

DAT 015 – Maskinorienterad Programmering 2011/2012

MC68HC12,
Arbetsbok för MC12
CPU12 Reference Guide

Ur innehållet:
Programmerarens bild
Översikt, "single-chip-computer" DG256
(Exempel)

Instruktionsuppsättning

"ISA" – Instruction Set Architecture

- ⊕ Vilka operationer kan utföras ?
 - Instruktionsgrupper
- ⊕ Hur lagras operanderna förutom i minnet ?
 - Korttidslagring
- ⊕ Hur nås operander i minnet?
 - Adresseringsätt
- ⊕ Vilka typer/storlekar av operander kan hanteras ?
 - Generella/speciella register, registerstorlek

Programmerarens bild – datatyper/storlek

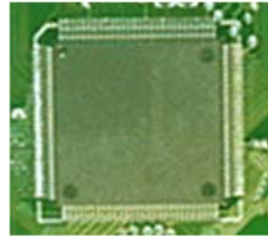
	char (8)	short int (16)	long int (32)	long int (64)	floating point (IEEE)	pointers
68HCS12	X	X				16/20 bit
Coldfire V1	X	X	X			32 bit
Coldfire V4	X	X	X		X	32 bit
PowerPC	X	X	X		X	32 bit
PowerPC (64)	X	X		X	X	64 bit
8086	X	X				16/20 bit
80386	X	X	X			32 bit
80486	X	X	X		X	32 bit
X86-32	X	X	X		X	32 bit
X86-64	X	X		X	X	64 bit

Programmerarens bild – adresserbart minne

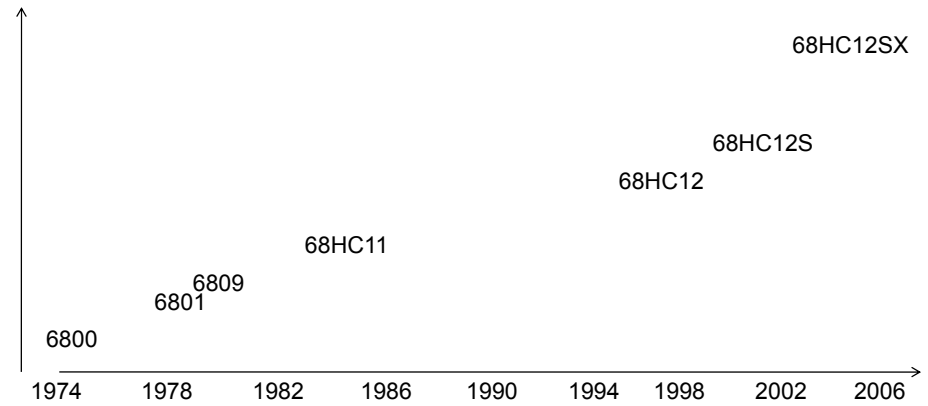
ADRESSBUSS	RANDOM ACCESS
16 bitar	$2^{16} = 65\,536$ byte = 64 kbyte
20 bitar	$2^{20} = 1\,048\,576$ byte = 1 024 kbyte = 1 Mbyte
24 bitar	$2^{24} = 16\,777\,216$ byte = 16 384 kbyte = 16 MByte
32 bitar	$2^{32} = 4\,294\,967\,296$ byte = 4 194 304 kbyte = 4 096 Mbyte = 4 Gbyte
64 bitar	$2^{64} = 1,844674407 \cdot 10^{19}$ byte = 16 Ebyte

Freescale 68HCS12

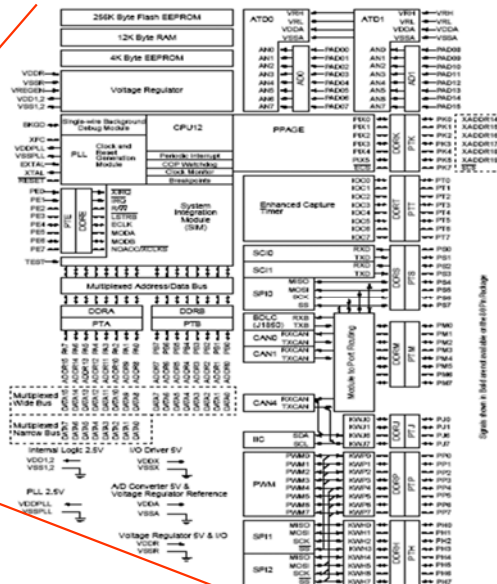
- HCS12 adressrum, IO och minne
- CPU12, klockor och räknare
- "Random Access"- Minne
 - RWM, FLASH, EEPROM
- Periferienheter
 - Parallell Input/Output:
 - Seriell kommunikation
 - AD
 - PWM



