti, jer bajt adresiranje zahteva jedan tag za svaki bajt, a sve ovo rezultira velikim prekoračenjem (tj. gubljenjem prostora).

Specifikacija strukture podataka

Struktura podataka može imati veći broj oblika, počev od proste skalarne promenljive do polja ili zapisa. Veliki broj HLL-ova ne podržava operacije nad strukturnim tipovima podataka. U takvim situacijama, programom se vrši dekompozicija strukturnih tipova podataka na skalarne, nad kojima se sada operacije obavljaju. Ovo je prikazano programskim segmentom koji se koristi za množenje matrica A i B, tako da se dobije R=A*B.

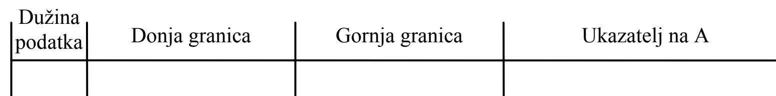
```
const   N=100;
var     I,J,K    : integer;
```

```
        R,A,B    : array [0..N,0..N] of integer;
begin
        for I:=0 to N do
                for J:=0 to N do
                        begin
                                R[I,J]:=0;
                                for K:=0 to N do
                                        R[I,J]:=A[I,K]*B[K,J]+R[I,J];
                        end;
end;
```

Arhitektura koja podržava izvršenje aritmetičkih operacija nad skalarima bi mogla da izvrši ovaj program. Ovo znači da se specifikacija strukture podataka eksplicitno ne podržava od strane arhitekture, ali se zbog toga mora izvršiti sekvenca instrukcija pomoći koje će se simulirati neka struktura podataka.

Zbog toga što postoje različiti načini kako se podaci mogu struktuirati, najveći broj arhitektura ne podržava strukturne podatke. Umesto toga, preslikavanje ovih struktura na one koje se podržavaju od strane arhitekture se obično prepušta kompilatoru ili interpretatoru. Arhitekturna podrška za operacije nad vektorima i nizovima nije standardna, kao i arhitekturna podrška za pristup elementima polja. Kod ovakvih struktura obično se koriste deskriptori.

Deskriptori se mogu koristiti za podršku za pristup elementima polja, mada se konceptualno oni mogu koristiti za specifikaciju bilo koje strukture podataka. Da bi podržao pristup skalaru u strukturi podataka, deskriptor sadrži sve informacije koje opisuju strukturu podataka. Na primer, deskriptor na slici 4.7 se može koristiti za opis vektora A[L..U] od FP brojeva.



Sl. 4.7. Deskriptor za polje A[L..U].

Polja deskriptora sadrže sve informacije koje su neophodne za pristup elementu vektora A[L..U]. Prva tri polja specificiraju dužinu polja elementa, gornju i donju granicu polja. Podatak koji u deskriptoru ukazuje na dužinu često se koristi za adresno izračunavanje, ali ne i za operacije koje specificiraju izbor tip operanda. Zadnje polje sadrži pokazivač na podatke polja A.

Prednost deskriptora je što oni mogu automatski da obavljaju sve operacije koje se odnose na pristup elementu strukture. Kada se, na primer, pristupa elementu A[I], proverava se vrednost I da bi se utvrdilo da li je važeći uslov $L \leq I \leq U$, da bi se nakon toga korektno izračunala adresa elementa A[I], pošto su poznate vrednosti o dužini polja i dužini elementa.

Nedostatak deskriptora se ogleda u ugradnji specijalnog hardvera koji je potreban za podršku njihovog rada. Dobar kompilator može često generisati kod pomoću koga se može eliminisati veći broj funkcija koje se koriste za pristup podacima, a koriste deskriptor.

Na primer, za sledeći program nije potrebna provera indeksa I.

```
const   N=100;
var     I  : integer;
        A :array [0..N] of integer;
begin
        for I:=0 to N do
        A[I]:=A[I]*25;
end;
```

4.3.3. Prostor podataka i specifikacija lokacije podataka

Instrukcijama je neophodno specificirati i prostor podataka kome se vrši obraćanje. Oblasti podataka koje se najčešće koriste su: glavna memorija; registri opšte namene; U/I prostor; magacinski prostor; specijalni registri (kao što su statusni registri). Oblast podataka može se specificirati samo na dva mesta, u opkod prostoru ili u prostoru operanada. Oblast podataka se ne može specificirati u samoj sebi, jer je oblast podataka subjekt specifikacije i ne možemo se njoj obraćati, ako je ne znamo.

Specifikacija oblasti podataka u opkod prostoru

U retkim slučajevima opkod kao oblast se koristi za specifikaciju oblasti podataka. Na primer, kod mikoprocesora MC68020 instrukciju PEA (Push Effective Address) implicitno se vrši obraćanje magacinskoj oblasti podataka. Kod arhitektura koje imaju izdvojeni U/I prostor (Intel 80x86) instrukcijama IN i OUT implicitno se vrši obraćanje U/I prostoru.

