QA32 - RA32 Assemblator *för MC68340 (MC68)*

Instruktionslista

ABCD "add decimal with extend"

Syntax:

```
ABCD        Dy,Dx
ABCD        -(Ay),-(Ax)
```

Beskrivning:
Källoperanden adderas till destinationsoperanden tillsammans med X-biten i CCR. Operationen utförs med användning av BCD-aritmetik. Operanderna kan anges på två olika sätt:

1. data register till data register: dvs operanderna finns i de dataregister som anges av instruktionen.
2. minnesadress till minnesadress: operandernas adress anges av det minskade innehållet i något adressregister.

Operandens storlek är alltid 8 bitar (*byte*).

Flaggor	
X	1 om carry (decimal) genererats, 0 annars
N	Odefinierad
Z	0 om resultatet är skilt från 0, påverkas annars inte
V	Odefinierad
C	1 om carry (decimal) genererats, 0 annars

ADD "add binary"
ADDA "add address"
ADDI "add immediate"
ADDQ "add quick"

Syntax:

```
ADD.<x>     <ea>,Dn
ADD.<x>     Dn,<ea>
ADDI.<x>    #<data>,<ea>
ADDQ.<x>    #<data>,<ea>
ADDA.<x>    <ea>,An
```

Attribut:
Storlek <x> kan vara W eller L då någon operand är ett adressregister, <x> kan vara B, W eller L i övriga fall.
Beskrivning:
Källoperand och destinationsoperand adderas, resultatet placeras i destinationsoperanden.
Anmärkning:
ADDQ.<x> är en kortare form av ADDI.<x> som kan användas om den adderade konstanten är i intervallet 1-8. An *kan* då användas som destination. Om destinationsoperanden är ett adressregister skall annars formen ADDA.<x> användas. Om källoperanden är en konstant (<data>) skall formen ADDI.<x> (alternativt ADDQ.<x>) användas.

<ea> som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-
<ea> som källa			
Dn	X	(xxx).W	X
An	X	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	1 om carry genererats, 0 annars
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill genererats, 0 annars
C	1 om carry genererats, 0 annars

Anmärkning:
Instruktionen ADDA.<x> påverkar ej flaggorna.

ADDX "add extended"

Syntax:

```
ADDX.<x>    Dy,Dx
ADDX.<x>    -(Ay),-(Ax)
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Adderar källoperanden och X-bit till destinationsoperanden och placerar resultatet i destinationen.

Flaggor	
X	1 om carry genererats, 0 annars
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill genererats, 0 annars
C	1 om carry genererats, 0 annars

AND "AND logical"
ANDI "AND immediate"

Syntax:

```
AND.<x>     <ea>,Dn
AND.<x>     Dn,<ea>
ANDI.<x>    #<data>,Dn
```

Attribut:
Storlek <x> kan vara B (*byte*), W(*word*) eller L (*long*)
Beskrivning:
Utför logiskt AND mellan källoperand och destinations-operand. Resultatet placeras i destinationsoperanden.
Anmärkning:
Om källoperanden är #<data> skall instruktionsformen ANDI.<x> användas.

<ea>som källa			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X
<ea>som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

ANDI to CCR "and immediate to condition code register"
ANDI to SR "and immediate to status register"

Syntax:

```
ANDI        #<data>,SR
ANDI        #<data>,CCR
```

Beskrivning:
Utför logiskt AND mellan <data> och processorregister, resultatet placeras i processorns register. Om destinations-operanden är CCR kommer de 8 minst signifikanta bitarna i <data> att maskas med det tidigare innehållet i CCR och på nytt placeras i CCR. Om destinations-operanden är SR, dvs processorns statusregister kommer <data> att maskas med det tidigare innehållet i SR och på nytt placeras i SR. **ANDI to CCR** kan alltid utföras medan **ANDI to SR** endast utförs om processorn är i *supervisor mode*.
Flaggor: Bestäms av operand och tidigare innehåll

ASL "arithmetic shift left"
ASR "arithmetic shift right"

Syntax:

```
ASd.<x>     Dx,Dy
ASd.<x>     #<data>,Dy
ASd         <ea>
```

Attribut:
Storlek <x> kan vara B (byte), W (*word*) eller L (*long*)
Beskrivning:
Destinationsoperanden skiftas aritmetiskt i den riktning som anges av instruktionen. *<antal steg>* kan anges på två olika sätt, om källoperanden är ett dataregister anger innehållet i detta register antalet skift. I den andra formen anges antalet skift som en konstant (1-8). I en tredje instruktionsform kan innehållet på en minnesadress skiftas. Detta skift är alltid ett steg och storleken på operanden är alltid *word*.

ASL: Operanden skiftas *vänster*. Den mest signifikanta skiftade biten kopieras till C-biten respektive X-biten i CC-registret. På den minst signifikanta bitens plats skiftas 0 in.

ASR: Operanden skiftas *höger*. Den mest signifikanta skiftade biten kopieras, dvs operandens tecken bibehålls. Den minst signifikanta skiftade biten kopieras till C-biten respektive X-biten i CC-registret.

<ea>			
Dn	**-**	**(xxx).W**	**X**
An	**-**	**(xxx).L**	**X**
(An)	**X**	**#<data>**	**-**
(An)+	**X**		
-(An)	**X**		
(d16,An)	**X**	**(d16,PC)**	**-**
(d8,An,Xn)	**X**	**(d8,PC,Xn)**	**-**
(bd,An,Xn)	**X**	**(bd,PC,Xn)**	**-**

Flaggor	
X	se ovan
N	1 om resultatet är 0, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Ettställs om den mest signifikanta biten ändras någon gång under operationen. Nollställs annars
C	se ovan

Bcc "branch conditionally"

Syntax:

```
Bcc.S  <label>      8-bitars offset
Bcc.B  <label>      8-bitars offset
Bcc    <label>      se anmärkning
Bcc.W  <label>      16-bitars offset
Bcc.L  <label>      32-bitars offset
```

Anmärkning:
Med denna form kommer assemblatorn att försöka välja den kortast möjliga offseten.
Beskrivning:
Om det angivna villkoret är uppfyllt utförs hoppet, annars fortsätter exekveringen vid den påföljande instruktionen. Testen ger resultatet av tidigare instruktion som påverkat flaggorna i CCR.

Följande villkor (cc) kan anges:

BHI	High	$\overline{C} \wedge \overline{Z}$
BLS	Low or Same	$C \vee Z$
BCC	Carry Clear	$\overline{C}$
BCS	Carry Set	C
BNE	Not Equal	$\overline{Z}$
BEQ	Equal	Z
BVC	Overflow Clear	$\overline{V}$
BVS	Overflov Set	V
BPL	Plus	$\overline{N}$
BMI	Minus	N
BGE	Greater or Equal	$N \wedge Z \vee \overline{V} \wedge \overline{Z}$
BLT	Less Than	$N \wedge \overline{Z} \vee \overline{V} \wedge Z$
BGT	Greater Than	$N \wedge Z \wedge \overline{Z} \vee \overline{V} \wedge \overline{Z} \wedge \overline{Z}$
BLE	Less or Equal	$Z \vee V \wedge \overline{Z} \vee \overline{V} \wedge Z$

Flaggor: Påverkas ej

BCHG "test a bit and change"

Syntax:

