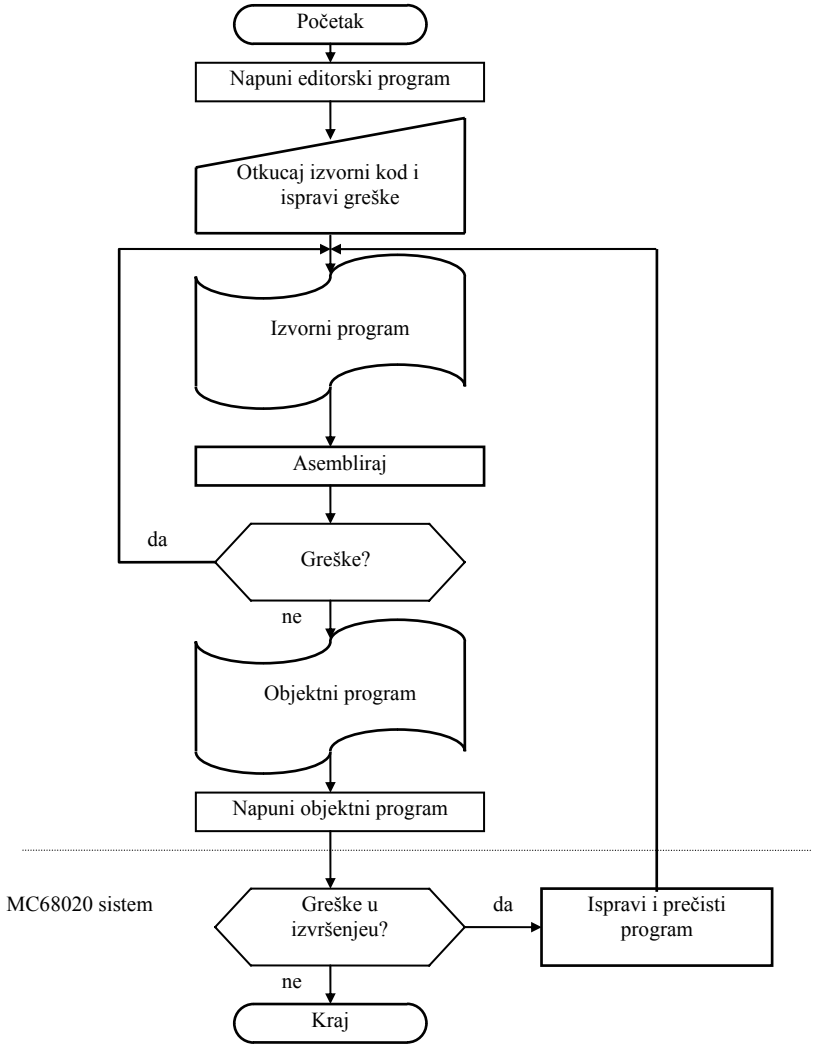
2. ASEMBLERSKI JEZIK MIKROPROCESORA MC68020

2.1. Uvod

Razvoj softvera čine:

- analiza problema,
- projektovanje softvera,
- kodiranje problema,
- čišćenje programa od grešaka (*debugginig*), i
- testiranje.

Odgovarajuća dokumentacija mora da postoji za svaki korak. Programske aktivnosti u pojednostavljenoj formi su prikazane na slici 2.1.



Sl. 2.1. Programiranje mikroračunarskog sistema zasnovanog na MC68020.

Program editor se koristi za kreiranje izvornog programa napisanog na asemblerskom jeziku koji se zatim prevodi od strane asemblera. Na ovom nivou razvoja za nalaženje grešaka u programu koristi se asemblerski listing. U listingu se daje izvorni program na asemblerskom jeziku i njegov ekvivalent na mašinskom jeziku u slučaju kada nije detektovana greška u asembleru. Greške u izvornom programu naznačene su u listingu. Kada u programu ne postoje greške, generiše se objektni kod. Objektni kod se puni u memoriju ciljne mašine radi izvršavanja. Izvršenje programskog koda se kontroliše u toku prečišćavanja programa od strane programa koji se zove dibager. Ovaj program omogućava korisniku da sagleda izvršenje programa instrukcija-po-instrukciji i da prikaže neposredne rezultate nakon izvršenja svake instrukcije. Greške u toku kreiranja programa uočavaju se u ovom stepenu. Da bi se korigovale greške izvorni program mora da se ponovo edituje i asemblira.

2.2. Listing na asemblerskom jeziku

Kao što smo prethodno uočili, asemblerskim procesom vrši se provera izvornih iskaza radi otkrivanja grešaka. Svaki iskaz može biti instrukcija mikroprocesora MC68020, asemblerska direktiva ili komentar. Simbolička instrukcija kao što je

```
ADD.W     D1,D2
```

postaje instrukcija na mašinskom jeziku koja se može izvršiti. Mnemonik ADD, obim operanda i adrese operanada prepoznaju se od strane asemblera i konvertuju u binarni mašinski kod. Asemblerske direktive, sa druge strane, su instrukcije za asembler a ne za CPU. Direktiva ORG (početak) specificira gde se program puni u memoriju. Na primer, direktiva

```
ORG       $10000
```

ukazuje da će se program puniti u memoriju počev od heskadecimalne lokacije $10000.

Komentari radi boljeg sagledavanja programa od strane programera mogu postojati u izvornom programu. Ovi komentari se ignorišu od strane asemblera, ali se štampaju u listingu.

Na slici 2.2 prikazan je jedan tipičan listing asemblerskog programa mikroprocesora MC68020.

```
00010000                    1.          TTL         SLIKA 2.2
                            2.          LLEN        100          ; Duzina linije
                            3.          ORG         $10000       ; Pocetak u memoriji
                            4. *
                            5. * Saberi cetiri 16-bitna broja smestena u lokacijama
                            6. * $20000 do $20006. Vratiti sumu u D1[15:0]
                            7. *
00010000 7200               8. INIT     MOVE.L      #0,D1        ; Suma dobija vrednost nuli
00010002 227C 00020000      9.          MOVEA.L     $20000,A1; Adresa prvog broja
00010008 7404              10.          MOVE.L      #4,D2        ; Postavi brojac na 4
                           11. *
                           12  * Definisanje petlje za sabiranje vrednosti
                           13. *
0001000A D259              14. LOOP     ADD.W       (A1)+,D1     ; Saberi brojeve
0001000C 5342              15.          SUB.W       #1,D2        ; Uvecaj brojac
0001000E 66 FA             16.          BNE         LOOP         ; Dok ne bude (D2)=0
                           17. *
00010010 4E4F              18.          TRAP        #15          ; Povratak na monitor
00010012 0063              19.          DC.W        $0063
00010014                   20.          END
```

Sl. 2.2. Tipični asemblerski listing.

Tumačenje:

- prva kolona ukazuje na vrednost programskog brojača; u datom primeru on se početno puni na vrednost $10000.
- druga kolona je mašinsko-jezička translacija i prikazuje opreč za svaku instrukciju iza koje mogu da slede reči proširenja zahtevane od strane instrukcije.
- treća kolona predstavlja decimalni brojač linija, tj. lokacioni brojač koji u toku asembliranja čuva trag o lokaciji instrukcija.
- INIT ukazuje na lokaciju prve instrukcije.
- LOOP definiše početak sekvence instrukcija koje se u konkretnom programu ponavljaju četiri puta.
- direktive LLEN, ORG i END definišu obim listinga, početak i kraj programa, respektivno.

- TTL (title) je takođe direktiva koja je opciona i koristi se za identifikaciju programa, tj. interne napomene programera.
- DC vrednost koja se dodeljuje od strane asemblerske direktive.
- * ukupna linija se tretira kao komentar, gde je * prvi znak u liniji.
- ; ostatak linije je komentar.

2.3. Kros-softver za razvoj

U dosadašnjoj diskusiji, pretpostavili smo da su aplikacioni programi bili razvijeni radi izvršenja na ciljnoj mašini. Alternativni pristup omogućava da se aplikacioni programi kreiraju na drugom računaru i prevode u izvršnu verziju (mašinski jezik) za ciljnu mašinu. Ova tehnika je poznata pod imenom kros-razvoj, a asembler se zove kros-asembler.

Prednost kros-razvojnog pristupa se ogleda u tome što programer bez (recimo) razvojnog sistema za MC68020, kao što je Motorola SYS1131, može da koristi programsko razvojne mogućnosti glavnog računara (hosta), tj. drugi tip mašine. Drugim rečima, host ima instalirane programe kao što su tekst editor za pripremu teksta, smeštanje programa na disku, štampanje programa radi korekcije ili dokumentacije. Da bi se izvršio kros-asemblirani program, instrukcije na mašinskom jeziku se moraju procesirati bilo od strane simulatora bilo od strane računarskog sistema zasnovanog na MC68020. Pomoću simulatora, operacije MC68020 se simuliraju na kros-razvojnom računaru. Ovaj pristup je koristan kada hardver nije dostupan, a dobar je zbog toga što obezbeđuje softverski model za MC68020, a aspekti zavisni na hardvera,kao što je sinhronizacija (tajming), moraju se testirati na ciljnom računaru zasnovanom na MC68020. Ovo se postiže prenošenjem kros-asembliranog programa sa sistema gde je program razvije u memoriju ciljnog sistema.

2.4. Linkovanje

Prvi korak kod razvoja softvera je kreiranje izvornog programa i njegovo prevođenje pomoću asemblera u relokatibilni objektni modul. Objektni modul u opštem slučaju ne sadrži apsolutne adrese pomoću kojih se vrši obraćanje memorijskom sistemu procesora MC68020, nego relokatibilne adrese koje će biti definisane pomoću programa "linkage editor". "Linkage editor" se koristi da kombinuje nekoliko objektnih modula u "load modul". Program "Link file" lista imena i drugih informacija o modulima koji se povezuju. Linkovani modul se može smestiti na disk ili puniti u memoriju ciljnog računara.

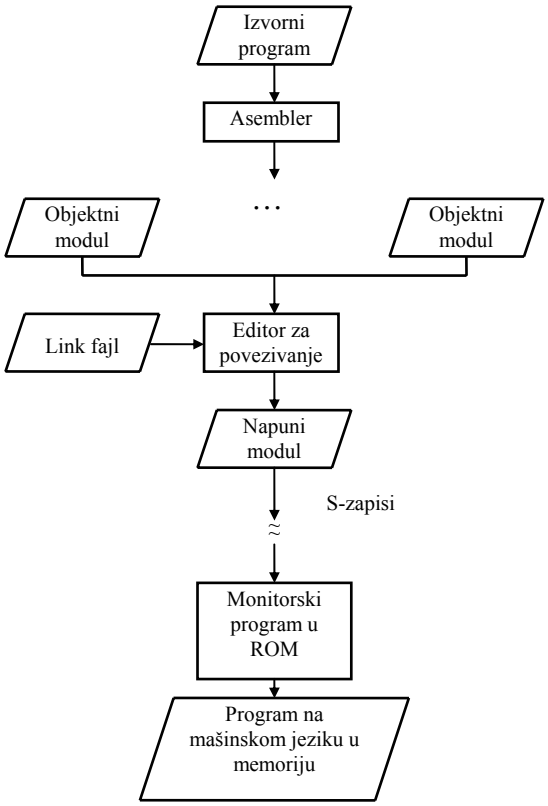
Na slici 2.3 prikazan je princip linkovanja.

2.5. Karakteristike asemblerskog jezika

Svaki iskaz asemblerskog jezika sastoji se od četiri polja: labela, operacija, operand(i) i komentar. Polja se razdvajaju pomoću blanko ili drugih delimitera saglasno zahtevanom formatu. Format opšteg iskaza na asemblerskom jeziku prikazan je u Tabeli 2.1.

Tab. 2.1. Format asemblerskog jezika.

Polje labela	Polje za kod operacije ili direktivu	Polje za operande	Polje za komentar
[<LABELA>]	<kod operacije> ili <direktiva>	[<operand1>[,<operand2>]]	[<komentar>]



Sl. 2.3. Princip linkovanja.

2.5.1. Labela

Labela je opciona kod najvećeg broja instrukcija i direktiva. Kada se koristi, ona predstavlja adresu. Labeli se dodeljuje vrednost lokacionog brojača. Na primer, u iskazu

OVDE MOVE.W D1,D3

labela OVDE definiše lokaciju instrukcije u memoriji nakon što je program napunjen i može se koristiti za definisanje početka programskog segmenta kod kasnijeg referenciranja.

Primer 2.1:

Na slici 2.4 prikazano je nekoliko primera labela koje se koriste kao adrese.

```
00010000                    1.          TTL         SLIKA 2.4
                            2.          LLEN        100             ; Duzina linije
                            3.          ORG         $10000          ; Pocetak u memoriji
                            4.  *
                            5.  * Saberi cetiri 16-bitna broja smestena u lokacijama
                            6.  * $20000 do $20006. Vratiti sumu u D1[15:0]
                            7.  *
00010000  7200              8.  INIT    MOVE.L      #0,D1           ; Suma dobija vrednost nuli
00010002  227C  00020000    9.          MOVEA.L     $20000,A1; Adresa prvog broja
00010008  7404             10.          MOVE.L      #4,D2           ; Postavi brojac na 4
                           11.  *
                           12   * Definisanje petlje za sabiranje vrednosti
                           13.  *
0001000A  D259             14.  LOOP    ADD.W       (A1)+,D1        ; Saberi brojeve
0001000C  5342             15.          SUB.W       #1,D2           ; Uvecaj brojac
0001000E  66  FA           16.          BNE         LOOP            ; Dok ne bude (D2)=0
                           17.  *
00010010  4E4F             18.          TRAP        #15             ; Povratak na monitor
00010012  0063             19.          DC.W        $0063
00010014                   20.          END
```

Sl. 2.4. Korišćenje labela.