Specifikacija oblasti podataka u operandskom prostoru

Specifikacija oblasti podataka u operandskom prostoru se može izvesti implicitno ili eksplicitno. Implicitna specifikacija može da koristi redosled da bi se odredilo o kom operandu je reč. Na primer, kod formata instrukcije za PDP-10 (slika 4.4) prvi operand (AC) uvek se odnosi na jedan od 16 registara opšte namene, dok drugi operand uvek specificira lokaciju u glavnoj memoriji. Eksplicitna specifikacija oblasti podataka u operandskom prostoru se najčešće koristi. Neki od tipičnih primera za MC68020 su:

```
EOR.W     D3,0XF874.W   ; D3 specificira registarski prostor, a 0XF874
                        ; lokaciju reči u prostoru glavne memorije
DIVS.W    #8,(SP)+      ; #8 specificira konstantu 8 u instrukcionom nizu a
                        ; (SP)+ specificira oblast magacina dok se
                        ; pokazivač magacina inkrementira
```

Specifikacija lokacije podataka

Nakon što se oblast podataka specificira, neophodno je definisati lokaciju podataka. ovo ukazuje gde se podaci u okviru izabranog prostora podataka mogu naći, a često se specificiraju relativno u odnosu na početak prostora podataka. Lokacija podataka treba, na primer, da specificira koji se od registara u registarskom prostoru za podatke koristi.

4.3.4. Specifikacija vrednosti podataka

Vrednost podataka se može takođe specificirati u jednoj od tri prethodno pomenute oblasti, mada se oblast podataka najčešće koristi za specifikaciju vrednosti podataka. Prostor operanada se koristi za specifikaciju lokacije podataka, a to je obično memorijska lokacija ili registar. Specijalna mesta gde se nalazi specifikacija vrednosti podataka su i opkod-prostor i prostor operanada.

Specifikacija vrednosti podataka u opkod prostoru

Specifikacija malih konstanti u opkod prostoru se često izvodi sa ciljem da se štedi memorija. Na primer, instrukcijom MOVE.L #0, Mem; kod MC68020, zahtevaju se dve 16-bitne reči da se specificira neposredni (duga reč) podatak. Kod instrukcije CLR.L Mem, ova vrednost se specificira od strane opkoda, pa se na taj način dužina instrukcije smanji za dve reči. Drugi primer su instrukcije INC i DEC, koje inkrementiraju i dekrementiraju promenljivu za 1.

Specifikacija vrednosti podataka u operandskom prostoru

Ovaj metod se koristi kod svih instrukcija za specifikaciju neposrednih vrednosti (konstanti). Specifikacija konstanti u operandskom prostoru povećava brzinu izvršenja, jer se ne vrši obraćanje konstantama u memoriji, pa se na taj način štedi dodatni memorijski ciklus čitanja. Pored toga štedi se memorija, jer bitovi kojima se specificira adresa konstante u ovom slučaju nisu potrebni.

4.3.5. Indirektna specifikacija komponenata instrukcije

U dosadašnjoj analizi ukazali smo da se komponente instrukcija (tip podatka, struktura podataka itd.) specificiraju direktno u prostorima za opkod, operanada ili podataka. Ipak, ovo nije uvek neophodno. Svaka komponenta instrukcije se može specificirati indirektno preko indeksa u tabeli ili preko pokazivača. I pored toga što se teoretski sve komponente mogu specificirati indirektno u svakom od pomenutih prostora, ograničićemo se samo na onu specifikaciju koja se najčešće koristi: indirektna specifikacija operanada.

Indirektna specifikacija operanada znači da lokacija, i kojoj se standardno čuva vrednost operanda, sada sadrži vrednost na osnovu koje se vrši obraćanje operandu. Indirektna specifikacija operanda je važna u onim slučajevima kada se adresa izračunava u toku izvršenja programa.

4.4. Zavisnost između komponenata instrukcije

Instrukcija sadrži sledeće komponente (bilo da su ili ne eksplicitno specificirani): operacija; tip podatka, prostor podataka, vrednost podataka i lokaciju podataka. Na žalost ove komponente nisu međusobno potpuno nezavisne. Na primer, postoji relacija između tipa podataka i operacije: nije svaka instrukcija osetljiva na svaki tip podataka (na primer množenje znakova). Druga relacija važi za tip podataka i dužinu podataka: neki tipovi podataka imaju fiksne dužine. Veliki broj takvih zavisnosti postoji ali u daljoj analizi ukažimo samo na složene podatke. Standardno jedan podatak se specificira jednim operandom. Da bi omogućili kompleksnije izračunavanje adresa, neophodno je da se koristi nekoliko operanada (uglavnom dva, pri čemu svaki ima po nekoliko komponenata) pomoću kojih se zajednički određuje lokacija podataka. Ovo se zove složeni operand (compound operand). U ovom slučaju postoji zavisnost između operanada u prostoru operanada u vezi sa tim kako se operandi određuju, imajući u vidu određivanje prethodnog operanda.

Primer 4.4:

Kod VAX-11 moguće je direktno adresirati element polja AŠIĆ koristeći se indeksnim adresnim načinom rada. Vrednost I (indeks relativan u odnosu na početnu vrednost A) specificira se prvim operandom i može se smestiti u registar (Rx). Prvi operand takođe specificira indeksni način rada, dok se početna adresa A specificira drugim operandom kao standardni VAX-11 operand, a bilo koji adresni način rada se može koristiti. Ovo znači da se adresa A može identično specificirati za bilo koji instrukcioni operand mašine VAX-11.

Na slici 4.8. prikazan je specifikator operanda (za VAX-11) za operand 1 koji specificira indeksni adresni način rada (mode=4). Vrednost indeksnog registra se množi dužinom podatka (u bajtovima). Dužina (specificirana tipom podatka u opkod prostoru) može biti 1, 2, 4 ili 8 bajtova. Operand 2 je standardni operand sa uobičajenim formatom i specificira početnu adresu A. Lokacija elemenata u polju određuje se na sledeći način:

Loc:=value_operand2+(Rx)*data_length