```
BCHG.L      Dn,Dm
BCHG.L      #<data>,Dn
BCHG.B      Dn,<ea>
BCHG.B      #<data>,<ea>
```

Beskrivning:
En bit hos destinationsoperanden testas, resultatet av testen återspeglas av Z-biten i CC-registret. Destinationsoperandens testade bit inverteras därefter. Om destinationsoperanden är ett dataregister anger källoperanden den aktuella biten (modulo 32), dvs vilken som av de 32 bitarna i dataregistret kan manipuleras. Om destinationsoperanden är en minnesadress kommer 8-bitar att läsas från denna adress, därefter utförs operationen (modulo 8) och slutligen skrivs resultatet tillbaks i minnet. Den testade biten kan anges på två olika sätt, i instruktionens första form

```
BCHG.B      Dn,<ea>
```

anges bit-numret av innehållet i det specificerade dataregistret. I instruktionens andra form

```
BCHG.B      #<data>,<ea>
```

anges bit-numret av en konstant.

Anmärkning:
Observera att bitnumret anges som bit-namn, dvs den minst signifikanta biten betecknas 0 och den mest signifikanta biten betecknas 31.

<ea> som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	Påverkas ej
Z	1 om den testade biten är 0, 0 annars
V	Påverkas ej
C	Påverkas ej

BCLR "test a bit and clear"

Syntax:

```
BCLR.L      Dn,Dm
BCLR.L      #<data>,Dn
BCLR.B      Dn,<ea>
BCLR.B      #<data>,<ea>
```

Beskrivning:
En bit hos destinationsoperanden testas, resultatet av testen återspeglas av Z-biten i CC-registret. Destinationsoperandens testade bit nollställs därefter. Om destinations-operanden är ett dataregister anger källoperanden den aktuella biten (modulo 32), dvs vilken som av de 32 bitarna i dataregistret kan manipuleras. Om destinations-operanden är en minnesadress kommer 8-bitar att läsas från denna adress, därefter utförs operationen (modulo 8) och slutligen skrivs resultatet tillbaks i minnet. Den testade biten kan anges på två olika sätt, i instruktionens första form

```
BCLR.B Dn,<ea>
```

anges bit-numret av innehållet i det specificerade dataregistret. I instruktionens andra form

```
BCLR.B #<data>,<ea>
```

anges bit-numret av en konstant.
Anmärkning:
Observera att bitnumret anges som bit-namn, dvs den minst signifikanta biten betecknas 0 och den mest signifikanta biten betecknas 31.

<ea> som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	Påverkas ej
Z	1 om den testade biten är 0, 0 annars
V	Påverkas ej
C	Påverkas ej

BRA "branch always"

Syntax:

```
BRA.S   <label>     8-bitars offset
BRA.B   <label>     8-bitars offset
BRA     <label>     se anmärkning
BRA.W   <label>     16-bitars offset
BRA.L   <label>     32-bitars offset
```

Anmärkning:
Med denna form kommer assemblatorn att försöka välja den kortast möjliga offseten.
Beskrivning:
Exekveringen fortsätter på den adress som ges av PC+offset. "offset" är ett tal på tvåkomplementform. Vid adresssberäkningen innehåller PC adressen till BRA-instruktionen + 2.
Flaggor: Påverkas ej

BSET "test a bit and set"

Syntax:

```
BSET.L      Dn,Dm
BSET.L      #<data>,Dn
BSET.B      Dn,<ea>
BSET.B      #<data>,<ea>
```

Beskrivning:
En bit hos destinationsoperanden testas, resultatet av testen återspeglas av Z-biten i CC-registret. Destinationsoperandens testade bit ettställs därefter. Om destinations-operanden är ett dataregister anger källoperanden den aktuella biten (modulo 32), dvs vilken som av de 32 bitarna i dataregistret kan manipuleras. Om destinations-operanden är en minnesadress kommer 8-bitar att läsas från denna adress, därefter utförs operationen (modulo 8) och slutligen skrivs resultatet tillbaks i minnet. Den testade biten kan anges på två olika sätt, i instruktionens form

```
BSET.B      Dn,<ea>
```

anges bit-numret av innehållet i det specificerade dataregistret. I instruktionens andra form

```
BSET.B      #<data>,<ea>
```

anges bit-numret av en konstant.
Anmärkning:
Observera att bitnumret anges som bit-namn, dvs den minst signifikanta biten betecknas 0 och den mest signifikanta biten betecknas 31.

<ea> som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	Påverkas ej
Z	1 om den testade biten är 0, 0 annars
V	Påverkas ej
C	Påverkas ej

BSR "branch to subroutine"

Syntax:

```
BSR.S  <label>      8-bitars offset
BSR.B  <label>      8-bitars offset
BSR    <label>      se anmärkning
BSR.W  <label>      16-bitars offset
BSR.L  <label>      32-bitars offset
```

Anmärkning:
Med denna form kommer assemblatorn att försöka välja den kortast möjliga offseten.
Beskrivning:
Adressen till instruktionen omedelbart efter BSR placeras på stacken. Exekveringen fortsätter på den adress som ges av PC+offset. "offset" är ett tal på tvåkomplement-form. Vid adresssberäkningen innehåller PC adressen till BSR-instruktionen + 2.
Flaggor: Påverkas ej

BTST "test a bit"

Syntax:

```
BTST.L      Dn,Dm
BTST.L      #<data>,Dn
BTST.B      Dn,<ea>
BTST.B      #<data>,<ea>
```

Beskrivning:
En bit hos destinationsoperanden testas, resultatet av testen återspeglas av Z-biten i CC-registret. Destinationsoperandens testade bit påverkas inte. Om destinations-operanden är ett dataregister anger källoperanden den aktuella biten (modulo 32), dvs vilken som av de 32 bitarna i dataregistret kan testas. Om destinations-operanden är en minnesadress kommer 8-bitar att läsas från denna adress. Den testade biten kan anges på två olika sätt, i instruktionens form