2.5.2. Kodovi operacija

Drugo polje izvornog iskaza mora da sadrži mnemonik instrukcije ili asemblersku direktivu. Dužina operanda se specificira kao deo instrukcionog polja i obeležava se sa "." koja se pridružuje opkodu, i sastoji se od B,W ili L za specifikaciju bajta, reči ili duple reči, respektivno.

Na primer, instrukcija

MOVE.W D1,D2

definiše operande dužine reč.

2.5.3. Operandi

Lokacijama operanada instrukcije pristupa se u saglasnosti sa adresnim načinima rada za svaki operand. Opšti format operanada prikazan je u Tabeli 2.2.

Tab. 2.2. Opšti format operanda.

Operand	Format		Tipična referenca ili operand	Primer
Nema	OPR		Spoljni uređaj	RESET
Implicitan	OPR		PC, SP ili SR	NOP, TRAPV, RTS
Neposredni	OPR	<vrednost>	Upravljanje procesorom ili instrukcije koje zahtevaju vrednost	TRAP, STOP
Jedan	OPR	<adresa>	Relativna adresa	BRA
			Adresa Instrukcije	JMP
			Adresa operanda	CLR, NEG
Dva	OPR	<vredost>,odredište>	Neposredna vrednost na odreište ili dve adrese	ADD, MOVE
	OPR	<izvor>,<odredište>		

Napomena:
1. OPR je proizvoljni valjani kod operacije.
2. Moguće se manje varijacije prikazanih formata.

Primer 2.2:

Na sl. 2.5 prikazan je jedan listing na asemblerskom jeziku kao primer specifikacije operanada.

```
                          1.          TTL         SLIKA 2.5
                          2.          LLEN        100
00010000                  3.          ORG         $10000
                          4.  *
                          5.  * Razne instrukcije
                          6.  *
00010000  4E70            7.          RESET
                          8.  *
00010002  4E4F            9.  *       TRAP        #15
                          10. *
00010004  4E75            11          RTS
                          12. *
                          13. * Jednoadresne
                          14. *
00010006  4241            15.         CLR.W       D1              ; Direktno registarsko za podatke
00010008  4278 1000       16.         CLR.W       $1000           ; Apsolutno
0001000C  4251            17.         CLR.W       (A1)            ; Indirektno
0001000E  4261            18.         CLR.W       -(A1)           ; Predekrementiranje
00010010  4259            19.         CLR.W       (A1)+           ; Postinkrementiranje
00010012  4269 0002       20.         CLR.W       2(A1)           ; Indirektno sa razmeštajem
00010016  4271 1002       21.         CLR.W       (2,A1,D1.W)     ; Indirektno sa indeksiranjem
0001001A  4271 A802       22.         CLR.W       (2,A1,A2.L)
                          23. *
                          24. * Dvoadresne (specificirana adresa izvora)
                          25. *
0001001E  3401            26.         MOVE.W      D1,D2           ; Direktno registarsko za podatke
00010020  3409            27.         MOVE.W      A1,D2           ; Direktno adresno registarsko
```

```
00010022  3438  1000          28.       MOVE.W      $1000,D2          ; Apsolutno
00010026  3411                29.       MOVE.W      (A1),D2           ; Indirektno
00010028  3421                30.       MOVE.W      -(A1),D2          ; Predekrementiranje
0001002A  3419                31.       MOVE.W      (A1)+,D2          ; Postinkrementiranje
0001002C  3421  0002          32.       MOVE.W      2(A1),D2          ; Indirektno sa razmeštajem
00010030  3431  1002          33.       MOVE.W      (2A1,D1.W),D2     ; Indirektno sa indeksiranjem
00010034  343C  0005          34.       MOVE.W      #5,D2             ; Neposredno
00010038  3439  00010040      35.       MOVE.W      ????,D2           ; Relativno
                              36. *
0001003E  4E4F                37.       TRAP        #15               ; Povratak
00010040  0063                38.       DC.W        $0063
00010042                      39.       END
```

Sl. 2.5.

RESET i RTS instrukcije ne zahtevaju operande.
TRAP instrukcija mora da ima trap broj koji se specificira kao neposredna vrednost.
CLR zahteva jednu adresu u polju operanda.
MOVE zahteva dve adrese.

2.5.4. Izrazi kao operandi

Asembler prepoznaje određene simbole u polju operanda. Simbol može označavati apsolutnu adresu, neposrednu vrednost, ili bilo koji drugi važeći operand. *Izraz* predstavlja kombinaciju simbola, konstanti, algebarskih operatora i zagrada koje asembler "procenjuje" da bi odredio adresu ili vrednost operanda.

Za specifikaciju konstantne vrednosti, koja se često zove *literal*, koristi se neposredno adresiranje. Kod najvećeg broja asemblera, konstante se predstavljaju bilo kao brojevi, bilo kao ASCII znaci. Ove konstante su najprostiji oblik izraza i specificiraju se korišćenjem definicije u Tabeli. 2.3.

Tab. 2.3. Asemblerski simboli za izraze.

Simbolički izraz	Interpretacija
$<Broj>	Heksadecimlni broj
<Broj>	Decimalni broj
"<String>'	ASCII niz znakova
#<Broj>	Neposredni operand
#'<String>'	Neposredni operand
U izrazu	
+	Sabiranje
-	Oduzimanje
*	Množenje
/	Deljenje
( )	Grupisanje

Primer 2.3:

```
                    MOVE.W          #000,D1
```

definiše 16-bitnu heksadecimalnu vrednost kao neposredni izvorni operand.

Primer 2.4:

Niz 'ABCD' prepoznaje se od strane asemblera i konvertuje se u ASCII kod

```
                    41 42 43 44
```

i kada se smešta u memoriju zauzima četiri bajta.

Primer 2.5:

Neposredna vrednost -1 se izračunava u instrukciji

```
                    MOVE.W          #-1,D1
```

i smešta se kao #$FFFF sa instrukcijom. Ekvivalentna specifikacija je

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

Primer 2.6:

Na slici 2.6 prikazani su različiti načini korišćenja labela i izraza.

```
                          1.                TTL               SLIKA 2.6
                          2.                LLEN              100
00010000                  3.                ORG               $10000
                          4.  *
                          5.  * Koriscenje labela i izraza
                          6.  *
00010000  227C  00020000  7.  START          MOVEA.L           $20000,A1; Prva adresa
00010006  7404            8.                MOVE.L            #4,D2             ; Brojac
                          9.  *
                         10.  *
                         11. Suma se dodaje pocetnoj vrednosti u D1
                         12. *
00010008  D259           13.  LOOP           ADD.W             (A1)+,D1          ; Saberi 4 broja
0001000A  5382           14.                 SUBQ.L            #1,D2
0001000C  66  FA         15.                 BNE               LOOP
0001000E  4E71           16.  ENDLP NOP
                         17. *
                         18. * Duzina programa u bajtovima
                         19. *
00010010  760E           20.                 MOVE.L            #(ENDLP-START),D3 ; (D3)=14
                         21. *
                         22. * Duzina programa u recima
                         23. *
00010012  7B07           24.                 MOVE.L            #(ENDLP-START)/2,D4 ; (D4)=7
                         25. *
                         26. * Sadrzaj instrukcije na labeli START u D5
                         27. *
00010014  2A39  00010000 28.                 MOVE.L            START,D5          ; (D5)=227C  0002
                         29. *
                         30. * Adresa pocetka u A2
                         31. *
0001001A  247C  0001000  32.                 MOVEA.L           #START,A2         ; (A2)=10000
                         33. *
                         34. * ASCII string u D6
                         35. *
00010020                 36.                 MOVE.L            #'DONE',D6        ; (D6)=444F 4EE5
                         37. *
00010026  4E4F           38.                 TRAP              #15
00010028  0063           39.                 DC.W              $0063
0001002A                 40.                 END
```

Sl. 2.6. Korišćenje labela i izraza.

2.5.5. Asemblerske direktive

Kod najvećeg broja asemblera postoje asemblerske direktive koje su u suštini instrukcije za asembler, a ne za procesor. Akcija koja se obavlja od strane svake direktive javlja se samo kada se asemblira izvorni program. Glavne kategorije direktiva se odnose na upravljanje asembliranjem, definiciju simbola, definiciju podataka i dodelu memorije i upravljanje listingom (Tabela 2.4).

Tab. 2.4. Asemblerske direktive.

Direktiva i format	Značenje
Upravljanje asemblerom	
ORG <izraz>	Početak
END	
Definisanje simboličkih imena	
<labela> EQU <izraz>	Dodelju je vrednost imenu <labela>
Definisanje podataka i memorijskog prostora	
[<labela>] DC.<l> <vrednost(i)>	Definiše konstantu(e)
[<labela>] DS.<l> <broj>	Rezerviše prostor
Upravljanje listingom	
LLEN <N>	Dužina linije
LIST	Listanje (podrazumeva se)
NOLIST	Zabrana listanja
SPC <N>	<N> praznih linija
PAGE	Sledeća strana

Napomena:
1. Srednje zagrade označavaju opciona polja.
2. <l> označava B, W ili L.

Primer 2.7:

EQU direktiva se koristi da izjednači broj sa simbolom. Vrednost može predstavljati adresu ili konstantu. Iskaz

```
TTYOUT    EQU    $7FFF
```

dodeljuje vrednost $7FFF simbolu TTYOUT. Iskazom

```
          MOVEA.L    #TTYOUT,A1
```

prenosi se broj $7FFF u adresni registar A1.

Primer 2.8:

Dve direktive DC (Define Constant) i DS (Define Storage) dostupne su za inicijalizaciju vrednosti u memoriji i rezervaciju memorijskog prostora, respektivno.

DC direktiva slična je DATA iskazu u FORTRAN-u, kod koje se definisanoj promenljivoj dodeljuje inicijalna vrednost, a DS direktiva slična je DIMENSION iskazu, koji rezerviše prostor za promenljive ali im ne dodeljuje vrednosti.

Iskazom

```
          INITV      DC.W 20
```

uslovljava se da decimalna vrednost 20 bude upisana kao reč na lokaciji INITV.

Direktivom

```
          DS.W       $10
```

rezerviše se 16 reči u memoriji.

Direktivom

```
ADDPR1    DC.W       LABEL+1
```

smestiće se adresa LABEL+1 na 16-bitnu lokaciju ADDPR1.

Primer 2.9:

LLEN direktiva određuje broj karakterta koji se štampaju u liniji.

SPC direktiva uslovljava da se specificirani broj blanko linija pojavi kod kucanja sa ciljem da se potencira čitljivost.

PAGE direktiva uslovljava postavljanje na početak nove stranice svaki put kada se pozove.

Način korišćenja LIST i NOLIST direktiva je sledeći:

```
LIST
(segment I)
NOLIST
(segmemt II)
```

```
                    LIST
                    (segment III)
                    END
```

Primer 2.10:

Na slici. 2.7 prikazan je primer korišćenja asemblerskih direktiva.

```
                          1.                TTL               SLIKA 2.7
                          2.                LLEN              100
00010000                  3.                ORG               $10000
          # 00000001      4. ONE            EQU               1                 ; Konstanta
          # 00020000      5. PROGRAM        EQU               $20000            ; Pocetna adresa
          # 00000063      6. RETURN         EQU               $0063
                          7. *
                          8. * Oblast podataka
                          9. *
00010000  0A 05 07       10. INITDT         DC.B              10,5,7            ; Bajtovi - decimalni
          -- poravnanje granica
00010004  0000000A       11.                DC.L              10,5,7            ; Duge reci
          00000005
          00000007
00010010  FF 10 AF        12.               DC.B              $FF,$10,$AF       ; Bajtovi heksadecimalno
          -- poravnanje granica
00010014  000000FF       13.                DC.L              $FF,$20,$AE       ; Duge reci
          00000020
          000000AE
00010020  41 42 43 44 45  14.                DC.B              'ABCDEFGH'        ; Bajtovi ASCII
          46 47 48
00010028  41 00 00 00 42  15.                DC.L              'A','BC'          ; Duge reci
          43 00 00
                          16. *
00010030  00010000        17. INIVADD        DC.L              INITDT            ; Adresni ulaz
00010034  <14>            18. COMMON         DS.W              10                ; 10 reci
00010048  <20>            19. HEXVAL         DS.W              $10               ; 16 reci
00010068  <3>             20. BYTES DS.B                 3                  ; 3 bajta
          -- poravnanje granica
0001006C  <0>             21.                DS                0                 ; Parna granica
                          22. *
                          23. * Duzina se izracunava kao poslednja lokacija podataka minus prva lokacija podataka:
                          24. * (BYTES+3-COMMON)
                          25. *
          # 00000037       26. LENGTH        EQU               (BYTES+3-COMMON) ; Broj bajtova
                          27. *
00020000                  28.                ORG               PROGRAM
                          29. *
                          30. * Brisanje COMMON vrednosti
                          31. *
00020000  123C 0037       32. BEGIN          MOVE.B            #LENGTH,D1        ; Brojac
00020004  227C 00010034   33.                MOVEA.L           #COMMON,A1        ; Adresa prve reci
                          34. *
0002000A  12FC 0000       35. LOOP           MOVE.B            #0,(A1)+          ; Brisi podrucje COMMON
0002000E  5301            36.                SUBQ.B            #ONE,D1
00020010  66 F8           37.                BNE               LOOP
                          38. *
00020012  4E4F            39.                TRAP              #15
00020014  0063            40.                DC.W              $0063
00020016                  41.                END
```

Sl. 2.7. Korišćenje asemblerskih direktiva.

2.6. Instrukcije za prenos podataka, programsko upravljanje i poziv potprograma

Instrukcije i njihove varijacije izdvojene su u kategorije koje se zasnivaju na operacijama koje se obavljaju. Analizirajmo prvo klase instrukcija, koje su date u Tabeli 2.5. Instrukcije MOVE, MOVEQ, MOVEM, EXG i SWAP predstavljaju instrukcije za prenos podataka i date su u Tabeli 2.6.

Tab. 2.5. Odabrane instrukcije.