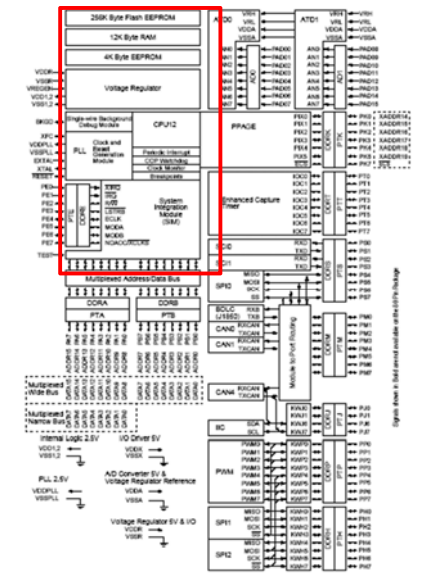
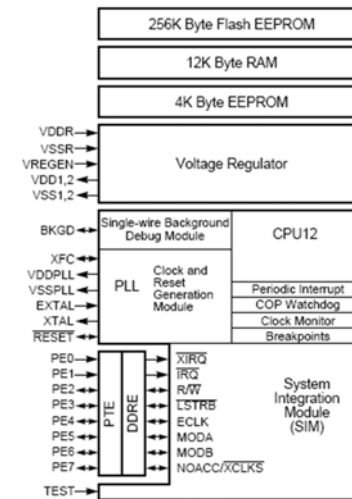
Historik



HCS12DG256, blockdiagram



HCS12DG256, "core"



HCS12DG256, "core"

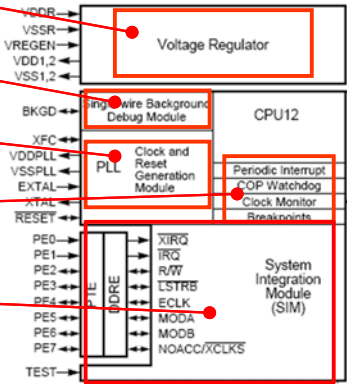
Spänningsregulatorer (flera olika spänningar används internt)

"Background Debug Mode" för test/avlusning

En kristall utgör bas för alla klockfrekvenser i systemet

Realtidsklocka och andra klockfunktioner

Programmerbara funktioner



Primärminne

Icke flyktigt minne

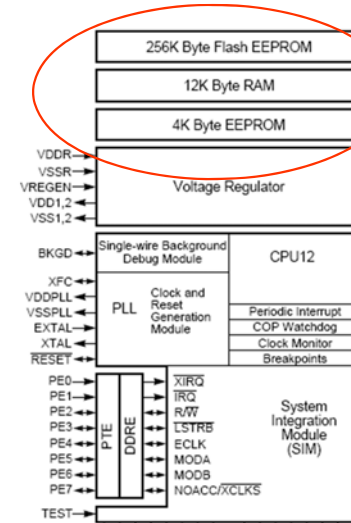
Upp till 256 Kbyte i "minnesbankar"

48 kB utan användning av "bankar"

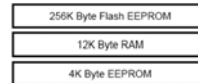
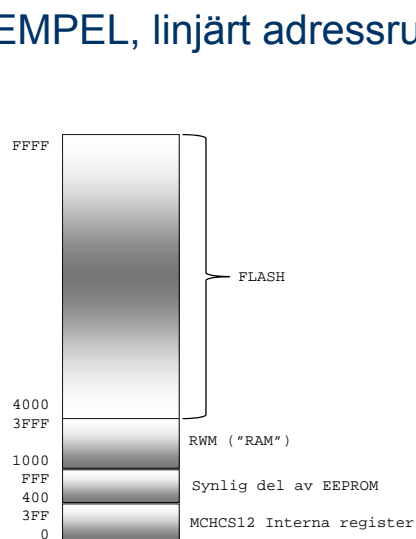
4 kB EEPROM