```
BTST.B      Dn,<ea>
```

anges bit-numret av innehållet i det specificerade dataregistret. I instruktionens form

```
BTST.B      #<data>,<ea>
```

anges bit-numret av en konstant.
Anmärkning:
Observera att bitnumret anges som bit-namn, dvs den minst signifikanta biten betecknas 0 och den mest signifikanta biten betecknas 31.

<ea> som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	Påverkas ej
Z	1 om den testade biten är 0, 0 annars
V	Påverkas ej
C	Påverkas ej

CHK "check registers against bounds"

Syntax:

```
CHK    <ea>,Dn
```

Attribut: <x> kan vara B (*byte*), W (*word*) eller L (*long*).
Beskrivning:
De 16 minst signifikanta bitarna i Dn jämförs med "övre gräns" som utgörs av operanden på effektiva adressen. Detta tal betraktas som tal på tvåkomplements-form. Om innehållet i Dn är mindre än noll, eller större än operanden på effektiva adressen startas *exception processing*.

<ea> (chk)			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	1 om Dn < 0, 0 om Dn > ea-operand, annars odefinierad
Z	Odefinierad
V	Odefinierad
C	Odefinierad

CHK2 "check registers against bounds 2"

Syntax:

```
CHK2.<x>      <ea>,Rn
```

Attribut: <x> kan vara B (*byte*), W (*word*) eller L (*long*).
Beskrivning:
Jämför värdet i Rn med undre gräns (LB) och övre gräns (UB). Effektiva adressen innehåller LB:UB. Rn kan vara ett data- eller adress-register.

<ea> (chk2)			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	-		
-(An)	-		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	Odefinierad
Z	1 om Rn är lika med någon av gränserna, 0 annars O
V	Odefinierad
C	1 om Rn utanför gränserna, 0 annars

CLR "clear an operand"

Syntax:

```
CLR.<x>      <ea>
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Samtliga bitar (8,16 eller 32 beroende på <x>) i operanden nollställs.

<ea>			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	Nollställs alltid
Z	Ettställs alltid
V	Nollställs alltid
C	Nollställs alltid

CMP "compare"
CMPA "compare address"

Syntax:

```
CMP.<x>      <ea>,Dn
CMPA.<x>     <ea>,An
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*) utom då någon operand är ett adressregister då endast W och L är tillåtna.
Beskrivning:
Subtraherar källoperanden från destinationsoperanden. Operationen påverkar endast innehållet i CC-registret.
Anmärkning:
Instruktionsformen CMPI (se nedan) *ska* användas om källoperanden är en konstant.

<ea> som källa			
Dn	X	(xxx).W	X
An	X	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill genererats, 0 annars
C	1 om lånesiffra genererats, 0 annars

CMPI "compare immediate"

Syntax:

```
CMPI.<x>     #<data>,<ea>
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*).
Beskrivning:
Subtraherar källoperanden från destinationsoperanden. Operationen påverkar endast innehållet i CC-registret.

<ea> som destination			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill genererats, 0 annars
C	1 om lånesiffra genererats, 0 annars

CMPM "compare memory"

Syntax:

```
CMPM.<x>     (Ay)+,(Ax)+
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Jämför minnesinnehåll och påverkar flaggorna, ökar därefter innehållet i de adressregister som angetts, med 1 om storleken är byte, 2 om storleken är *word* och med 4 om storleken är *long*.

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill genererats, 0 annars
C	1 om lånesiffra genererats, 0 annars

CMP2 "compare register against bounds

Syntax:

```
CMP2.<x>     <ea>,Rn
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Jämför innehållet i Rn med en övre gräns (UB) respektive undre gräns (LB) och påverkar flaggorna. Effektiva adressen utgör adress till både UB och LB, LB först. Rn kan vara ett adress- eller data-register. Instruktionen är identisk med chk2 bortsett från att chk2-instruktionen kan generera *trap*.

<ea> som källa			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	-		
-(An)	-		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	Odefinierad
Z	1 om Rn är lika någon gräns, 0 annars
V	Odefinierad
C	1 om Rn utanför gränserna, 0 annars

DBcc "test condition, decrement and branch"

Syntax:

```
DBcc    Dn,<label>      16-bitars offset
```

Beskrivning:
Instruktionen används för att skapa programslingor. Den har tre parametrar: ett villkor (jämför "branch conditionally"), ett dataregister och en offset. Instruktionen testar först villkoret för att se om detta är uppfyllt. Om detta är sant utförs inget mer. Om termineringsvillkoret inte är sant minskas innehållet i det angivna dataregistret med 1. Om resultatet i dataregistret nu är -1 fortsätter exekveringen med nästa instruktion, om inte fortsätter exekveringen vid den adress som anges av PC+"offset". Första testen ger resultatet av tidigare instruktion som påverkat flaggorna i CCR.

Operandstorleken är alltid *word*.

Följande villkor (cc) kan anges:

DBHI	High	$\overline{C} \wedge \overline{Z}$
DBLS	Low or Same	$C \vee Z$
DBCC	Carry Clear	$\overline{C}$
DBCS	Carry Set	C
DBNE	Not Equal	$\overline{Z}$
DBEQ	Equal	Z
DBVC	Overflow Clear	$\overline{V}$
DBVS	Overflov Set	V
DBPL	Plus	$\overline{N}$
DBMI	Minus	N
DBGE	Greater or Equal	$N \wedge V \vee \overline{N} \wedge \overline{V}$
DBLT	Less Than	$N \wedge \overline{V} \vee \overline{N} \wedge V$
DBGT	Greater Than	$N \wedge V \wedge \overline{Z} \vee \overline{N} \wedge \overline{V} \wedge \overline{Z}$
DBLE	Less or Equal	$Z \vee N \wedge \overline{V} \vee \overline{N} \wedge V$
DBRA	False	0
DBT	True	1
DBF	False	0

Anmärkning:
I formen DBF (eller DBRA), dvs. villkoret är aldrig uppfyllt, kan instruktionen användas för att åstadkomma ett givet antal iterationer.