	Prenos podataka
MOVE	Kopiranje
MOVEQ	Brzo kopiranje (neposredno)
MOVEM	Kopiranje više registara
EXG	Razmena vrednosti registara
SWAP	Razmena polovina registra
	Upravljanje programom
Bezuslovno	
BRA	Granaj se uvek
JMP	Skoči
Uslovno grananje i poređenje	
Bcc	Uslovno grananje
DBcc	Testiraj uslov, dekremetiraj i granaj se
CMP	Poredi
CMPI	Poredi neposredno
CMPM	Poredi memorijske lokacije
TST	Testiraj
	Potprogrami
BSR	Granaj se na potprogram
JSR	Skoči na potprogram
RTR	Povratak i obnavljanje (CCR)
RTS	Povratak iz potprograma

Tab. 2.6. Instrukcije za prenos podataka.

Instrukcija	Sintaksa	Dužina operanda u bitovima	Adresni načini rada		Uticaj na kodove uslova
			Izvor	Odredište	
Kopiranje	MOVE.<l> <EAs>,<EAd>	8, 16, 32	Svi[2]	Za podatke	N, Z, V={0}, C={0}
Brzo kopiranje	MOVEQ #<d_8>,<Dn>	32	Neposredno (znakovno prošireno)	Dn	N, Z, V={0}, C={0}
Kopiranje više registara	MOVEM.<l_1> <EA>,<lista>[1]	16 ili 32	Lista registara	Upravljački ili predekremntni	Ne
	MOVEM.<l_1> <EA>,<lista>[1]	16 ili 32	Upravljački ili postinkrementni	Lista registara	Ne
Izmena registara	EXG <Rx>,<Ry>	32	Rx	Ry	Ne
Razmena polovina registara	SWAP <Dn>	16	(Dn)[31:16]↔(Dn)[15:0]	—	N, Z, V={0}, C={0}

Napomene:
1. <EA> efektivna adresa
   <l> B, W ili L
   <l_1> W ili L
   <Rn> Bilo koji Dn ili An
   <lista> lista registara
2. Ako je operand obima bajt, <An> ne može biti izvor
3. U instrukciji MOVEM.W operandi koji se prenose u registre u znakovno prošireni do 32 bita u odredišnom registru.

2.6.1. MOVE instrukcija

Kod MC68020 postoji skup operacija za kopiranje podataka. MOVE instrukcija je jedna od najfleksibilnijih instrukcija, jer ona može koristiti sve adresne načine rada kako za izvorni tako i za odredišni operand. Ova fleksibilnost je neophodna, jer se MOVE instrukcija koristi često.

Shodno tabeli 2.2 uočimo da se vrednost koja se prenosi MOVE instrukcijom tretira kao označena celobrojna vrednost specificirane dužine. Kao rezultat operacije postavljaju se uslovni markeri N i Z, a V=C={0}. Zbog toga je testiranje operanada na vrednost negativan ili nula moguće odmah posle MOVE instrukcije.

Na primer, sekvenca

```
MOVE.W    DUZINA,D1
BEQ       SKOK
```

će usloviti grananje na instrukciju ispred koje stoji labela SKOK, ako je sadržaj lokacije DUZINA bio nula, jer se instrukcijom BEQ (Branch if Equal to Zero) obavlja grananje kada je Z={1}.

Primer 2.11:
Instrukcijom

MOVEM.W D0/D1/A2-A4,$11000

vrši se prenos LS sadržaja registara D0, D1, A2, A3 i A4 na lokacije $11000, $11002, $11004, $11006 i $11008, respektivno.

Kod korišćenja MOVEM treba se koristiti sledećom tabelom.

Tab. 2.7. Instrukcija MOVEM

(a) Lista registara	
(1) Odabrabi registri se odvajaju sa ″/″ (na primer D1/D3/D4)	
(2) Uzastopni registri se specificiraju sa ″-″ (na primer D1-D4)	
(b) Redosled prenosa	
Redosled	Tip
D0-D7, A0-A7 na više adrese	1. Registar u memoriju (različiti upravljački) 2. Memorija u registar
A7-A0, D7-D0 na niže adrese	Registar u memoriju (predekrementiranje)

2.6.2. Instrukcije za upravljanje tokom programa

Kod svakog programa postoji potreba za izborom sekvence instrukcija koja će se izvršavati na osnovu rezultata izračunavanja. Na taj način tok programa imaće različiti put kroz program u zavisnosti od rezultata ovih izračunavanja. U principu razlikujemo dva načina upravljanja tokom programa:

- bezuslovni,
- uslovni.

Kod uslovnih grananja testiraju se uslovni markeri sa ciljem da se odredi da li je uslov za promenu toka programa ispunjen ili ne.

Instrukcije za bezuslovno grananje

Instrukcijama BRA (Branch Always) i JMP (Jump) uslovljava se bezuslovni prenos upravljanja promenom vrednosti PC-a (Tabela 2.8). Nezavisno od toga da li se izvršava BRA ili JMP, adresa nove instrukcije mora biti parna, a ako to nije slučaj javiće se greška. JMP se razlikuje od BRA, jer se adresa skoka može specificirati različitim adresnim načinima rada, a adresa grananja je uvek relativna u odnosu na PC.

Tab. 2.8. Bezuslovno grananje i skok.

Sintaksa	Efekat	Adresni načini rada
BRA <disp>	(PC) = (PC) + <disp>	Relativni
JMP <EA>	(PC) = <EA>	Upravljački

Napomene:
1. U BRA <disp>, <disp> je označeni 8-bitni, 16-bitni ili 32-bitni ceo broj.
2. (PC) je adresa instrukcije BRA + 2.
3. Nema uticaja na kodove uslova.

Primer 2.12:
Efektivna adresa kod JMP instrukcije može se definisati registarskim indirektnim ili PC indirektnim adresiranjem. Kada je adresa rutine definisana baznom adresom koja se čuva u adresnom registru, instrukcijom

JMP (A1)

uslovljava se prenos upravljanja na instrukciju na koju ukazuje (A1).

Primer 2.13:
Razmeštaj ili indeksni registar je moguće koristiti za modifikaciju bazne adrese u adresnom registru, kao što je slučaj u sledećoj instrukciji

JMP (0,A1,D1.L*4)

Ovom instrukcijom vrši se grananje na adresu duge reči na koju ukazuje (A1) indeksiran sa (D1).

Instrukcije uslovnog grananja

Kod instrukcija Bcc moguć je izbor upravljačkog puta u programu u zavisnosti od uslova:

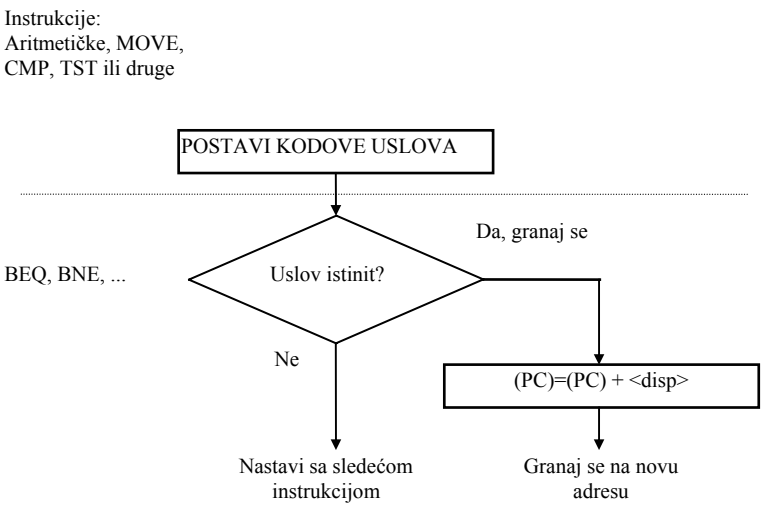
IF (uslov istinit)
THEN (grananje na novu sekvencu)
ELSE (izvrši narednu instrukciju)

Nova sekvenca instrukcija može biti na nižoj ili višoj memorijskoj adresi, relativno u odnosu na instrukciju grananja. Format instrukcije ima oblik

Bcc <label>

Način rada Bcc instrukcije prikazan je na slici 2.8.

Aritmetičke operacije kao i veći broj drugih instrukcija postavljaju uslovne markere u zavisnosti od rezultata pojedinih operacija. Ako je uslov istinit vrednost razmeštaja se dodaje tekućoj vrednosti PC-a. U tom trenutku u PC je smeštena adresa instrukcije uslovnog grananja plus 2. Lista mogućih uslova prikazana je na slici 2.9 zajedno sa uslovima koji uslovljavaju grananje.



Bcc <labela>

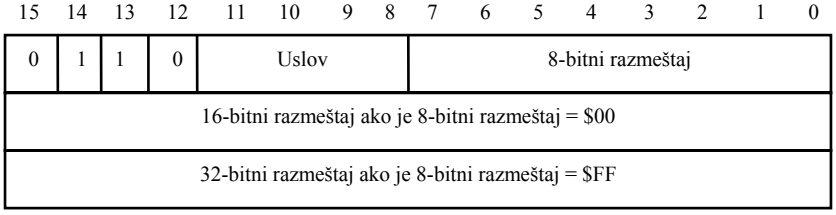
Sl. 2.8. Efekat Bcc instrukcije.

CC	prenos obrisan	0100	$\overline{C}$
CS	prenos postavljen	0101	C
EQ	jednako	0111	Z
GE	veće ili jednako	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	veće od	1110	$N \cdot V \cdot Z + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	više	0010	$\overline{C} \cdot \overline{Z}$
LE	manje ili jednako	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

LS	manje ili isto	0011	$C+Z$
LT	manje od	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	minus	1011	N
NE	nejednako	0110	$\overline{Z}$
PL	plus	1010	$\overline{N}$
VC	nema prekoračenja	1000	$\overline{V}$
VS	prekoračenje	1001	V

Nema uticaja na kodove uslova

Format instrukcije:



Sl. 2.9. Instrukcije uslovnog grananja.

Grananje ako je nula ili nije nula

Instrukcije uslovnog grananja BEQ i BNE su logički suprotne. Nakon što prethodna instrukcija postavi uslovne markere, instrukcijom

BEQ <labela>

obaviće se grananje ako je rezultat bio nula. Grananje ako rezultat (uslov) nije bio nula obaviće se instrukcijom

BNE <labela>

Grananje posle CMP ili TST

Instrukcije CMP (Compare) i TST (test) koriste se da postave markere uslova u zavisnosti od vrednosti operanda. Nakon izvršenja ove instrukcije moguće je koristiti instrukciju uslovnog grananja pomoću koje se usmerava dalji tok (izvršenje) programa.

Instrukcijom

CMP X,Y

porede se dva operanda na sledeći način

(X)-(Y)

i postavljaju se markeri uslova N, Z, V i C.

Instrukcijom

TST Y

vrši se procena jednog operanda na osnovu izračunavanja (Y)-0 tako da se uvek brišu V i C, ali se N i Z postavljaju u zavisnosti od rezultata. Obe instrukcije (CMP i TST) postavljaju markere uslova ali ne modifikuju operande.

U Tabeli 2.8 dat je pregled instrukcija CMP i TST.

Tab. 2.8. Instrukcije za poređenje i testiranje.

Instrukcija	Sintaksa	Adresni način rada		Efekat	Uticaj na kodeve uslova
		Izvor	Odredište		
Poredi	CMP.<l> <EA>,<Dn>	Svi	<Dn>	(Dn)-(EA)	N, Z, V, C
Poredi neposredno	CMP.<l> #<d>,<EA>	d	Za podatke	(EA)-d	N, Z, V, C
Poredi memoriju	CMPM.<l> (Am)+,(An)+	(Am)+	(An)+	((An))-((Am))	N, Z, V, C
Testiraj	TST.<l> <EA>	-	Za podatke	(EA)-0	N, Z, V={0}, C={0}

Napomene:
1. <l> označava B, W ili L.
2. Ako je An izvor za CMP, dozvoljeni su samo operandi obima reč (W) ili duga reč (L).

Kod instrukcije CMP postoje još dve varijante njenog izvođenja a to su CMPI (Compare Immediate) i CMPM (Compare Memory). Instrukcijom CMPI poredi se neposredna vrednost sa specificiranim operandom.

Primer 2.14:

Instrukcijom

CMPI.W #36,$1234

poredi se vrednost 36 sa sadržajem lokacije $1234.

Instrukcija CMPM se koristi za poređenje sekvenci bajtova, reči ili dugih reči koje su smeštene u memoriji. Za oba operanda dozvoljen je samo postinkrementirajući adresni način rada.

Instrukcijom

CMPM.B (A1)+,(A2)+

porede se bajtovi adresirani sa A1 i A2 a zatim se adrese povećavaju i ukazuju na naredne bajtove.

Imajući u vidu da u toku izvršavanja programa iza instrukcije CMP ili TST sledi instrukcija Bcc, u Tabeli 2.9 prikazan je pregled instrukcija Bcc nakon čijeg izvršenja se može obaviti grananje.

Tab. 2.9. Bcc instrukcije posle CMP i TST.

		Uslov grananja	
Instrukcija	Rezultat	Neoznačeni	Označeni
CMP X,Y	(Y) = (X)	BEQ (Jednako)	BEQ (Jednako)
	(Y) ≠ (X)	BNE (Nejednako)	BNE (Nejednako)
	(Y) > (X)	BHI (Više)	BGT (Veće)
	(Y) ≥ (X)	BCC (Prenos obrisan)	BCC (Veće ili jednako)
	(Y) < (X)	BCS (Prenos postavljen)	BCS (Manje)
	(Y) ≤ (X)	BLS (Niže ili isto)	BLS (Manje ili jednako)
TST X	(X) = 0	BEQ	BEQ
	(X) ≠ 0	BNE	BNE
	(X) > 0	BNE	BGT
	(X) < 0	-	BLT, BMI
	(X) ≥ 0	-	BGE, BPL
	(X) ≤ 0	-	BLE

Napomene:
1. U CMP X,Y: odredište je registar za podatake.
2. TST postavlja C={0} i V={0}.
3. BMI (Granaj se na minus) je isto kao i BLT, a BPL (Granaj se na plus) je isto kao i BGE kada je V={0}.