Flyktigt minne

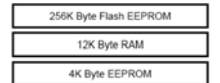
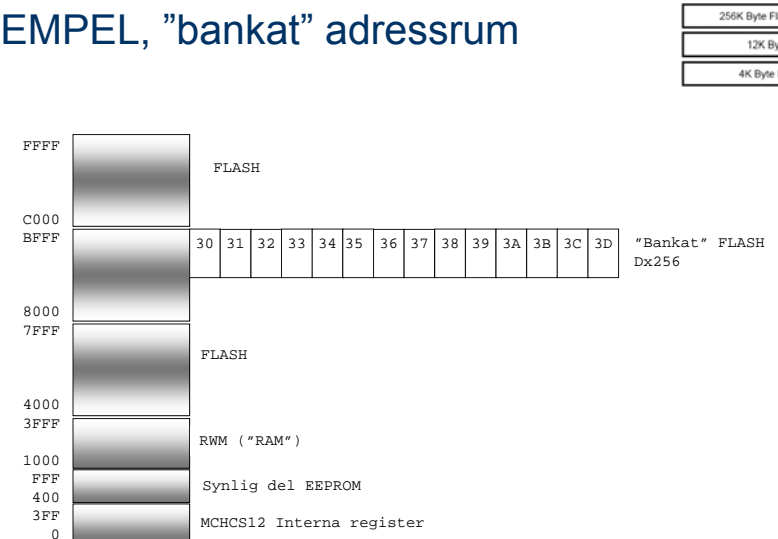
12 kB RAM (=RWM)



EXEMPEL, linjärt adressrum

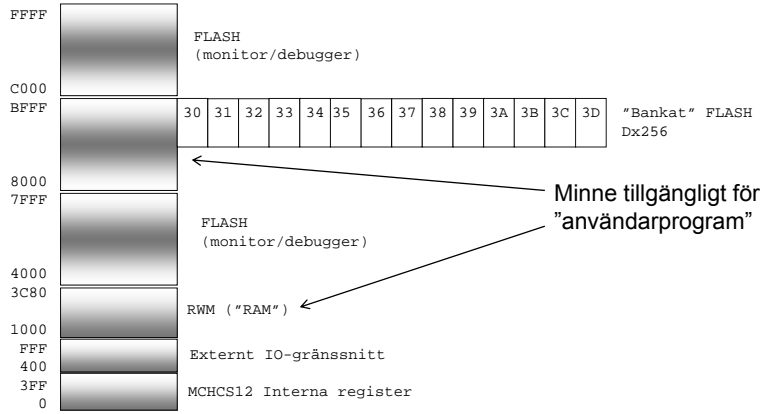


EXEMPEL, "bankat" adressrum



EXEMPEL, i laborationsdator MC12

256K Byte Flash EEPROM
12K Byte RAM
4K Byte EEPROM



Minne tillgängligt för "användarprogram"

Periferikretsar i HCS12DG256

AD – Analog till Digital omvandling

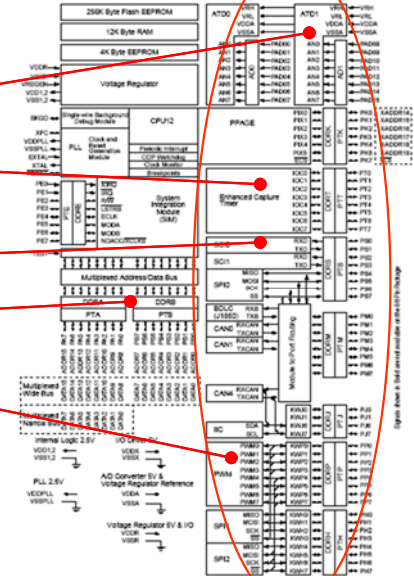
ECT- Räkarkretsar för noggrann tidmätning

SCI – Asynkron seriekommunikation

Parallell In-Utmatning

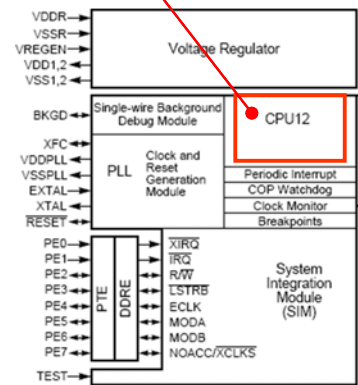
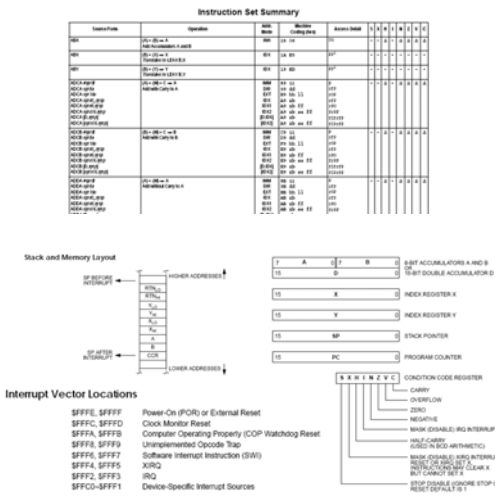
PWM – Pulsbreddsmodulering

Etc...