EXEMPEL:

```
        MOVE.L #loopcount-1,D0
loop:
        ...
        ...
        DBRA            D0,loop
```

Instruktioner mellan loop och DBRA utförs *loopcount* gånger

Flaggor: Påverkas ej

DIVS "signed divide"
DIVSL "signed divide"
DIVU "unsigned divide"
DIVUL "unsigned divide"

Syntax:

```
DIVS          <ea>,Dn
DIVS.W        <ea>,Dn  32/16➔16r-16q
DIVU          <ea>,Dn
DIVU.W        <ea>,Dn  32/16➔16r-16q
DIVS.L        <ea>,Dq  32/32➔32q
DIVU.L        <ea>,Dq  32/32➔32q
DIVS.L        <ea>,Dr:Dq
                       64/32➔32r-32q
DIVU.L        <ea>,Dr:Dq
                       64/32➔32r-32q
DIVSL.L       <ea>,Dr:Dq
                       32/32➔32r-32q
DIVUL.L       <ea>,Dr:Dq
                       32/32➔32r-32q
```

Beskrivning:
Dividerar två tal och placerar resultatet i ett eller eventuellt två dataregister. Resultatet är en *rest* (remainder) och en *kvot* (quotient). Teckenbiten för "resten" är densamma som hos dividenden (DIVS) om "resten" är skild från 0.

Två speciella tillstånd kan uppträda:

- Division med 0, vilket föranleder en *trap*
- Spill kan uppträda under instruktionen. Detta detekteras och avspeglas i CC-registret men operanderna påverkas ej.

<ea> som källa			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	1 om kvoten negativ, 0 annars, odefinierad om spill
Z	1 om kvoten är 0, 0 annars, odefinierad om spill
V	1 om division spill genererats, 0 annars
C	Nollställs alltid

Anm: DIVS är kortform för DIVS.W, DIVU är kortform för DIVU.W, godtyckligt skrivsätt kan användas.

EOR "exclusive OR logical"
EORI "exclusive OR immediate"

Syntax:

```
EOR.<x>      Dn,<ea>
EORI.<x>     #<data>,Dn
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Utför exklusivt OR mellan källoperand och destinationsoperand. Resultatet placeras i destinationsoperanden.
Anmärkning:
Om källoperanden är #<data> skall instruktionsformen EORI.<x> användas.

<ea>som destination			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

EORI to SR "exclusive or immediate to status register"
EORI to CCR "exclusive or immediate to condition code register"

Syntax:

```
EORI         #<data>,SR
EORI         #<data>,CCR
```

Beskrivning:
Utför logiskt EXCLUSIVE OR mellan <data> och processorregister, resultatet placeras i processorns register. Om destinationsoperanden är CCR kommer de 8 minst signifikanta bitarna i <data> att maskas med det tidigare innehållet i CCR och på nytt placeras i CCR. Om destinationsoperanden är SR, dvs processorns statusregister kommer <data> att maskas med det tidigare innehållet i SR och på nytt placeras i SR. **EORI to CCR** kan alltid utföras medan **EORI to SR** endast utförs om processorn är i *supervisor mode*.
Flaggor: Bestäms av operand och tidigare innehåll

EXG "exchange registers"

Syntax:

```
EXG          Dx,Dy
EXG          Ax,Ay
EXG          Ax,Dy
```

Beskrivning:
Skiftar innehållen (32 bitar) mellan två register. Såväl data- som adressregister kan anges.

Flaggor: Påverkas ej

EXT "sign extend"

Syntax:

```
EXT.W        Dn
EXT.L        Dn
EXTB.L       Dn
```

Beskrivning:
Teckenutvidgar innehållet i ett dataregister från *byte* till *word* (EXT.W) eller från *word* till *long* (EXT.L) eller från *byte* till *long* (EXTB.L).

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

ILLEGAL "take illegal instruction trap"

Utför "illegal instruction trap (vektor 4).

Flaggor: Påverkas ej

JMP "jump"

Syntax:

```
JMP     <ea>
```

Beskrivning:
Exekveringen fortsätter på den adress som anges av effektiva adressen.

<ea>			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	-		
-(An)	-		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor: Påverkas ej

JSR "jump to subroutine"

Syntax:

```
JSR          <ea>
```

Beskrivning:
Adressen till nästa instruktion placeras på stacken. Exekveringen fortsätter på den adress som angetts av effektiva adressen.

<ea>			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	-		
-(An)	-		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor: Påverkas ej

LEA "load effective address"

Syntax:

```
LEA          <ea>,An
```

Beskrivning:
Effektiva adressen (*long*) laddas i angivet adressregister.

<ea> som källa			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	-		
-(An)	-		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor: Påverkas ej

LINK "link and allocate"

Syntax:

```
LINK         An,#<data>
LINK.W       An,#<data>
LINK.L       An,#<data>
```

Beskrivning:
Innehållet i det av källoperanden angivna adressregistret placeras på stacken, stackpekarens nya innehåll kopieras till adressregistret, slutligen adderas konstanten <data> till stackpekaren.
Anmärkning:
LINK och UNLK instruktionerna kan användas för att skapa en länkad lista av lokala data och parametrar.
Flaggor: Påverkas ej

LPSTOP "low-power stop"

Syntax:

```
LPSTOP       #<data>
```

Beskrivning:
Operanden överförs till statusregistret, programräknaren avanceras och pekar på nästa instruktion, därefter upphör instruktions-hämtningen och processorn försätts i *strömsnål mode*. Instruktionsexekvering återupptas efter en *trace*, *interrupt* eller *reset* exception. Trace-exception kan endast inträffa om T-biten i SR är 1 då instruktionen utförs.

LSL "logical shift left"
LSR "logical shift right"

Syntax:

```
LSd.<x>      Dx,Dy
LSd.<x>      #<data>,Dy
LSd          <ea>
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Destinationsoperanden skiftas i den riktning som anges av instruktionen. *<antal steg>* kan anges på två olika sätt, om källoperanden är ett dataregister anger innehållet i detta register antalet skift. I den andra formen anges antalet skift som en konstant (1-8). I en tredje instruktionsform kan innehållet på en minnesadress skiftas. Detta skift är alltid ett steg och storleken på operanden är alltid *word*.
LSL: Operanden skiftas vänster. Den mest signifikanta skiftade biten kopieras till C-biten respektive X-biten i CC-registret. På den minst signifikanta bitens plats skiftas 0 in. Operationen är ekvivalent med **ASL**.

LSR: Operanden skiftas höger. Den mest signifikanta biten ersätts av 0. Den minst signifikanta skiftade biten kopieras till C-biten respektive X-biten i CC-registret.