Instrukcija DBcc

Instrukcija koja obavlja testiranje, dekremetiranje i grananje predstavlja moćno sredstvo za upravljanje strukturama tipa petlja. Asemblerski format ove instrukcije je sledeći:

Dbcc <Dn>,<labela>

Ovom instrukcijom specificiraju se tri parametra: uslov "C", registar za podatke Dn i razmeštaj koji se predstavlja kao labela.

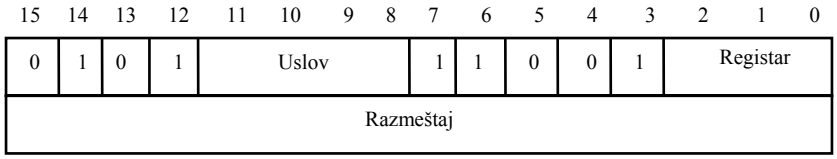
Različiti uslovi i format instrukcije prikazan je na slici 2.10.

CC	prenos obrisan	0100	$\overline{C}$
CS	prenos postavljen	0101	C
EQ	jednako	0111	Z
GE	veće ili jednako	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	veće od	1110	$N \cdot V \cdot Z + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	više	0010	$\overline{C} \cdot \overline{Z}$
LE	manje ili jednako	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

LS	manje ili isto	0011	$C+Z$
LT	manje od	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	minus	1011	N
NE	nejednako	0110	$\overline{Z}$
PL	plus	1010	$\overline{N}$
VC	nema prekoračenja	1000	$\overline{V}$
VS	prekoračenje	1001	V

Nema uticaja na kodove uslova

Format instrukcije:



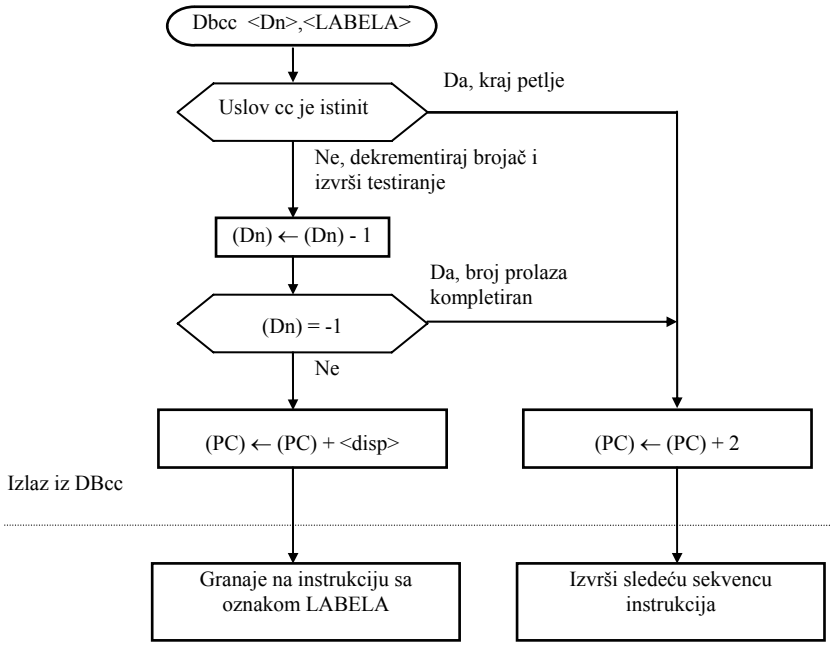
Sl. 2.10. Format instrukcije DBcc.

DBcc instrukcija uzrokuje da se izvršavanje petlje završi kada je bilo koji od specificiranih uslova:

- cc je istinit, ili
- kada broj koji se čuva u <Dn> postane -1.

Svaki put kada se instrukcija izvrši Dn se smanjuje za 1.

Dijagram toka instrukcije DBcc prikazan je na slici 2.11. Na dijagramu su prikazani uslovi koji uzrokuju izvršenje naredne instrukcije u programskoj sekvenci.



Sl. 2.11. Efekat DBcc instrukcije.

Instrukcijom DBcc može se testirati 14 logičkih uslova (isto i kod Bcc). Ipak instrukcija DBcc se ponekad zove instrukcija "Ne granaj se na uslov" (Don't Branch on Condition). Ako je uslov TRUE (istinit) ne vrši se grananje što je u suprotnosti sa načinom rada instrukcije Bcc.

Primer 2.15:

Struktura petlje kod koje DBcc je zadnja instrukcija u petlji, ima oblik

```
REPEAT
(telo petlje)
UNTIL (uslov)
```

Na primer, sekvenca instrukcija čija je struktura

```
LOOP...
        (telo petlje)
    TST         X           ; testiraj za ZERO
    DBEQ        <Dn>,LOOP   ; LOOP IF NOT ZERO
```

produžiće da se izvršava sve dok su sadržaji lokacije X nula ili sve dok brojač u <Dn> ne dostigne vrednost -1. TST instrukcijom postavljaju se markeri uslova u zavisnosti od vrednosti (X). DBEQ testira uslov i smanjuje (Dn) ako je uslov FALSE ali ne utiče na postavljanje markera uslova. Kada je specificiran uslov TRUE ili kada <Dn> sadrži -1, petlja se završava.

Pored 14 uslova testiranja koji se koriste od strane DBcc i Bcc instrukcija, kod DBcc postoje dva druga uslova, TRUE (T) i FALSE (F).

DBT instrukcijom se nikad ne vrši grananje. Kod njene logičke suprotnosti uvek se vrši grananje, sve dok brojač ne dostigne vrednost -1. Instrukcijom DBF ne testira se uslov, a koristi se za zamenu sekvence "dekrementiraj i testiraj za nulu" koja se često koristi za završetak petlji. Ipak postoji jedna razlika. Kako brojač mora da primi vrednost -1 pre nego što se okretanje u petlji zaustavi, inicijalizacija vrednosti brojača mora da bude za jedan manja u odnosu na broj potrebnih iteracija.

2.6.3. Korišćenje potprograma kod MC68020

Potprogram je sekvenca koja se tretira kao izdvojeni programski modul u okviru većeg programa. Potprogram se može pozivati ili izvršavti jedan ili veći broj puta u toku izvršenja programa. U opštem slučaju, potprogrami koji su pridruženi programu obavljaju specifične zadatke, od kojih svaki predstavlja jednostavnu proceduru. U suštini, potprogrami kod Pascala se zovu procedure. Svaki modul ili potprogram treba da predstavlja jedinstvenu celinu koja se može nezavisno testirati. Kada se u toku izvršenja pozove potprogram, ona se izvršava a upravljanje se zatim vraća narednoj instrukciji u programskoj sekvenci koja sledi iza poziva potprograma. Lokacija prve instrukcije potprograma se zove startna adresa. Ona se mora specificirati u svakom programu koji poziva potprogram. Ako se potprogram i program iz koga se vrši poziv potprograma asembliraju u isto vreme, početna adresa potprograma se može definisati labelom ispred svoje prve instrukcije. Ako se potprogram ili program iz koga se vrši poziv se ne asembliraju zajedno, početna adresa potprograma mora se eksplicitno definisati u instrukciji poziva. Ako potprogram ima nekoliko ulaznih tačaka, svaku adresu je potrebno specificirati. Takođe, kada se nezavisni programi zajedno asembliraju, "external" ili "global" obraćanja se obično definišu kada se vrši linkovanje (spajanje) svih modula.

Kod mikroprocesora MC68020 skup instrukcija koje se koriste za poziv ili povratak iz potprograma prikazan je u Tabeli 2.10.

Tab. 2.10. Instrukcije za rad sa potprogramima.

Instrukcija	Sintaksa	Efekat	Komentari
Granaj se na potprogram	BSR <disp>	1. (SP)←(SP)-4; ((SP))←(PC) 2. (PC)←(PC)+<disp>	<disp> je 8-bitni, 16-bitni ili 32-bitni osnačebi ceo broj
Skoči na potprogram	JSR <EA>	1. (SP)←(SP)-4; ((SP))←(PC) 2. (PC)←<EA>	<EA> je upravljački adresni način rada
Povratak i obnavlajnje kodova uslova	RTR	1. (CCR)←((SP))[7:0]; (SP)←(SP)+2 2. (PC)←((PC)); (SP)+4	(CCR) = (SR)[7:0]
Povratak iz potprograma	RTS	(PC)←((SP)); (SP)←(SP)+4	-

Napomene:
1. SP označava ukazatelj na sistemski magacin.
2. CCR je registar kodova uslova, tj. (SR)[7:0].
3. PC je programski brojač.

Kod instrukcije

BSR <labela>

operand <labela> uzrokuje da asembler izračuna razmeštaj između BSR instrukcije i instrukcije identifikovane sa <labela>. Razmeštaj se dodaje u toku izvršenja tekućem sadržaju programskog brojača (BSR lokacija plus 2) da bi se izračunala početna lokacija potprograma. Razmeštaj se smešta kao celobrojna vrednost u notaciji dvoičnog komplementa. BSR radi na isti način kao i instrukcija BRA sa izuzetkom što se adresa instrukcije koja sledi nakon BSR smešta u magacin. Na sličan način JSR je identična sa JMP sa izuzetkom što se povratna adresa čuva u magacinu. Opseg adrese skoka kod JSR je neograničen i bilo koji adresni način rada se može koristiti.

Primer 2.16:

Instrukcija

JSR 4(A5)

koristi indirektno adresiranje sa razmeštajem, i uslovljava skok na instrukciju koja je četiri bajta nakon adrese u A5.

Odgovarajuće instrukcije koje se koriste za povratak iz potprograma su RTR i RTS. RTR je u paru sa JSR, a RTS sa BRS.

2.7. Aritmetičke instrukcije za rad sa celobrojnom i BCD aritmetikom

Mikroprocesor MC68020 podržava rad svih osnovnih tipova aritmetičkih operacija sa celobrojnim vrednostima nad operandima različitog obima. Neke od operacija omogućavaju direktno izračunavanje u decimalnom obliku.

2.7.1. Sabiranje i oduzimanje

Binarno sabiranje/oduzimanje obavlja se nad 8-, 16- i 32-bitnim operandima pomoću ADD/SUB instrukcije i njenim varijacijama kao što je prikazano u Tabeli 2.11.

Instrukcije EXT i EXTB koriste se za znakovno proširenje sa jedne dužine na drugu. Ove instrukcije su korisne kada se manipuliše sa operandima različitog obima.

Instrukcijom NEG formira se dvoični komplement specificiranog operanda. Kod svake instrukcije (Tabela 2.11) postoje ograničenja koja se odnose na dozvoljene adresne načine rada. U toku izvršenja svake instrukcije odredišni operand specificirane dužine se zamenjuje rezultatom.

Primer 2.17:

```
                ADD.B       D1,D5
```

zamenjuje (D5)[7:0] sa sumom (D5)[7:0]+(D1)[7:0]

Instrukcijom

```
                ADD.L       D1,(A1)
```

sabira se 32-bitni sadržaj registra D1 sa sadržajem lokacije adresirane sa A1.

Instrukcijom

```
                ADDI.B #20,(A1)
```

sabira se 20 sa bajtom adresiranim od strane A1.

Instrukcijom

```
                SUBQ,W      #1,A1
```

oduzima se 1 od sadržaja A1. Obe ADDQ i SUBQ imaju efekat nad 32-bitnim sadržajem An registra.

Instrukcijom

```
                EXT.W       D1
```

proširuje se ili kopira bit [7] odredišnog registra D1 u bitove [15:8].

Instrukcijom NEG obavlja se sledeće izračunavanje

0-(odredište)

pa se formira dvoični komplement. Na primer, ako lokacija $1000 sadrži vrednost 1, instrukcijom

```
                NEG.B       $1000
```

zamenju se vrednost sa $FF.

Tab. 2.11.

	Adreseni načini rada	
Sintaksa	Izvor	Odredište
Sabiranje ili oduzimanje		
ADD.<l> <EA>,<Dn>	Svi	Dn
SUB.<l> <EA>,<Dn>		
ADD.<l> <Dn>,<EA>	Dn	Memorijski
SUB.<l> <Dn>,<EA>		
ADDI.<l> #<d>,<EA>	#<d>	Za podatke
SUBI.<l> #<d>,<EA>		
ADDQ.<l> #<d_3>,<EA>	#<d_3>	Različiti
SUBQ.<l> #<d_3>,<EA>		
Znakovno proširenje		
EXT.W <Dn>	(Dn)[7:0]	(Dn)[15:0]
EXT.L <Dn>	(Dn)[15:0]	(Dn)[31:0]
EXTB.L <Dn>	(Dn)[7:0]	(Dn)[31:0]
Promena znaka		
NEG.<l> <EA>	-	Za podatke

Napomene:
1. Ako je efektivna adresa izvora u instrukcijama ADD ili SUB neki od adresnih registara, tada je obim operanda reč ili duga reč.
2. <d_3> je vrednost između 1 i 8.
3. Ako je An odredište, samo su instrukcije koje rade sa rečima ili dugim rečima dozvoljene, pri čemu nema uticaja na kodove uslova.
4. <l> označava B, W ili L u svim instrukcijama osim kao u primedbama 1 i 3.
5. Osim kao u primedbi 3, aritmetičke instrukcije utiču na sve kodove uslova. EXT i EXTB postavljaju N i Z prema rezultatu, a brišu V i C.

Tab. 2.12.

Instrukcija	Uslov grananja	Rezultat
NEOZNAČENO ADD, SUB C={0}		
	BEQ	X = 0
	BNE	X ≠ 0
OZNAČENO ADD, SUB C={0}		
	BEQ	X = 0
	BNE	X ≠ 0
	BPL, BGE	X ≥ 0
	BLE	X ≤ 0
	BMI, BLT	X < 0
	BGT	X > 0

(a) Označeni i neoznačeni testovi.