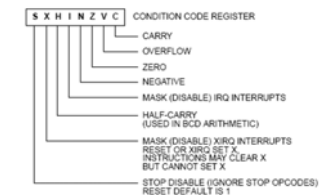
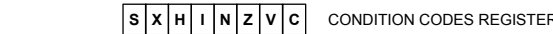
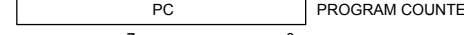
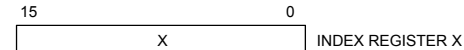
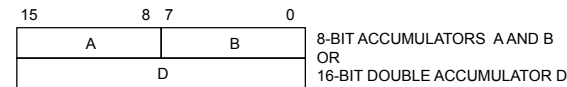


HCS12DG256, "core"

Centralenhet CPU12



Registeruppsättning CPU12



Adresseringsätt

Vi känner igen de flesta adresseringsätten i från FLEX.

Indexerade adresseringsätt kan även användas med register X, Y och SP ibland också med PC (PC-relativt)

Nytt här är också "Indirekt adressering"

Addressing Mode	Source Format	Abbreviation	Description
Inherent	INST (no externally supplied operands)	INH	Operands (if any) are in CPU registers
Immediate	INST #opr8i or INST #opr16i	IMM	Operand is included in instruction stream 8- or 16-bit size implied by context
Direct	INST opr8a	DIR	Operand is the lower 8-bits of an address in the range \$0000 - \$00FF
Extended	INST opr16a	EXT	Operand is a 16-bit address
Relative	INST rel8 or INST rel16	REL	An 8-bit or 16-bit relative offset from the current pc is supplied in the instruction
Indexed (5-bit offset)	INST oprx5,yssp	IDX	5-bit signed constant offset from x, y, sp, or pc
Indexed (pre-decrement)	INST oprx3,-yys	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8
Indexed (pre-increment)	INST oprx3,+yys	IDX	Auto pre-increment x, y, or sp by 1 ~ 8
Indexed (post-decrement)	INST oprx3,yys-	IDX	Auto post-decrement x, y, or sp by 1 ~ 8
Indexed (post-increment)	INST oprx3,yys+	IDX	Auto post-increment x, y, or sp by 1 ~ 8
Indexed (accumulator offset)	INST abd,yssp	IDX	Indexed with 8-bit (A or B) or 16-bit (D) accumulator offset from x, y, sp, or pc
Indexed (9-bit offset)	INST oprx9,yssp	IDX1	9-bit signed constant offset from x, y, sp, or pc (lower 8-bits of offset in one extension byte)
Indexed (16-bit offset)	INST oprx16,yssp	IDX2	16-bit constant offset from x, y, sp, or pc (16-bit offset in two extension bytes)
Indexed-Indirect (16-bit offset)	INST [oprx16,yssp]	[IDX2]	Pointer to operand is found at... 16-bit constant offset from x, y, sp, or pc (16-bit offset in two extension bytes)
Indexed-Indirect (D accumulator offset)	INST [D,yssp]	[D,IDX]	Pointer to operand is found at... x, y, sp, or pc plus the value in D

Inherent

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail	S	X	H	I	N	Z	V	C
CBA	(A) - (B) Compare 8-Bit Accumulators	INH	18 17	00	-	-	-	-	Δ	Δ	Δ	Δ

Maskinkod för instruktionen

Cykel för cykel beskrivning

Flaggpåverkan

Omedelbar (Immediate) 8-bit/16-bit

LDAA #opr8i	(M) ⇒ A	IMM	86 11	P	-	-	-	-	Δ	Δ	0	-
LDAA opr8a	Load Accumulator A	DIR	96 dd	rFP								
LDAA opr16a		EXT	B6 hh 11	rOP								
LDAA oprx0_yssp		IDX	A6 xb	rFP								
LDAA oprx9_yssp		IDX1	A6 xb ff	rPO								
LDAA oprx16_yssp		IDX2	A6 xb ee ff	rPPP								
LDAA [D,yssp]		[D,IDX]	A6 xb	rIFrFP								
LDAA [oprx16,yssol]		[IDX2]	A6 xb ee ff	rtrrFP								