<ea>			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	se ovan
N	1 om resultatet är 0, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	se ovan

MOVE "move data"
MOVEA "move address"
MOVEQ "move quick"

Syntax:

```
MOVE.<x>     <ea>,<ea>
MOVEQ        #<data>,Dn
MOVEA.<x>    <ea>,An
```

Attribut:
Storlek <x> kan vara: B (*byte*), W (*word*) eller L (*long*) utom för adresseringsmod An, där storlek kan vara W eller L. MOVEQ är alltid *long*.
Beskrivning:
Kopierar källoperand till destination. MOVEQ *kan* användas om källoperanden är en konstant med högst 8 bitar och destinationsoperanden är ett dataregister. Konstanten teckenutvidgas före operationen. MOVEA.<x> *ska* användas om destinations-operanden är ett adressregister.

<ea> som källa			
Dn	X	(xxx).W	X
An	X	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X
<ea> som destination			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

Anm: MOVEA.<x> påverkar ej flaggorna.

MOVE from CCR "move from condition code register"

Syntax:

```
MOVE        CCR,<ea>
```

Beskrivning:
Innehållet i CC registret kopieras till destinationen. Detta är en teckenutvidgande 8 till 16-bitars operation eftersom endast innehållet i CCR (8 bitar) läses.

<ea> som destination			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor: Påverkas ej

MOVE to CCR "move to condition code register"

Syntax:

```
MOVE        <ea>,CCR
```

Beskrivning:
Källoperanden kopieras till processorns flaggregister. Operandens storlek är 16 bitar men endast de 8 minst signifikanta bitarna används.

MOVE from SR "move from status register"

Syntax:

```
MOVE        SR,<ea>
```

Beskrivning:
Innehållet i statusregistret (16 bitar) kopieras till destinationen.

Anmärkning:
Detta en *priviligierad* instruktion.

<ea> som destination			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor: Påverkas ej

MOVE to SR "move to status register"

Syntax:

```
MOVE        <ea>,SR
```

Beskrivning:
Källoperanden kopieras till processorns statusregister. Operandens storlek är 16 bitar och hela statusregistret påverkas.
Anmärkning:
Instruktionen utförs endast om processorn är i *supervisor mode*, om inte sker en *privilege violation* trap

<ea> som källa			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor: Bestäms av operand.

MOVE USP "move user stack{ XE "stack" } pointer"

Syntax:

```
MOVE        USP,An
MOVE        An,USP
```

Beskrivning:
Innehållet i *user stack pointer* kopieras till ett adressregister respektive: innehållet i ett adressregister kopieras till *user stack pointer*. Instruktionen är priviligierad och användbar endast då processorn är i *supervisor mode* och skall ändra den stackpekare som är aktiv i *user mode* alternativt spara den stackpekare som är aktiv i *user mode*.
Flaggor: Påverkas ej

MOVEC "move control register"

Syntax:

```
MOVEC       Rn,Rc
MOVEC       Rc,Rn
```

Attribut: Inga, storlek är alltid 32 bitar (*long*)
Beskrivning:
Instruktionens första form kopierar innehållet i ett generellt register (32 bitar) Rn (Dn eller An) till ett kontrollregister Rc. Instruktionens andra form kopierar innehållet i ett kontrollregister (32 bitar) till ett generellt register.

Kontrollregister:

SFC	source function code register
DFC	destination function code register
USP	user stack pointer
VBR	vector base register

Flaggor: Påverkas ej

MOVEM "move multiple registers"

Syntax:

```
MOVEM.<x>     <register>,<ea>
MOVEM.<x>     <ea>,<register>
```

Attribut:
Storlek <x> kan vara: W (*word*) eller L (*long*)
Beskrivning:
Godtyckligt antal registerinnehåll kan kopieras till konsekutiva adresser i minnet. Hela registerinnehållet (32 bitar) kan kopieras (MOVEM.L) eller endast de 16 minst signifikanta bitarna (MOVEM.W). Listan av register kan anges på flera sätt. Enskilda register separeras med '/', konsekutiva register separeras med '-'.

EXEMPEL: *lista av register*

```
D0/D1/A0        register d0,d1 och a0
D0-D2/A0-A1     register d0,d1,d2,a0 och a1
```

EXEMPEL: *spara registerinnehåll*
Instruktionen kan användas för att spara registerinnehåll på stacken med:

```
MOVEM.L         D0-D7/A0-A6,-(A7)
```

Den andra formen av MOVEM.<x> kopierar konsekutivt minnesinnehåll till de angivna registren.

EXEMPEL: *återställ registerinnehåll*
Hela registeruppsättningen kan återställas från stacken med:

```
MOVEM.L         (A7)+,D0-D7/A0-A6
```

<ea> som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	-		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-
<ea> som källa			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	-		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor: Påverkas ej

MOVEP "move peripheral data"

Syntax:

```
MOVEP.<x>     Dx,(d16,An)
MOVEP.<x>     (d16,An),Dx
```

Attribut:
Storlek <x> kan vara W (*word*) eller L (*long*)
Beskrivning:
Kopierar data mellan dataregister och udda eller jämna adresser i minnet. Instruktionen kan användas då 8-bitars periferikretsar adresseras på en 16-bitars databuss. Om EA är en jämn adress kommer påföljande jämna adresser att användas. Om EA är en udda adress kommer påföljande udda adresser att användas. Om storleken anges till *long* kommer 4 bytes att kopieras med början på den mest signifikanta byten i dataregistret. Om storleken är *word* kommer två bytes från dataregistret att kopieras med början på den byte som bildas av bitarna 8-15, därefter den byte som bildas av bitarna 0-7.

Flaggor: Påverkas ej

MOVES "move address space"

Syntax:

```
MOVES.<x>     Rn,<ea>
MOVES.<x>     <ea>,Rn
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Instruktionens första form kopierar innehållet i Rn (Dn eller An) till det adressrum som anges av DFC.
Instruktionens andra form kopierar data från det adressrum som anges av SFC till ett register Dn eller An.
Anmärkning:
Instruktionen utförs endast om processorn är i *supervisor mode*, om inte sker en *privilege violation* trap.

<ea> som källa/destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor

MULS "signed multiply"
MULU "unsigned multiply"

Operation: Destination * Källa ⇒ Destination
Syntax:

```
MULS    <ea>,Dn
MULS.W  <ea>,Dn         16•16➔32
MULU    <ea>,Dn
MULU.W  <ea>,Dn         16•16➔32
MULS.L  <ea>,DI         32•32➔32
MULU.L  <ea>,DI         32•32➔32
MULS.L  <ea>,Dh-DI      32•32➔64
MULU.L  <ea>,Dh-DI      32•32➔64
```

Beskrivning:
Multiplicerar två tal och placerar resultatet i det eller de dataregister som angetts som destination. MULS förutsätter tal på tvåkomplementform medan MULU utför multiplikation på tal utan tecken.

<ea> som källa			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill, 0 annars
C	Nollställs alltid

Anm: Spill kan endast uppkomma då två 32-bitars operander multipliceras och resultatet placeras i *ett* dataregister. MULS är kortform för MULS.W, MULU är kortform för MULU.W, godtyckligt skrivsätt kan användas.

NBCD "negate decimal with extend"

Syntax:

```
NBCD        <ea>
```

Beskrivning:
Operanden och X-biten subtraheras från 0. Resultatet placeras på operandens adress. Operationen utförs med BCD-aritmetik. Instruktionen skapar alltså 10-komplementet av operanden om X är 0, och 9-komplementet av operanden om X är 1. Operandens storlek är alltid 8 bitar (byte).

<ea>			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	1 om lånesiffra (decimal) genererats, 0 annars
N	Odefinierad
Z	0 om resultatet är skilt från 0, påverkas annars inte
V	Odefinierad
C	1 om lånesiffra (decimal) genererats, 0 annars

NEG "negate"

Syntax:

```
NEG.<x>     <ea>
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Operanden subtraheras från 0, resultatet placeras i destinationen, med andra ord: tvåkomplementering av operanden.

<ea>			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	1 om carry genererats, 0 annars
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill genererats, 0 annars
C	0 om resultatet är 0, 1 annars

NEGX "negate with extend"

Syntax:

```
NEGX.<x>    <ea>
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Operanden och X-bit subtraheras från 0, resultatet placeras i destinationen.

<ea>			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	1 om lånesiffra genererats, 0 annars
N	1 om resultatet negativt, 0 annars
Z	0 om resultatet är skilt från 0, påverkas ej annars
V	1 om spill genererats, 0 annars
C	1 om lånesiffra genererats, 0 annars

NOP "no operation"

Syntax:

```
NOP
```

Beskrivning:
Programräknaren ökas och pekar därefter på nästa instruktion.
Flaggor: Påverkas ej.

NOT "logical complement"

Syntax:

```
NOT.<x>     <ea>
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Utför ett-komplementering av operanden.

<ea>som destination			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

OR "inclusive OR"
ORI "inclusive OR immediate"

Syntax:

```
OR.<x>      <ea>,Dn
OR.<x>      Dn,<ea>
ORI.<x>     #<data>,Dn
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Utför logiskt OR mellan källoperand och destinations-operand. Resultatet placeras i destinationsoperanden.
Anmärkning:
Om källoperanden är #<data> skall instruktionsformen ORI.<x> användas.

<ea>som källa			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X
<ea>som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

ORI to SR "or immediate to status register"
ORI to CCR "or immediate to Condition Code Register"

Syntax:

```
ORI         #<data>,SR
ORI         #<data>,CCR
```

Beskrivning:
Utför logiskt OR mellan <data> och register, resultatet placeras i processorns register. Om destinationsoperanden är CCR kommer de 8 minst signifikanta bitarna i <data> att maskas med det tidigare innehållet i CCR och på nytt placeras i CCR. Om destinationsoperanden är SR, dvs processorns statusregister kommer <data> att maskas med det tidigare innehållet i SR och på nytt placeras i SR. **ORI to CCR** kan alltid utföras medan **ORI to SR** endast utförs om processorn är i *supervisor mode*.
Flaggor: Bestäms av operand och tidigare innehåll

PEA "push effective address"

Syntax:

```
PEA         <ea>
```

Beskrivning:
Effektiva adressen (*long*) bestäms och placeras på stacken.

<ea>			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	-		
-(An)	-		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor: Påverkas ej

RESET "reset peripherals"

Syntax:

```
RESET
```

Beskrivning:
Processorn aktiverar *reset*-signalen och tvingar därmed periferienheter till ett väldefinierat tillstånd. Processorns tillstånd påverkas ej utan exekveringen fortsätter med nästa instruktion.
Anmärkning:
Instruktionen utförs endast om processorn är i *supervisor mode*, om inte sker en *privilege violation* trap

Flaggor: Påverkas ej

ROL "rotate left without extend"
ROR "rotate right without extend"

Syntax:

```
ROd.<x>     Dx,Dy
ROd.<x>     #<data>,Dy
ROd         <ea>
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Destinationsoperanden roteras i den riktning som anges av instruktionen. <*antal steg*> kan anges på två olika sätt, om källoperanden är ett dataregister anger innehållet i detta register antalet skift. I den andra formen anges antalet skift som en konstant (1-8). I en tredje instruktionsform kan innehållet på en minnesadress skiftas. Detta skift är alltid ett steg och storleken på operanden är alltid *word*.

ROL: Operanden skiftas vänster. Den mest signifikanta skiftade biten kopieras till den minst signifikanta skiftade biten och till C-biten i CC-registret.

ROR: Operanden skiftas höger. Den minst signifikanta skiftade biten kopieras till C-biten i CC-registret respektive den mest signifikanta skiftade biten.

<ea>			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	se ovan
N	1 om resultatet är 0, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	se ovan

ROXL "rotate left with extend"
ROXR "rotate right with extend"

Syntax:

```
ROXd.<x>      Dx,Dy
ROXd.<x>      #<data>,Dy
ROXd          <ea>
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Destinationsoperanden roteras i den riktning som anges av instruktionen. *<antal steg>* kan anges på två olika sätt, om källoperanden är ett dataregister anger innehållet i detta register antalet skift. I den andra formen anges antalet skift som en konstant (1-8). I en tredje instruktionsform kan innehållet på en minnesadress skiftas. Detta skift är alltid ett steg och storleken på operanden är alltid *word*.

ROXL: Operanden skiftas vänster. X-biten i CC-registret kopieras till den minst signifikanta skiftade biten. Den mest signifikanta skiftade biten kopieras till C-biten respektive X-biten i CC-registret.

ROXR: Operanden skiftas höger. X-biten i CC-registret kopieras till den mest signifikanta skiftade biten. Den minst signifikanta skiftade biten kopieras till C-biten respektive X-biten i CC-registret.

<ea>			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	se ovan
N	1 om resultatet är 0, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	se ovan

RTD "return and deallocate"

Syntax:

```
RTD           #<disp>
```

Beskrivning:
Kopierar värdet överst på stacken till programräknaren, adderar därefter <disp>, teckenutvidgat från 16 bitar till 32, till stackpekaren. Föregående programräknarvärde förloras.
Flaggor: Påverkas ej

RTE "return from exception"

Syntax:

```
RTE
```

Beskrivning:
Det översta ordet (16 bitar) på stacken kopieras till statusregistret. Stackpekaren ökas med 2. Den 32-bitars adress som nu ligger överst på stacken kopieras till programräknaren. Därefter ökas stackpekaren med 4.
Anmärkning:
Instruktionen utförs endast om processorn är i *supervisor mode*, om inte sker en *privilege violation* trap.
Flaggor: Se ovan

RTR "return and restore condition codes"

Syntax:

```
RTR
```

Beskrivning:
Det 8-bitars ord som ligger överst på stacken kopieras till CC-registret. Stack-pekaren ökas med 2. Därefter placeras den 32-bitars adress som nu ligger överst på stacken i PC och stackpekaren ökas med 4.
Flaggor: Se ovan

RTS "return from subroutine"

Syntax:

```
RTS
```

Den 32-bitars adress som ligger överst på stacken kopieras till PC, stackpekaren ökas med 4.
Flaggor: Påverkas ej

SBCD "subtract decimal with extend"

Syntax:

```
SBCD          Dy,Dx
SBCD          -(Ay),-(Ax)
```

Beskrivning:
Källoperanden subtraheras från destinationsoperanden tillsammans med X-biten i CCR. Operationen utförs med användning av BCD-aritmetik. Operanderna kan anges på två olika sätt:

1. data register till data register: dvs operanderna finns i de dataregister som anges av instruktionen.
2. minnesadress till minnesadress: operandernas adress anges av det minskade innehållet i något adressregister.

Operandens storlek är alltid 8 bitar (byte).

Flaggor	
X	1 om lånesiffra (decimal) genererats, 0 annars
N	Odefinierad
Z	0 om resultatet är skilt från 0, påverkas annars inte
V	Odefinierad
C	1 om lånesiffra (decimal) genererats, 0 annars

Scc "set according to condition"

Syntax:

```
Scc             <ea>
```

Beskrivning:
Om det angivna villkoret är uppfyllt sätts operanden till 1, annars sätts operanden till 0. Villkoret är resultat av tidigare instruktion som påverkat flaggorna i CCR. Operandens storlek är alltid *byte*
Följande villkor (cc) kan anges:

SHI	High	$\overline{C} \wedge \overline{Z}$
SLS	Low or Same	$C \vee Z$
SCC	Carry Clear	$\overline{C}$
SCS	Carry Set	C
SNE	Not Equal	$\overline{Z}$
SEQ	Equal	Z
SVC	Overflow Clear	$\overline{V}$
SVS	Overflov Set	V
SPL	Plus	$\overline{N}$
SMI	Minus	N
SGE	Greater or Equal	$N \wedge V \vee \overline{V} \wedge \overline{V}$
SLT	Less Than	$N \wedge \overline{V} \vee \overline{V} \wedge V$
SGT	Greater Than	$N \wedge V \wedge \overline{Z} \vee \overline{V} \wedge \overline{V} \wedge \overline{Z}$
SLE	Less or Equal	$Z \vee V \wedge \overline{V} \vee \overline{V} \wedge V$
ST	True	1
SF	False	0

<ea>			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor: Påverkas ej

STOP "load status register and stop"

Syntax:

```
STOP            #<data>
```

Beskrivning:
Operanden (word) kopieras till statusregistret. Programräknaren pekar på nästa instruktion i koden. Processorn avbryter exekveringen. Exekvering av nästa instruktion återupptas då ett RESET tillstånd, eller ett avbrott med högre prioritet än processorns uppträder
Anmärkning:
Instruktionen utförs endast om processorn är i *supervisor mode*, om inte sker en *privilege violation* trap
Flaggor: Se ovan

SUB "subtract binary"
SUBA "subtract address"
SUBI "subtract immediate"
SUBQ "subtract quick"

Syntax:

```
SUB.<x>         <ea>,Dn
SUB.<x>         Dn,<ea>
SUBI.<x>        #<data>,<ea>
SUBQ.<x>        #<data>,<ea>
SUBA.<x>        <ea>,An
```

Attribut:
Storlek <x> kan vara w eller l då någon av operanderna är ett adressregister. <x> kan vara B, W eller L i övriga fall.
Beskrivning:
Källoperand subtraheras från destinationsoperand, resultatet placeras i destinationsoperanden.
Anmärkningar:
SUBQ.<x> är en kortare form av SUBI.<x> som kan användas om den subtraherade konstanten är i intervallet 1-8. I detta fall kan även ett adressregister anges som destinationsoperand. I övrigt gäller att: om destinationsoperanden är ett adressregister *ska* formen SUBA.<x> användas. Om källoperanden är en konstant *ska* formen SUBI.<x> användas,

<ea> som källa			
Dn	X	(xxx).W	X
An	X	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X
<ea> som destination			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	1 om lånesiffra genererats, 0 annars
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill genererats, 0 annars
C	1 om lånesiffra genererats, 0 annars

Anmärkning:
Instruktionen SUBA.<x> påverkar ej flaggorna.

SUBX "subtract with extend"

Syntax:

```
SUBX.<x>      Dy,Dx
SUBX.<x>      -(Ay),-(Ax)
```

Attribut:
Storlek <x> kan vara B (*byte*), W (*word*) eller L (*long*)
Beskrivning:
Subtraherar källoperanden och X-bit från destinations-operanden och placerar resultatet i destinationen.

Flaggor	
X	1 om lånesiffra genererats, 0 annars
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om spill genererats, 0 annars
C	1 om lånesiffra genererats, 0 annars

SWAP "swap register halves"

Syntax:

```
SWAP          Dn
```

Beskrivning:
Instruktionen skiftar innehållen (16 bitar) mellan de mest, respektive minst, signifikanta orden i ett dataregister.

Flaggor	
X	Påverkas ej
N	1 om mest signifikanta biten av det 32-bitars resultatet är 1, 0 annars
Z	1 om det 32-bitars resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

TAS "test and set an operand"

Syntax:

```
TAS           <ea>
```

Beskrivning:
Testar operanden som anges av effektiva adressen. N- och Z-flaggorna sätts beroende på operanden. Den mest signifikanta biten i operanden sätts till 1. Operationen är odelbar, dvs hela instruktionen utförs alltid för att möjliggöra synkronisering i multiprocessor applikationer. Operandes storlek är alltid *byte*

<ea> som destination			
Dn	X	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	-
(d8,An,Xn)	X	(d8,PC,Xn)	-
(bd,An,Xn)	X	(bd,PC,Xn)	-

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

TBLS "table look-up and interpolate signed"
TBLSN
TBLU "table look-up and interpolate unsigned"
TBLUN

Syntax:

```
TBLS.<x>      <ea>,Dx   Resultat avrundas
TBLSN.<x>     <ea>,Dx   Resultat avrundas ej
TBLU.<x>      Dym:Dyn,Dx  Resultat avrundas
TBLUN.<s>     Dym:Dyn,Dx  Resultat avrundas ej
```

Attribut:
Storlek <x> kan vara B (*byte*) W (*word*) eller L (*long*).
Beskrivning:
TBLS, TBLSN, TBLU och TBLUN används tillsammans med *tabeller* som innehåller styckevis linjära funktioner (formen "table lookup and interpolate" TBLx <ea>,Dn) eller *data i register* (formen "register interpolate" TBLx Dym:Dyn,Dx) för att bestämma funktionsvärden.
För "table lookup and interpolate" innehåller Dx (bit 15-0) den oberoende variabeln X. <ea> pekar ut en tabell, innehållande tal *med* tecken, storleken av dessa tal kan vara *byte*, *word* eller *long*. I tabellen ges en linjäriserad representation av den beroende variabeln Y som funktion av X. I allmänhet består den oberoende variabeln X av en 8-bitars heltalsdel och en 8-bitars bråktalsdel, dvs en *implicit* decimalpunkt mellan bit 7 och 8 i Dx. Heltalsdelen, skalad med storleken, används som offset i tabellen och bråktalsdelen används för att interpolera fram funktionsvärdet ur två konsekutiva poster i tabellen.
För "register interpolate" sker interpolationen mellan värden i register Dym och Dyn i stället för poster i tabellen. I denna form används endast bråktalsdelen och heltalsdelen ignoreras. Denna form kan användas tillsammans med "table lookup and interpolate" för att modellera funktioner av flera oberoend variabler.

Avrundning (TBLS)	
Justerad bråktalsdifferens	Avrundning
≤–1/2	–1
>–1/2 och <1/2	0
≥1/2	+1
Avrundning (TBLU)	
Justerad bråktalsdifferens	Avrundning
≥1/2	+1
<1/2	0

Returvärdet lagras i Dx. Bråktalsdelen placeras alltid i Dx[7..0], om storleken är *byte*{ XE "byte" } placeras heltalsdelen i Dx[15..8], om storleken är *word* placeras heltalsdelen i Dx[23..8], om storleken är *long* placeras de 24 minst signifikanta bitarna av heltalsdelen i Dx[31..8]. För TBLS och TBLSN *teckenutvidgas* heltalsdelen från *byte* och *word* resultat. För TBLU och TBLUN lämnas motsvarande delar av registret opåverkat.

<ea> som källa			
Dn	-	(xxx).W	X
An	-	(xxx).L	X
(An)	X	#<data>	-
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	1 om den mest signifikanta biten i resultatet är 1, 0 annars
Z	1 om resultatet är 0, 0 annars
V	1 om heltalsdelen av det icke avrundade resultatet är utanför området $-(2^{23}) \le$ Resultat $\le (2^{23})-1$, 0 annars
C	Nollställs alltid

TRAP "trap"

Syntax:

```
TRAP    #<Vector>
```

Beskrivning:
Processorn initierar *exception processing*. Vektornumret (0-15) anger vilken *exception* rutin som skall exekveras.
Anmärkning:
I *db68* används TRAP #14 för inbyggda subrutiner och TRAP #15 för brytpunktshantering.
Flaggor: Påverkas ej

TRAPcc "trap on condition"

Syntax:

```
TRAPcc
TRAPcc.W      #<data>
TRAPcc.L      #<data>
```

Beskrivning:
Om angivet villkor är *sant* som resultat av tidigare operation kommer *exception processing* (vektor 7) att vidtas. I annat fall är operationen ekvivalent med *no operation*. data kan användas av en exception-rutin för vidare hantering. Följande former kan användas:

TRAPHI	High	$\overline{C} \wedge \overline{Z}$
TRAPLS	Low or Same	$C \vee Z$
TRAPCC	Carry Clear	$\overline{C}$
TRAPCS	Carry Set	C
TRAPNE	Not Equal	$\overline{Z}$
TRAPEQ	Equal	Z
TRAPVC	Overflow Clear	$\overline{V}$
TRAPVS	Overflov Set	V
TRAPPL	Plus	$\overline{N}$
TRAPMI	Minus	N
TRAPGE	Greater or Equal	$N \wedge V \vee \overline{N} \wedge \overline{V}$
TRAPLT	Less Than	$N \wedge \overline{V} \vee \overline{N} \wedge V$
TRAPGT	Greater Than	$N \wedge V \wedge \overline{Z} \vee \overline{N} \wedge \overline{V} \wedge \overline{Z}$
TRAPLE	Less or Equal	$Z \vee N \wedge \overline{V} \vee \overline{N} \wedge V$
TRAPT	True	1
TRAPF	False	0

Flaggor: Påverkas ej

TRAPV "trap on overflow"

Syntax:

```
TRAPV
```

Beskrivning:
Om processorns V-flagga (overflow) är 1 som resultat av tidigare operation kommer *exception processing* att vidtas. I annat fall är operationen ekvivalent med *no operation*.
Flaggor: Påverkas ej

TST "test an operand"

Syntax:

```
TST.<x>       <ea>
```

Attribut:
Storlek <x> kan vara W eller L om operanden är ett adressregister. B, W eller L annars.
Beskrivning:
Jämför operanden med 0. Resultatet av jämförelsen påverkar innehållet i CC-registret.

<ea>			
Dn	X	(xxx).W	X
An	X	(xxx).L	X
(An)	X	#<data>	X
(An)+	X		
-(An)	X		
(d16,An)	X	(d16,PC)	X
(d8,An,Xn)	X	(d8,PC,Xn)	X
(bd,An,Xn)	X	(bd,PC,Xn)	X

Flaggor	
X	Påverkas ej
N	1 om resultatet negativt, 0 annars
Z	1 om resultatet är 0, 0 annars
V	Nollställs alltid
C	Nollställs alltid

UNLK "unlink"

Syntax:

```
UNLK          An
```

Beskrivning:
Inehållet i det angivna adressregistret kopieras till stackpekaren, därefter placeras adressen överst på stacken i adressregistret och stackpekaren ökas med 4.

Flaggor: Påverkas ej