Mnenonik	Uslov	Kodiranje	Test
T	tačno	0000	1
F	netačno	0001	0
HI	više	0010	$\overline{C} \cdot \overline{Z}$
LS	manje ili isto	0011	$C+Z$
CC (HS)	prenos obrisan	0100	$\overline{C}$
CS (LO)	prenos postavljen	0101	C
NE	nejednako	0110	$\overline{Z}$
EQ	jednako	0111	Z
VC	nema prekoračenja	1000	$\overline{V}$
VS	prekoračenje	1001	V
PL	plus	1010	$\overline{N}$
MI	minus	1011	N
GE	veće ili jednako	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	manje od	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	veće od	1110	$N \cdot V \cdot Z + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	manje ili jednako	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

(b) Testovi uslovnih kodova.

2.7.2. Instrukcije za množenje i deljenje

Skup instrukcija za množenje/deljenje celobrojnih vrednosti prikazan je u Tabeli 2.13. Kao što se vidi iz Tabele 2.13 kod MC68020 postoje posebne instrukcije za množenje/deljenje sa neoznačenim i označenim (dvoični komplement) celobrojnim vrednostima.

Množenje neoznačenih brojeva

Instrukcijom MULU.W obavlja se množenje dva neoznačena 16-bitna operanda, a dobija se 32-bitni rezultat. Na primer, instrukcijom

```
MULU.W    $10,D2
```

sa D2[15:0]=$0002, rezultira u (D2)=$0000 0020 ili 32 decimalno.

Množenje označenih brojeva

Kada se množe označene celobrojne vrednosti, rezultat je pozitivan ili negativan i zavisi od znaka operanda.

Instrukcijom

MULS.W #-1,D2

sa (D2)[15:0]=$0002, dobiće se u (D2)=$FFFF FFFF ili -2 u dvoičnom komplementu.

Deljenje neoznačenih brojeva

Instrukcijom DIVU.W obavlja se deljenje

Y/W=Q+R/W

gde je 32-bitna neoznačena celobrojna vrednost, W je 16-bitna neoznačena vrednost, Q je 16-bitni količnik, a R je 16- bitni ostatak.

Na primer, instrukcijom

DIVU.W #2,D1

deli se 32-bitni operand u D1 sa 2. Ako je pre izvršenja instrukcije (D1)=$0000 0005 nakon izvršenja instrukcije dobiće se (D1)=$0001 0002.

U toku operacije deljenja javljaju se dva specijalna slučaja:

- deljenje nulom, ili
- premašaj količnika.

Oba ova uslova smatraju se greškom. Kod ovakvog slučaja procesor prekida svoj rad i prelazi se na izvršenje izuzetnog uslova, tj. prelazi na izvršenje trap rutine "deljenje nulom".

Deljenje označenih brojeva

Instrukcija DIVS.W se izvršava na isti način kao i DIVU.W, ali su operandi označene celobrojne vrednosti.

Instrukcijom

DIVS.W #3,D1

sa (D1)=$FFFF FFF6 izračunava se -10/3 kao rezultat (D1)=$FFFF FFFD ili Q=-3 a R=1.

Tab. 2.13. Instrukcije za deljenje i množenje.

Sintaksa	Efekat
Množenje	
MULU.W <EA>,<Dn>	(Dn)[31:0]←(Dn)[15:0]*(EA)[15:0]
MULS.W	
MULU.L <EA>,<Dn>	(Dn)[31:0]←(Dn)[31:0]*(EA)[31:0]
MULS.L	
Deljenje	
DIVU.W <EA>,<Dn>	(Dn)[31:0]/(EA)[15:0]
DIVS.W	(Dn)[15:0]←količnik
	(Dn)[31:16]←ostatak
DIVU.L <EA>,<Dn>	(Dn)[31:0]/(EA)[15:0]
DIVS.L	(Dn)[31:0]←količnik

Napomene:

1. Za izvornu adresu <EA> dozvoljeni su zamo adresni načini rada za podatke (tj. An je zabranjen).
2. MULU.L i MULS.L izazivaju da je V={1} ako je proizvod veći od 32 bita.
3. Kod instrukcija deljenja, deljenje nulom izaziva *trap*; prekoračenje se iskazuje sa V={1}.
4. Kod označenog deljenja, ostatak ima znak deljenika.

2.7.3. Aritmetika sa celobrojnim vrednostima u višestrukoj preciznosti

Kod merenja koja se sprovode u nauci termin tačnost se odnosi na korektnost merenja, a preciznost se odnosi na iznos detalja koji se koriste za predstavljanje rezultata. Kod numeričkih vrednosti, iznos preciznosti se obično izražava zadavanjem broja značajnih cifara u numeričkoj vrednosti. Ako se za neku veličinu oceni da nema dovoljnu preciznost za datu aplikaciju, potrebno je koristiti dodatne značajne cifre da bi se generisao precizniji rezultat.

Aritmetičke jedinice kod mikroprocesora operišu sa maksimalno m cifara u toku izvršenja aritmetičkih operacija. Ova maksimalna dužina se zove jednostruka preciznost. Kod MC68020 maksimalna jednostruka preciznost je 32 bita. Sekvence većih dužina se ne mogu obrađivati kao jedinstveni aritmetički operandi od strane procesora. Zbog toga, da bi proširili preciznost, nekoliko m-cifarskih operanada se matematički mogu posmatrati kao jedinstvena vrednost. Ako se kombinuje K operanada, vrednost će biti dugačka $K \times m$ cifara. Kod aritmetike u dvostrukoj preciznosti je K=2, tj. $2 \times 32 = 64$ bita. Aritmetičke operacije nad operandima višestruke preciznosti se obavljaju korišćenjem procesorskih instrukcija nad svakim m-cifarskim delom vrednosti, a nakon toga se vrši kombinovanje rezultata.

Kod MC68020 postoje specijalne instrukcije pomoću kojih se izvode operacije +, -, /, * i operacija negacija nad celobrojnim vrednostima u dvostrukoj preciznosti.

Sabiranje i oduzimanje

Instrukcije ADDX (Add with Extend), SUBX (Subtract with Extend) i NEGX (Negate with Extend) definisane su u Tabeli 2.14. Razlika između instrukcija koje koriste proširenje i instrukcije za +, -, /, * i negaciju, koje smo već proučili, ogleda se u korišćenju markera uslova X i Z od strane instrukcija proširenja.

Tab. 2.14. Aritmetičke instrukcije sa proširenjem.

	Adresni načini rada	
Sintaksa	Izvor	Odredište
Sabiranje ili oduzimanje sa proširenjem		
ADDX.<l> <Dm>,<Dn>	<Dm>	<Dn>
SUBX.<l> <Dm>,<Dn>		
ADDX.<l> -(Am),-(An)	Predekrementni	Predekrementni
SUBX.<l> -(Am),-(An)		
Promena znaka sa proširenjem		
NEGX.<l> <EA>	-	Za podatke

Napomena: <l> označava B, W ili L.

Kao što je prikazano u Tabeli 2.15 instrukcije proširenja koriste X (Extend bit) u toku izvršenja ovih operacija. Ako je bit X bio postavljen u toku prethodne operacije, instrukcije ADDX, SUBX i NEGX uzimaju to u obzir kada se izvršavaju. Osnovna namena X bita je da doda prenos (ADDX) ili oduzme pozajmica (SUBX) kada se izračunava viših m cifara vrednosti u duploj preciznosti.

Tab. 2.15. Efekat instrukcija sa proširenjem.

ADDX.<l>	<src>,<dst>	(dst)[l]←src[l] + (dst)[l] + X
SUBX.<l>	<src>,<dst>	(dst)[l]←dst[l] - (src)[l] - X
NEGX.<l>	<EA>	(EA)[l]←0 - (EA)[l] - X

Napomene:
1. C, N i V markeri se postavlajju kao i kod ostalih aritmetičkih instrukcija.
2. Z je obrisan ako je rezultat različit od nule, inače se ne menja.
3. X se postavlja na isti način kao i C bit.
4. <l> označava B, W ili L.
5. [l] ukazuje na odgovarajuće bitove u instrukciji.

Primer 2.18:

Sekvencom instrukcija

```
ADD.L   D1,D3
```

ADDX.L D2,D4

vrši se sabiranje 64-bitne vrednosti D2/D1 sa 64-bitnom vrednosti u D4/D3. X bit je postavljen na {1} ako se sabiranjem LS dela (D1 i D3) javi prenos. Drugom instrukcijom dodaje se vrednost prenosa sumi.

Sekvencom instrukcija

NEG
NEGX

obavlja se negacija celobrojne vrednosti u duploj preciznosti.

Množenje

Instrukcije MULU.W i MULU.L i njima odgovarajuće MULS.W i MULS.L generišu 32-bitni rezultat. MULU.L i MULS.L mogu usloviti premašaj ako proizvod dva 32-bitna operanda premašuje 32-bitnu dužinu. U ovom slučaju, potrebna su 64 bita, pa se zbog toga kod MC68020 koriste instrukcije prikazane u Tabeli 2.16.

Tab. 2.16. Instrukcije za 64-bitno množenje.

Sintaksa	Eefekat
Označeni	
MULS.L <EA>,<Dh>:<Dl>	(Dh)[31L0]:(Dl)[31:0]←(<EA>)*(D1)[31:0]
Neoznačeni	
MULU.L <EA>,<Dh>:<Dl>	

Napomene:
1. <Dh> i <Dl> predstavljaju proizvoljni par registara za podatke.
2. <EA> je specificirano proizvoljnim adresnim načinom rada.
3. <EA> sadrži 32-bitni operand a (Dl)[31:0] sadrži drugi operand. Nižih 32 bita proizvoda nalaze se u (Dl) a viših 32 bita u (Dh).

Primer 2.19:

Instrukcijom

MULS.L D0,D6:D7

množi se (D0)*(D7), a smešta 64-bitni rezultat u (D6):(D7).

Deljenje

Instrukcije mikroprocesora MC68020 koje se koriste za deljenje označenih i neoznačenih brojeva imaju dve forme koje omogućavaju izračunavanje 32-bitnog količnika sa 32-bitnim ostatkom. Kao što je prikazano u Tabeli 2.17, jedna forma (DIVU.L, DIVS.L) omogućava da se 64-bitni deljenik podeli sa 32-bitnim deliteljem i da se pri tome dobije 64-bitni rezultat. Ostale instrukcije (DIVSL.L i DIVUL.L) dele 32-bitne operande i generišu 64-bitni rezultat. Kao i kod ostalih instrukcija deljenja, deljenje nulom generiše trap. Takođe ako se količnik ne može izraziti kao 32-bitna vrednost javiće se premašaj (V={1}).

Tab. 2.17. Instrukcije za deljenje u višestrukoj preciznosti.

Sintaksa	Efekat
DIVS.L <EA>,<Dr>:<Dq>	(Dq)[31:0]:(Dr)[31:0]=
DIVU.L <EA>,<Dr>:<Dq>	(Dr)[31:0]:(Dq)[31:0] / (< EA >)[31:0]
DIVS.L <EA>,<Dr>:<Dq>	(Dq)[31:0]:(Dr)[31:0]=
DIVU.L <EA>,<Dr>:<Dq>	(Dq)[31:0] / (< EA >)[31:0]

Napomene:
1. <Dq> sadrži nižih 32 bita deljenika pre deljenja. 32-bitni količnik se smešta u taj registar posle deljenja.
2. <Dr> sadrži viših 32 bita 64-bitnog deljenika i 32-bitni ostatak posle deljenja.
3. Prekoračenje se javlja ako je količnik veći od 32-bitnog celog broja. Deljnje nulom izaziva trap.

4. <EA> se može specificirati proizvoljnim adresnim načinom rada osim adresno registarskog direktnog.

2.7.4. Decimalna aritmetika

U skupu instrukcija mikroprocesora MC68020 postoji instrukcija za aritmetičke operacije nad decimalnim vrednostima predstavljenim u BCD kodu. Tri instrukcije za BCD aritmetiku definisane su u Tabeli 2.18.

Instrukcijama je moguće realizovati operacije +, - i negacija BCD vrednosti. Za svaku instrukciju dužina operanada je 8 bitova, što predstavlja dve BCD cifre.

Primer 2.19:

Instrukcijom

ABCD D1,D2

vrši se decimalno sabiranje operanada tipa bajt.

Instrukcijom NBCD formira se desetični komplement dvocifarskog operanda kada je X={0}, a devetični kada je X={1}.

Tab. 2.18. Instrukcije decimalne aritmetike.

Sintaksa	Efekat
Sabiranje	
ABCD <Dm>,<Dn>	(Dn)[7:0]←(Dn)[7:0] + (Dm)[7:0] + X
ABCD -(Am),-(An)	(dest)←(dest) + (src) + X
Oduzimanje	
SBCD <Dm>,<Dn>	(Dn)[7:0]←(Dn)[7:0] - (Dm)[7:0] - X
SBCD -(Am),-(An)	(dest)←(dest) - (src) - X
Promena znaka	
NBCD <EA>	(EA)←0 - (EA) - X

Napomene:
1. Sve operacije izvršavaju decimalnu aritmetiku nad dve BCD cifre.
2. N i V markeri su nedefinisani.
3. C se postavlja ako se javi decimalni prenos (ili pozajmica).
4. Z je obrisano ako je rezultat različit od nule, inače je nepromenjen.

Višestruka preciznost kod decimalne aritmetike

Operacije nad BCD brojevima sa više od dve cifre normalno se obavljaju nad operandima koji se čuvaju u memoriji, a ne u registrima. To je zbog toga što ABCD i SBCD mogu da operišu samo nad LS bajtom registra Dn. Ako se decimalni niz cifara čuva u registru Dn za manipulaciju je potrebno koristiti instrukcije rotiranja. Kada se podatak čuva u memoriji tada se koristi adresiranje sa predekrementiranjem.

Primer 2.20:

Instrukcijom

ABCD -(A1),-(A2)

vrši se prvo dekrementiranje za 1 registra A1 a zatim registra A2. Zatim se dva broja na adresiranim bajt lokacijama plus vrednost X bita sabiraju, a rezultat smešta u odredišnu lokaciju adresiranu sa A2.

Decimalni nizovi sa više od dve cifre smeštaju se u memoriju, počev od MS cifara prema LS ciframa. Decimalni broj 123456 na lokaciji $1000 smešta se na sledeći način

(1000)=12
(1001)=34
(1002)=56

Sabiranje/oduzimanje treba da počne sa inicijalne adrese $1003 i da se pri tome koristi adresiranje sa predekrementiranjem.

2.8. Konverzija podataka i instrukcije PACK i UNPK

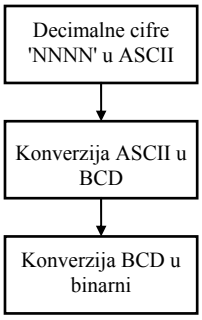
Kod MC68020 podaci se standardno predstavljaju kao binarni, ili BCD kada se govori o celobrojnim vrednostima, a za znake (karaktere) koristi se ASCII kodiranje. ASCII kodiranje se obično koristi za podatke koji se

predaju/primaju ka/iz perifernih uređaja. Konverzija između ovih reprezentacija veoma često se koristi jer se aritmetička obrada u procesoru vrši u binarnoj ili BCD prezentaciji.

Na slici 2.12a) prikazani su tipični koraci za konverziju decimalnih brojeva u ASCII, a zatim u binarnu prezentaciju. ASCII znaci za decimalne cifre se prvo konvertuju u binarne brojeve u opsegu 0-9. Po četiri bita za svaku cifru su takođe BCD vrednosti u memoriji. Zatim se niz BCD cifara konvertuje u binarni broj. U ovom slučaju mora se voditi računa o pozicionoj vrednosti svake BCD cifre. Na primer, ASCII niz "123" kao ulazna vrednost smešta se u memoriji kao $31, $32 i $33. Ovaj se zatim konvertuje u tri BCD cifre 1, 2 i 3, a zatim u binarnu vrednost 01111011.

Za izlazne binarne vrednosti koje treba štampati kao decimalne brojeve neophodno je izvršiti suprotnu konverziju od binarne u BCD, a zatim u ASCII. Binarna vrednost iz prethodnog primera je bila 7B, a njen ekvivalent u ASCII kod je $37 i $42.

Na slici 2.12b) prikazani su ASCII ekvivalenti za decimalne i heksadecimalne cifre.



a) Konverzija ulaznih podataka

Značenje	Binarna reprezentacija u memoriji	Za konverziju u ASCII	ASCII u memoriji
Decimalne cifre 0-9 (BCD)	0000 0000 0000 0001 ⋮ 0000 1001	Dodaj 0011 0000_2 ($30)	$30 - $39
Heksadecimalne cifre 0-9	0000 0000 0000 0001 ⋮ 0000 1001	Dodaj $30	$30 - $39
Heksadecimalne cifre A-F	0000 1010 0000 1011 ⋮ 0000 1111	Dodaj $37	$41 - $46

b) Konverzija decimalnih ili heksadecimalnih u ASCII radi izlaza

Sl. 2.12. Konverzija vrednosti podataka.

Konverzija između ASCII i BCD koristeći instrukcije PACK i UNPK

U Tabeli 2.19 prikazana je sintaksa i aktivnost instrukcija PACK i UNPK.

PACK instrukcija se prvenstveno koristi za konverziju izvornog operanda od dvobajtnog (dva karaktera) ASCII niza u pakovani BCD kod. Rezultat se može direktno koristiti od strane instrukcije za BCD aritmetiku.

Instrukcija

```
PACK        D2,D3,{000
```

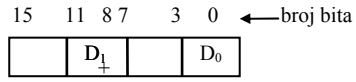
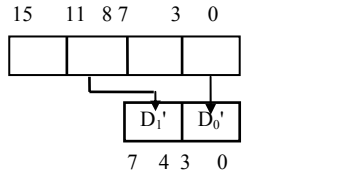
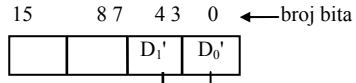
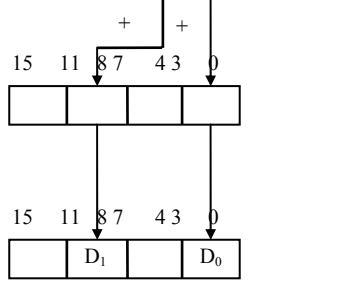
prihvata četiri LS bita svakog bajta izvorne reči i konvertuje ih u odredišni bajt, konverzija iz ASCII u BCD je završena ako D2 sadrži dva ASCII broja predstavljena kao 3X3Y gde su X i Y decimalne cifre.

Instrukcija

```
UNPK        D2,D3,#3030
```

konvertuje dve BCD cifre memorisane kao 00XY u registru D2, u vrednost 3X3Y u registru D3. Ako se umesto podataka $3030 koristio drugi podatak, BCD cifre konvertovaće se u oblik koji je različit od ASCII. Na primer, koristeći $0000 kao vrednost podatka, dvocifrena BCD vrednost konvertovaće se u nepakovani BCD.

Tab. 2.19. Instrukcije PACK i UNPK.

Instrukcija	Efekat
PACK -(Am), -(An),#<data>	
PACK <Dm>,<Dn>,#<data>	
UNPK -(Am), -(An),#<data>	
UNPK <Dm>,<Dn>,#<data>	

Napomene:
1. <data> je 16-bitni osnačeni ceo broj.
2. Kada su operandi u memoriji, najviši bajt od <Src> mora da bude na nižoj memorijskoj adresi.
3. Am i An su proizvoljni adresni registri. Dm i Dn su proizvoljni registri za podatke

2.9. Logičke operacije i operacije nad bitovima

Logičke operacije tretiraju operand kao skup izdvojenih logičkih promenljivih. U ovu kategoriju spadaju instrukcije AND, OR, EOR i NOT. Drugom kategorijom instrukcija ASL, ASR, LSL i LSR pomeraju se bitovi u okviru operanda (BTST, BSET, BCLR i BCHG). Druge dve instrukcije prikazuju rezultat uslovnog testa koji se izvodi modifikacijom indikator promenljive nazvane marker. To su instrukcije Scc i TAS.

2.9.1. Logičke operacije

Kod određenih aplikacija pogodno je tretirati svaki bit operanda kao individualnu logičku promenljivu. Ako su X i Y logičke promenljive, u Tabeli 2.20 prikazana je istinitosna tablica kojom se definišu odgovarajuće logičke operacije.

Tab. 2.20. Rezultati logičkih operacija.

x	*y*	NOT *x*	*x* AND *y*	*x* OR *y*	*x* EOR *y*
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Napomena: *x* i *y* su logičke promenljive. Resultati za svaku operaciju su definisani istinitosnom tablicom tih operacija. Na primer, (*x* OR *y*) je tačno ili {1} ako bar jedna od promenljivih *x* i *y* ima vrednost {1}.

U Tabeli 2.21. data je lista logičkih instrukcija koja prikazuje asemblersku sintaksu i adresni način rada za svaku instrukciju.

Tab. 2.21. Instrukcije za logičke operacije.

Sintaksa	Adresni načini rada	
	Izvor	Odredište
Logičko I		
AND.<l> <EA>,<Dn>	Za podatke	<Dn>
AND.<l> <Dn>,<EA>	<Dn>	Memorijski
ANDI.<l> #<d>,<EA>	<d>	Za podatke
Logičko ILI		
OR.<l> <EA>,<Dn>	Za podatke	<Dn>
OR.<l> <Dn>,<EA>	<Dn>	Memorijski
OR.<l> #<d>,<EA>	<d>	Za podatke
Isključivo ILI		
EOR.<l> <Dn>,<EA>	<Dn>	Za podatke
EORI.<l> #<d>,<EA>	<d>	Za podatke
Negacija		
NOT.<l> <EA>	-	Za podatke

Napomene:
1. <l> označava B, W ili L.
2. <d> je 8-, 16- ili 32-bitna logička promenljiva u obliku neposredne vrednosti.
3. Markeri C i V su uvek obrisani, N i Z se postavljaju na osnovu rezultata.
4. Odredišna lokacija se modifikuje na osnovu rezultata.

U Tabeli. 2.22 prikazani su neki primeri korišćenja logičkih instrukcija.

Tab. 8.22. Primeri logičkih operacija.

Instrukcija	Operandi	Rezultat
ANDI.B #$F0,D1	{1111 0000}	(D1)[7:0] = {1101 0000}
	AND {1101 0001}	
ORI.B #03,D1	{0000 0011}	(D1)[7:0] = {1101 0011}
	OR {1101 0001}	
NOT.B D1	NOT {1101 0001}	(D1)[7:0] = {0010 1110}
EOR.B D1,D2	{1101 0001}	(D2)[7:0] = {0000 0100}
	EOR {1101 0101}	

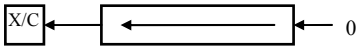
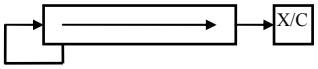
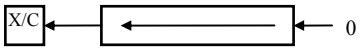
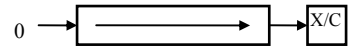
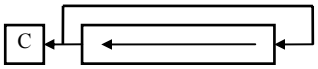
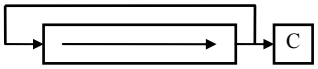
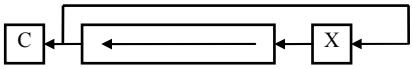
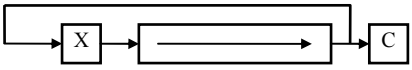
Napomena: (D1)[7:0] = {1101 0001} i (D2)[7:0] = {1101 0101} pre izvršenja svake instrukcije.

2.9.2. Instrukcije za pomeranje i rotiranje

Kod instrukcija za pomeranje i rotiranje vrši se pomeranje bitova u operandu na levo ili desno za određeni broj mesta. U principu postoje tri različita tipa ovih instrukcija:

a) aritmetičko pomeranje,
b) logičko pomeranje, i
c) rotiranje.

Na slici 2.13 prikazan je efekat operacija aritmetičkog pomeranja (ASL, ASR), logičkog pomeranja (LSL, LSR), instrukcije za rotiranje (ROL, ROR) i instrukcije za rotiranje ROXL i ROXR koje se koriste za manipulacije sa brojevima u višestrukoj preciznosti.

Instrukcija	Veličina operanda	Efekat
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	

Sl. 2.13. Instrukcije za pomeranje i rotiranje.

Sintaksa na asemblerskom jeziku ovih instrukcija prikazana je u Tabeli 2.23.

Tab. 2.23. Sintaksa instrukcija za pomeranje i rotiranje.

Aritmetičko pomeranje ulevo	Aritmetičko pomeranje udesno
ASL.<l> <Dm>,<Dn>	ASR.<l> <Dm>,<Dn>
ASL.<l> #<d>,<Dn>	ASR.<l> #<d>,<Dn>
ASL <EA>	ASR <EA>
Logičko pomeranje ulevo	**Aritmetičko pomeranje udesno**
LSL.<l> <Dm>,<Dn>	LSR.<l> <Dm>,<Dn>
LSL.<l> #<d>,<Dn>	RSR.<l> #<d>,<Dn>
LSL <EA>	RSR <EA>
Rotiranje ulevo	**Rotiranje udesno**
ROL.<l> <Dm>,<Dn>	ROR.<l> <Dm>,<Dn>
ROL.<l> #<d>,<Dn>	ROR.<l> #<d>,<Dn>
ROL <EA>	ROR <EA>

Napomene:
1. <l> označava B, W ili L kada je odredište <Dn>; <Dm> ili #<d> specificiraju broj pomeranja.
2. Kada je odredište memorijska lokacija dozvoljeni su jedino operandi dužine reči za <EA>.
3. Samo memorijski adresni načini rada su dozvoljeni za <EA>, isključujući registarski direktni i PC relativni adresni način rada.
4. ROXL i ROXR imaju istu sintaksu kao i instrukcije za rotiranje.
5. Markeri N i Z se postavljaju na osnovu rezultata, V je obrisan osim kod ASL.

Format opranada za opracije pomeranja i rotiranja prikazan je u Tabeli 2.24.

Tab. 2.24.

Format operanda	Broj pomeranja	Odredište
<Dm>,<Dn>	(Dm): opseg 0 - 63	(Dn)[l]
#<d>,<Dn>	#<d>: opseg 1-8	(Dn)[l]
<EA>	1	(EA)[15:0]

1. <l> označava B, W ili L kada je odredište <Dn>; <Dm> ili #<d> specificiraju broj pomeranja.
2. [l] ukazuje na odgovarajuće bitove u operaciji.
3. <EA> je memorijska adresa.
4. Nulti broj pomeraja u Dm ima posebno značenje.

Primer 2.21:
Instrukcijom

ANDI.W {00F,D1

svi bitovi registra D1 osim 4 LS bita se brišu

Primer 2.22:
Instrukcijom

LSR D3,D2

pomera se 16-bitni operand koji pripada LS reči registra D2 u desno za broj mesta specificiran sadržajem registra D3. Broj u D3 se tretira po modulu 64, tako da su moguća pomeranja samo od 0-63 mesta. Naravno, posle 16 logičkih pomeranja operanda tipa reč, levo ili desno, LS vrednost registra na svim mestima će imati upisane nula bitove.

Primer 2.23:
Instrukcijom

LSL.W #5,D3

obaviće se logičko pomeranje LS reči registra D3 za 5 mesta. Neposredno pomeranje se može specificirati u opsegu brojeva od 1 do 8.

Primer 2.24:
Instrukcijom

ASR (A2)

obaviće se pomeranje za jedno mesto u desno 16-bitnog operanda adresiranog od strane registra A2.

2.9.3. Instrukcije za manipulaciju bitovima i postavljanje markera

Kod velikog broja aplikacija koriste se logičke promenljive sa ciljem da ukažu na jedan od dva moguća rezultat operacije. Logičke promenljive koje se koriste na ovakav način zovu se markeri. Marker promenljiva ukazuje da se događaj (uslov) desio i koristi se kao sredstvo komunikacije sa rutinom koja testira ovu marker promenljivu. Kod MC68020 markeri se grupišu u okviru CCR-a. Stanje markera može ukazivati na to da li je neki uslov grananja ispunjen ili ne, na status perifernog uređaja (busy) i dr. Uslovi pod kojima marker treba postaviti ili obrisati mogu biti komplikovani, tj. mogu da sadrže veći broj test uslova. Kod MC68020 postoji posebna instrukcija Scc (Set According to Condition) koja omogućava promenljivoj koja se postavlja da ukaže na TRUE ili FALSE resultat u zavisnosti od vrednosti uslovnog koda (markera).

Instrukcije za manipulaciju bitovima

Instrukcije za manipulaciju bitovima operišu nad jedinstvenim bitom u Dn registru ili memoriji. Svaka instrukcija testira specificirani bit, pa u zavisnosti od njegove vrednosti postavlja Z marker uslova. Kao logička promenljiva, Z bit će se postaviti na komplementarnu vrednost specificiranog bita. Instrukcija BTST (Bit Test)

postavlja Z bit u zavisnosti od stanja tetsiranog bita. Instrukcije BCHG, BCLR i BSET postavljaju Z bit, a zatim uslovljavaju da se naznačeni bit menja, briše i postavlja na {1}, respektivno. Instrukcije za manipulaciju bitovima prikazane su u Tabeli 2.25.

Tab. 2.25.

Sintaksa	Efekat
Komplementiranje bita	
BCHG <Dn>,<EA>	Z = NOT(bn)
BCHG #<bn>,<EA>	THEN bn ← NOT(bn)
Brisanje bita	
BCLR <Dn>,<EA>	Z = NOT(bn)
BCLR #<bn>,<EA>	THEN bn ← {0}
Postavljanje bita	
BSET <Dn>,<EA>	Z = NOT(bn)
BSET #<bn>,<EA>	THEN bn ← {1}
Testiranje bita	
BTST <Dn>,<EA>	Z = NOT(bn)
BTST #<bn>,<EA>	

Napomene:
1. Ako je <Dn> odredište, obim operanda je 32 bita; inače, obim odredišnog operanda je bajt.
2. <bn> je broj bita operanda.
3. BTST dozvoljva sve adresne načine rada za odredište osim adresno registarskog direktnog.
4. BCHG, BCLR i BSET dozvoljavaju samo adresne načine rada za podatke za odredišni operand.

Primer 2.26:

Neka je (D1)=$0000 0001, a (D2)=$0000 88FF. Da bi specificirali bit poziciju 1 registra D2 koristićemo se instrukcijom

```
BCHG    D1,D2
```

kojom se postavlja Z={0}, a predstavlja komplement od (D2)[1]. Zatim operand u D2 postaje $0000 88FD, jer se bit 1 u D2 komplementira operacijom. Vrednost u D1 mora biti između 0 i 31.

Instrukcija

```
BCLR    #1,D2
```

ima isti efekat na D2.

Instrukcija Scc

Instrukcija Scc postavlja svih 8 bitova odredišne lokacije na {1} ako je uslov "cc" istinit. Ako uslov nije istinit svi bitovi su postavljeni na {0}. Uslovi (prenos obrisan, prenos postavljen, itd.) su isti kao i oni koji se koriste kod DBcc instrukcije.

Na primer, ako je marker Z={1}, instrukcijom SEQ D1 (Set if Equal to zero) upisuje se $FF u LS bajt registra D1.

Instrukcija Testiranje i Postavljanje (Test and Set)

TAS instrukcija se koristi za testiranje i modifikaciju operanda dužine bajt koji se čuva u Dn registru ili u memoriji. Njena aktivnost je prikazana na slici 2.14, a simbolički oblik instrukcije je

```
TAS     <EA>
```

Ako je operand nula, marker Z={1}, ako nije Z={0}. Ako je bit 7 operanda jednak {1}, N={1}. U tom pogledu TAS radi slično kao i TST (Test) instrukcija koja operiše nad vrednošću tipa bajt. U suštini, nakon ispitivanja od strane TAS instrukcije postavljaju se markeri N i Z, a MS bit operanda se postavlja na {1}.

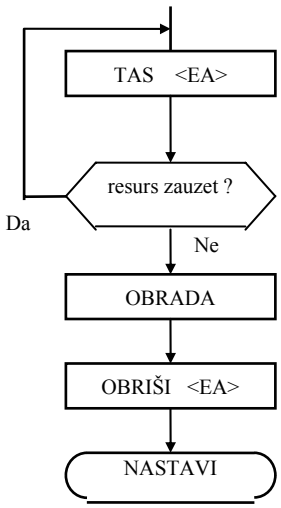
```
IF (EA) = 0 THEN
        postavi Z={1}
ELSE
        postavi Z={0}

IF (EA)[7]= {1} THEN
        postavi N={1}
ELSE
        postavi N={0}

Postavi (EA)[7] = {1}
```

Napomene:
1. (EA) je operand obima bajt adresiran adresnim načinom rada za podatke.
2. Ciklus čitanje-modifikacija-upis je nedeljiv.

(a) TAS operacija (TAS <EA>).



(b) Tipična upotreba.

Sl. 2.14.

Primer 2.27:
Ako bajt koji se koristi kao marker i adresira se od strane A1 ima inicijalnu vrednost $00, izvršenje instrukcije

```
TAS     (A1)
```

usloviće da Z={1} i promeniće operand na $80 nakon izvršenja instrukcije.

Primer 2.28:
Ako se marker promenljiva postavljena na $80, testira u petlji kao što je

```
LOOP:   TAS     (A1) ; testira marker
        BNE     LOOP ; grana se nazad ako nije obrisan
```

naredna instrukcija u sekvenci se ne može izvršiti sve dok neki drugi program ili procesor ne obriše marker. Kada je marker već postavljen, kao što je u ovom slučaju, TAS instrukcija ga ne menja.

2.9.4. Instrukcije za manipulaciju nad bit-poljima

Logičke operacije o kojima smo govorili operišu nad bitovima koji su deo operanda tipa bajt, reč ili duga reč. Individualni bitovi se adresiraju bit brojem u okviru operanada. Kada mora da se adresira grupa bitova proizvoljne dužine u okviru operanda sa ciljem da se izvrši njegovo testiranje ili obavi nad tom grupom aritmetička ili logička operacija, tada skup bitova mora da se izolira. Kod MC68020 postoji skup instrukcija pomoću kojih se vrši izolacija grupe bitova.

Ove instrukcije direktno operišu nad grupom kontinualnih bitova koji se zovu bit polje. Instrukcije tipa bit-polje u suštini testiraju bit-polje kao adresibilni operand. Na slici 2.15 prikazan je način definisanja adrese bita i bit-polja u memoriji. Jedinstveni bit se adresira zadavanjem bazne adrese njegovog bajta u memoriji i bit broja u okviru bajta startujući od LS bita u bajtu.

Vrednost bit-polja specificira se instrukcijom koja operiše nad sadržajem tog polja pomoću tri parametra, Specifikatori su sledeći:

a) bazna adresa bajta u memoriji - predstavlja prvi bajt niza bitova u kome se nalazi bit polje,
b) pozicija ili ofset koji je relativan u odnosu na baznu adresu MS bita u polju (bit[0]),
c) obim polja koji se zadaje kao broj kontinualnih bitova u polju.

Primer 2.29:

Na slici 2.15 prikazano je bit polje koje u odnosu na baznu adresu zauzima prostor od -24 bita prema adresama do +15 bitova prema višim adresama. Polje koje je naznačeno kao "bit field" čine tri bita definisana sledećim parametrima:

<ofset> = 20
<širina> = 3

Na slici 2.15, broj bita 1 u bit poljima ima vrednost

<ofset> + 1 = 19

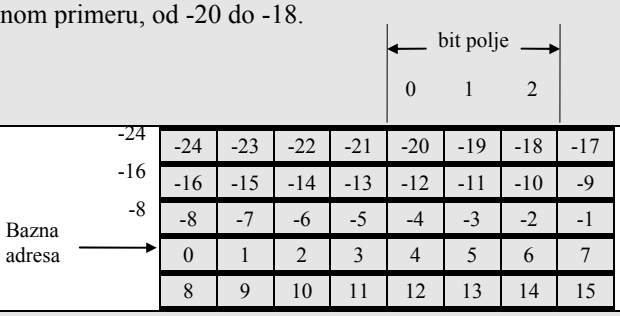
i predstavlja ofset (pomeraj) u odnosu na bazni bit celog polja. Celo polje zauzima prostor od

<bazna adrese>+<ofset>

do

<bazna adrese>+<ofset>+<širina-1>

ili, u konkretnom primeru, od -20 do -18.



Napomene:
1. Ofset bit polja = -20.
2. Širina bit polja = 3.
3. Pozicija bitova je indicirana decimalnim brojevima u odnosu na baznu adresu.

Sl. 2.15. Primer organizacije bit polja u memoriji.

Operacije sa bit-poljima

Instrukcije iz ove grupe, prikazane u Tabeli 2.26 mogu se podeliti u sledeće tri grupe:

a) one koje menjaju, brišu ili testiraju proizvoljno definisano bit-polje (bit field) u celom bit-polju (bit array),
b) one koje izbacuju ili insertuju proizvoljno definisano bit-polje u celom bit polju,
c) BFFF, pronalazi prvu (1) u bit polju u okviru celog bit-polja.

Tab. 2.26. Instrukcije za rad sa bit poljima.
(a) Sintaksa instrukcija

Sintaksa	Efekat
Testiraj bit polje i komplementiraj BFCHG <EA>{ofset:w}	Komplementira bit polje širine w
Testiraj bit polje i obriši BFCLR <EA>{ofset:w}	Briše bit polje širine w postavljajući svaki bit na {0}
Označeno izdvajanje bit polja BFEXTS <EA>{ofset:w},<Dn>	Izdvaja bit polje i kopira ga u <Dn>, desno poravnato i znakovno prošireno do 32 bita
Neoznačeno izdvajanje bit polja BFEXTU <EA>{ofset:w},<Dn>	Izdvaja bit polje i kopira ga u <Dn>, desno poravnato i prošireno nulama do 32 bita
Nađi prvu jedinicu u bit polju BFFFO <EA>{ofset:w},<Dn>	Pozicija prvog postavljenog bita ({1}) se smešta u <Dn>
Umetanje bit polja BFINS <Dn>,<EA>{ofset:w}	Kopira bit polje iz nižih bitova registra <Dn> u naznačeno bit polje u <EA>
Postavi bit polje BFSET <EA>{ofset:w}	Postavlja bit polje širine w; postavlja se svaki bit na {1}
Testiraj bit polje BFSET <EA>{ofset:w}	Testira se bit polje i na osnovu toga modifikuju vrednosti markera

Napomena: Bit pozicija u BFFO instrukciji je
<Dn> = <ofset> + (ofset do prvog bita jednakog {1})
ako je {1} nađena u bit polju. Inače, ako nema {1} u bit polju,
<Dn> = <ofset> + <w>

(b) Specifikacija instrukcija za rad sa bit poljima

Adresiranje <EA>	Adresni načini rada
BFCHG, BFCLR, BFINS, BFSET	Direktno registarsko za podatke i različiti upravljački načini rada, osim An, (An)+, -(An) i PC relativnog
BFEXTS, BFEXTU, BFFO, BFTST	Direktno registarsko za podatke i upravljački načini rada osim An, (An)+ i -(An)
Ofset i širina	
Ofset	(a) 0-31 kao neposredna vrednost (b) -2^{31} do 2^{31} - 1 u registru za podatke
Širina w	1-32 kao neposredna vrednost ili u registru za podatke

(c) Postavljanje markera

Instrukcija	Bitovi kodova uslova
BFXXX (sve)	V=C={0} X nepromenjen
BFCHG, BFCLR, BSET	N={1} ako je najviši bit polje jednak {1} *pre* izmene polja, inače je obrisan Z={1} ako su svi bitovi polja jednaki nuli *pre* izmene polja, inače je obrisan
BFEXTS, BFEXTU, BFFFO, BFSET	N={1} ako je najviši bit polje jednak {1}, inače je obrisan Z={1} ako su svi bitovi polja jednaki nuli, inače je obrisan
BFINS	N i Z se menjaju prema vrednosti *umetnutog* polja

Opšti oblik prve grupe je

BFXXX <EA>{ofset:w}

i važi za instrukcije BFCHG, BFCLR, BFSET i BFTST, u kojoj je w širina polja. Efektivna adresa <EA>, koja specificira baznu adresu celog bit-polja, može biti u Dn registru ili u memoriji.

Primer 2.30:

Instrukcijom

BFTST (A6) {D1:4}

specificira se bazna adresa u A6 pomoću An registarskog indirektnog adresiranja. Pomeraj se nalazi u D1, a bit-polje je širine četiri bita.

Instrukcije BFEXTS, BFEXTU i BFINS

Bit-polje se može izvući iz celog bit-polja pomoću instrukcija BFEXTS i BFEXTU. Instrukcija BFEXTS (Extract Bit Field Signed) se koristi kada je bit-polje označena celobrojna vrednost.

Primer 2.31:

Instrukcijom

BFEXTS (A0){0:14},D1

uzima se 14 bitova celog bit polja definisanih baznom adresom u A0 i prenose u bit-polje u D1 kao uređeni. Bit [0] bit polja (bit [13] u D1) se proširuje (kopira) u (D1)[31:14].

Nasuprot tome, instrukcija Extratct Bit Field Unsigned

BFEXTU (A0) {0:14}, D1

ima za efekat

(D1)[13:0]=bit-polje
(D1)[31:14]={0}

nezavisno od toga kakva je vrednost MS bita u polju.

Instrukcija BFINS (Insert Bit Field) kopira bit-polje iz jednog registra za podatke u drugi, ili u memoriju. Na taj način ona obavlja suprotnu aktivnost od BFEXTU.

Na primer, instrukcijom

BFINS D2,(A2){14:23}

kopira se bit-polje (D2)[22:0] u bit-polje koje počinje sa bit adresom (A2)+14 do

(A2)+14+(23-1)=(A2)+36

Uslovni marker Z se postavlja na {1} ako bit-polje u D2 sadrži sve nule. Ako je MS bit bit-polja u registru D2 (D2[22]) jednak {1}, uslovni marker N={1}. Inače Z={0} i N={0}, ako nijedan od uslova nije ispunjen.

Instrukcija BFFFO

Instrukcija BFFFO (Find First One in Bit Field) se koristi za pronalažanje pozicije prve jedinice u okviru bit-polja.

Na primer, instrukcijom

BFFFO (A2){6:11},D1

pretražuje se bit u polju širine 11 bitova za prvu {1}. Ako se pronađe {1} broj polja se smešta u D1. Ako je bit [n] u bit polju {1}, rezultat u D1 je

(D1)=6+[n]

gde je 6 pomeraj od bita 0 u celom bit-polju, a n=0,...10. Ako u bit polju ne postoji bit={1}, u odredišni registar se upisuje vrednost

(D1)=<ofset>+W

gde je W obim polja. U konkretnom primeru, ako je svaki bit u polju {0}, dobiće se

(D1)=6+11=17

a to predstavlja pomeraj bita u narednom bit-polju, celog bit-polja, koje sledi iza prethodnog bit-polja, pri čemu je početna adresa celog bit polja smeštena u A2. Vrednost u D1 se sada može koristiti kao ofset bit polja. Ako su uzastopna bit polja dužine W=11 nula, instrukcija

BFFFO (A2){D1:11},D1

se može koristiti kao indeks kroz bit polja kada celo bit-polje počinje sa adrese specificirane od strane (A2). Ako se instrukcija izvršava u petlji, D1 se menja na sledeći način

(D1): 17, 28, 39, ...

sve dok se ne naiđe na {1} u jednom od bit-polja. Kada se naiđe na {1} uslovni marker Z={0}. Uslovni marker N je {1} ako je MS bit (bit 0) bit-polja bio {1}.

2.10. Instrukcije koje manipulišu adresama

Sa izuzetkom vrednosti koje se čuvaju u registrima ili vrednosti koje se specificiraju neposrednim adresiranjem, operandima (mikroprocesora MC68020) se vrši obraćanje preko njihovih adresa. Kod MC68020 postoji poseban skup instrukcija koje operišu sa adresama kao što su CMPA, ADDA, SUBA, MOVEA, LEA i PEA. Za sve operacije koje uključuju manipulaciju adresama važeći adresni opseg (važi za MC68020) je od 0 do $FFFF FFFF. Kod instrukcija koje manipilišu operandima tipa reč 16-bitne adrese se znakovno proširuju na 32 bita pre nego što se one koriste od strane instrukcija. Zbog toga, kratke adrese (16-bitova) imaju važeći opseg od 0 do $7FFF, ili od $FFFF8000 do $FFFFFFFF.

2.10.1. Aritmetičko adresne manipulacije

Instrukcije ADDA (Add Address), SUBA (Subtract Address) i CMPA (Compare Address) operišu nad izvornim operandom koji se može adresirati bilo kojim adresnim načinom rada, ali se odredišni operand mora čuvati u An. Sintaksa ovih instrukcija data je u Tabeli 2.27.

Tab. 2.27. Efekat instrukcija ADDA, SUBA i CMPA.

Instrukcija	Sintaksa	Efekat	Uticaj na markere
Saberi adresu	ADDA.<l> <EA>,<An>	(An)←(An)+(EA)	Nema
Oduzmi adresu	SUBA.<l> <EA>,<An>	(An)←(An)-(EA)	Nema
Poredi adresu	CMPA.<l> <EA>,<An>	(An)-(EA)	N, Z, V i C

Napomene:

1. <l> označava jedino W ili L.
2. Ako je specificiran operand obima jedna reč (W), znakovno se proširuje do 32 bita.
3. Svi adresni načini rada su dozvoljeni za <EA>.

Primer 2.31:

Instrukcijom

```
ADDA.L    #20,A1
```

inkrementira se (A1) za 20.

Instrukcijom

```
ADDA.L    A2,A2
```

duplira se vrednost u A2.

Instrukcijom

```
SUBA.L    D1,A1
```

se smešta 32-bitna vrednost (A1)-(D1) u A1.

Kao i ADDA, tako i SUBA ne modifikuje markere uslova.

Instrukcija CMPA se koristi za upoređivanje dve adrese (manja, jednaka ili veća), ali se nakon njenog izvšenja postavljaju samo markeri važećih uslova. Adrese se smatraju neoznačenim celobrojnim vrednostima.

Primer 2.32:

Instrukcijama

```
CMPA.L    A1,A2     ; formira se (A2)-(A1)
BHI       LOOP      ; grananje ako je (A2)>(A1)
```

Instrukcijom CMPA izračunava se

(odredište)-(izvor)

a kao rezultat se postavljaju markeri uslova. U Tabeli 2.28 dat je pregled instrukcija uslovnog grananja koje su važeće nakon izvršenja instrukcije CMPA.

Tab. 2.28. Poređenje adresa.

Instrukcija	Grananje	Uslov
CMPA.L A1,A2	BHI (veće)	(A2) > (A1)
	BLS (manje ili isto)	(A2) ≤ (A1)
	BCC (veće ili isto)	(A2) ≥ (A1)
	BCS (manje)	(A2) < (A1)
	BNE (nejednako)	(A2) ≠ (A1)
	BEQ (jednako)	(A2) = (A1)

2.19.2. Instrukcije za prenos adresa

U ovu grupu spadaju instrukcije MOVEA, PEA i LEA. Sintaksa i adresni načini rada za ove tri instrukcije koje vrše prenos adresa prikazani su u Tabeli 2.29.

Tab. 2.29. Instrukcije za prenos adresa.

Instrukcija	Sintaksa	Dužina operanda (u bitovima)	Adresni načini rada	
			Izvor	Odredište
Kopiraj adresu	MOVEA.<l> <EA>,<An>	16 ili 32	Svi	An
Napuni efektivnu adresu	LEA <EA>,<An>	32	Upravljački	An
Smesti efektivnu adresu u stek	PEA <EA>	32	Upravljački	-(SP)

Napomene:

1. Nema uticaja na markere.
2. <l> označava jedino W ili L.
3. Za operande obima reč, izvorni operand se znakovno proširuje do 32 bita, a onda se svih 32 bita pune u adresni registar.

Primer 2.33:

Instrukcijom

```
MOVEA.L   TABELA,A1
```

kopira se 32-bitna vrednost lokacije TABELA u A1. Vrednost na lokaciji TABELA tretira se kao adresa.

Ako iza instrukcije MOVEA sledi instrukcija

MOVE.W (A1),D1

16-bitna vrednost kojoj se obraćamo pomoću adrese na lokacji TABELA se prenosi u LS reč registra D1. Pomoću pomenute dve naredbe obavlja se prenos tipa

(D1)[15:0]←((TABELA)[31:0])[15:0]

Instrukcija LEA

Instrukcija LEA se koristi za izračunavanje efektivne adrese na osnovu adresnog načina rada koji važi za izvor, i prenos te adrese u An. Registarski direktni, postinkrementirajući i predekrementirajući adresni načini rada nisu dozvoljeni.

Primer 2.34:

Instrukcijom

LEA TABELA,A1

rezultira u (A1)[31:0] ← TABELA, gde je TABELA adresa u asemblerskom programu.

LEA instrukcija omogućava da se adresa izračunava u toku izvršenja programa i prenosi u adresni registar. Tako na primer kod sledeće programske sekvence

```
LEA       (2,A1,D1.W),A0 ; izračunava se adresa
MOVE.W    (A0),D2        ; smešta vrednost u D2
MULU      #4,D2          ; 4*vrednost
MOVE.W    D2,(A0)        ; memorisanje
```

U konkretnom primeru prvo se izračunava adresa na osnovu indirektnog adresiranja sa indeksiranjem na sledeći način

(A1)+(D1)[15:0]+2

i ona smešta u A2. Operand tipa reč, adresiran od strane (A0), kopira se u D2, modifikuje i memoriše.

Instrukcija PEA

Ovom instrukcijom izračunava se efektivna adresa i smešta u magacin, a njena aktivnost se može izraziti kao

(SP)←(SP)-4
((SP))←<EA>

Primer 2.35:

U Tabeli 2.30 prikazani su rezultati nakon izvršenja instrukcija PEA, LEA i MOVEA. Kod PEA instrukcije pokazivač magacina je bio inicijaliziran na $0000 7FFF.

Tab. 2.30. Primeri adresnih manipulacija.

Sadržaj memorije (heksadecimalno)	Instrukcija	Rezultati
4850	PEA (A0)	($7FFA) = $00 ($7FFB) = $00 ($7FFC) = $10 ($7FFD) = $20
41F9	LEA $00012345,A0	(A0) = $00012345
0001		
2345		
307C	MOVEA.W]000,A0	(A0) = $FFFF8000
8000		

Napomena: Inicijalno je (A0) = $0000 a (SP) = $0000 7FFE.

2.11. Poziciono nezavisni programi

Kod najvećeg broja programa i primera koje smo do sada analizirali program se smešta na fiksne memorijske lokacije pri čemu je njegova početna adresa definisana direktivom ORG. Kada ove programe treba

premestiti u drugu memorijsku oblast, neophodno je obaviti njihovo ponovno asembliranje i pritom iznova specificirati njihov početak. Činjenica da se programi ne mogu premeštati ili realocirati u memoriji, bez ponovnog asembliranja, je nepovoljna u određenim situacijama.

Za program se kaže da je statički poziciono nezavisan, ako se on može puniti i izvršavati od bilo koje početne memorijske adrese. Najveći broj ROM programa su statički poziciono nezavisni, jer se početna adresa ROM programa definiše od strane sistemskog projektanta i zavisi od zahteva sistema. Na primer, FP rutina u ROM-u može imati početnu adresu kod jednog sistema na $2000, a kod drugog na $3000. Pisanje poziciono nezavisnih programa je tehnika kodiranja rutina tako da početna adresa bude nezavisna. Važna osobina ovih programa je da ne sadrže apsolutne adrese sa izuzetkom onih koje se odnose na adrese U/I uređaja. Dinamička poziciona nezavisnost omogućava da se ovi programi premeštaju i nakon što je njihovo izvršenje počelo.

2.11.1. Poziciono nezavisni program sa (PC)

Kada je lokacija u programu kojoj se obraćamo na fiksnom rastojanju od instrukcije koja se trenutno izvršava, tada da bi se kreirao poziciono nezavisni program, potrebno je koristiti PC relativni adresni način rada. Sve dok se relativni razmeštaj ne promeni, program se može korektno izvršavati nezavisno od toga gde je smešten u memoriji. Ako se kod svakog obraćanja memoriji koristi PC relativno adresiranje, program će se nezavisno, dinamički, pozicionirati jer se EA izračunava u toku izvršenja svake instrukcije. U Tabeli 2.31 prikazani su tipovi instrukcija i obraćanja memoriji koji su poziciono nezavisni.

Tab. 2.31. Poziciono nezavisne reference.

Kategorija	Adresni način rada
Instrukcije grananja	PC relativno sa razmeštajem
BRA	
Bcc	
DBcc	
Neposredne instrukcije	Neposredmo
Logičke (ORI, ANDI, EORI)	
Aritmetičke (ADDI, SUBI, CMPI, ADDQ, SUBQ)	
Kopiranja (MOVEQ)	
Apsolutna referenca na fiksnu lokaciju	Apsolutno dugo, apsolutno kratko
Relativne memorijske reference	PC relativno za razmeštajem, PC relativno sa indeksiranjem

Kao što je prikazano u Tabeli 2.32 najveći broj instrukcija mikroprocesora MC68020 koje specificiraju dva operanda omogućavaju samo da se za izvorni operand naznači PC relativni adresni način rada.

Primer 2.37:

Kod instrukcije

```
MOVE.W          X,Y
```

moguće je samo da se X specificira kao PC relativno.

Instrukcijom

```
MOVE.L          (PODATAK,PC),D1
```

kopiraju se 32-bita iz lokacije adresirane sa (PC)+PODATAK

(D1)=((PC)+PODATAK)

gde je podatak 8-, 16- ili 32-bitni razmeštaj sa znakom.

Instrukcijom

```
ADD.L           ([ADDR,PC]),D1
```

sabira se sa (D1) operand na koji ukazuje sadržaj lokacije (PC)+ADDR

(D1)=((PC+ADDR))+(D1)

Asemblerska direktiva

```
OPT             PCO
```

obezbeđuje PC relativno adresiranje kod obraćanja unazad u odnosu na labelu. Kao u slučaju instrukcije

```
MOVE.W          POLJE3,D0
```

kopira 16-bitnu vrednost na lokaciji (PC)+POLJE3 u D0, a kao rezultat imamo

(D0)[W]=((PC)+POLJE3)[W]

tj. obezbeđeno je da POLJE3 bude definisano kao labela pre nego što se to ostvaruje MOVE instrukcijom.

Obraćanja unapred su moguća korišćenjem direktive

OPT PCS

Tab. 2.32. Relativno adresiranje za instrukcije.

Izvor	Odreište[1]
ADD, ADDA	BRA, Bcc, DBcc
AND	
BFETS, BFEXTU, BFFFO, BFTST	BSR
CHK, CHK2	BTST
CMP, CMPA, CMP2	
cpRESTORE	CMPI
DIV, DIVSL, DIVU, DIVUL	JMP, JSR
LEA	TST
MOVE, MOVEA	
MOVE TO CCR	
MOVE TO SR	
MOVEM	
MULS, MULU	
OR	
PEA	
SUB, SUBA	

Napomene:
1. Odredište se ne menja ovim instrukcijama.
2. CALLM takođe dopušta PC relativno adresiranje.