LDD #opr16i	(MM+1) ⇒ A:B	IMM	CC jj kk	OP	-	-	-	-	Δ	Δ	0	-
LDD opr8a	Load Double Accumulator D (A:B)	DIR	DC dd	rFP								
LDD opr16a		EXT	FC hh 11	rOP								
LDD oprx0_yssp		IDX	BC xb	rFP								
LDD oprx9_yssp		IDX1	BC xb ff	rPO								
LDD oprx16_yssp		IDX2	BC xb ee ff	rPPP								
LDD [D,yssp]		[D,IDX]	BC xb	rIFrFP								
LDD [oprx16,yssp]		[IDX2]	BC xb ee ff	rIFrFP								

opr8i, 8-bitars konstant om 8-bitars register

Op16i, 16-bitars konstant om 16-bitars register

Direkt (Direct Page) Absolut (Extended)

LDAA #opr8i	(M) ⇒ A	IMM	86 11	P	-	-	-	-	Δ	Δ	0	-
LDAA opr8a	Load Accumulator A	DIR	96 dd	rFP								
LDAA opr16a		EXT	B6 hh 11	rOP								
LDAA oprx0_yssp		IDX	A6 xb	rFP								
LDAA oprx9_yssp		IDX1	A6 xb ff	rPO								
LDAA oprx16_yssp		IDX2	A6 xb ee ff	rPPP								
LDAA [D,yssp]		[D,IDX]	A6 xb	rIFrFP								
LDAA [oprx16,yssol]		[IDX2]	A6 xb ee ff	rtrrFP								

LDD #opr16i	(MM+1) ⇒ A:B	IMM	CC jj kk	OP	-	-	-	-	Δ	Δ	0	-
LDD opr8a	Load Double Accumulator D (A:B)	DIR	DC dd	rFP								
LDD opr16a		EXT	FC hh 11	rOP								
LDD oprx0_yssp		IDX	BC xb	rFP								
LDD oprx9_yssp		IDX1	BC xb ff	rPO								
LDD oprx16_yssp		IDX2	BC xb ee ff	rPPP								
LDD [D,yssp]		[D,IDX]	BC xb	rIFrFP								
LDD [oprx16,yssp]		[IDX2]	BC xb ee ff	rIFrFP								

opr16a, kan adressera hela adressintervallet 0000-FFFF

opr8a, kan enbart adressera intervallet 0000-00FF, anger minst signifikant byte av adressen

PC-relativ ("BRANCH"-instruktioner)

- 8-bitars offset (-128..127)
- 9-bitars offset (-256..255)
- 16-bitars offset (-32768..32767)

<code>BRA rel8</code>	Branch Always (if 1 = 1)	REL	20 rr
<code>IBEQ abd,ys, rel9</code>	(cnt) + 1 = cnt if (cnt) = 0, then Branch else Continue to next instruction Increment Counter and Branch if = 0 (cnt = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr
<code>IBNE abd,ys, rel9</code>	(cnt) + 1 = cnt if (cnt) not = 0, then Branch; else Continue to next instruction Increment Counter and Branch if = 0 (cnt = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr
<code>LBCC rel16</code>	Long Branch if Carry Clear (if C = 0)	REL	18 24 qq rr

Indexerade adresseringsätt:

- Register relativ, konstant offset

<code>LDAA #opr8</code>	(M) = A
<code>LDAA opr8a</code>	Load Accumulator A
<code>LDAA opr16a</code>	
<code>LDAA opr0_xysp</code>	
<code>LDAA opr9_xysp</code>	
<code>LDAA opr16_xysp</code>	
<code>LDAA [D,xysp]</code>	
<code>LDAA [oopr16,xysp]</code>	

`opr0_xysp` — Indexed addressing postbyte code:

<code>opr3,-xys</code>	Predecrement X or Y or SP by 1...8
<code>opr3,+xys</code>	Preincrement X or Y or SP by 1...8
<code>opr3,xys-</code>	Postdecrement X or Y or SP by 1...8
<code>opr3,xys+</code>	Postincrement X or Y or SP by 1...8
<code>opr5,xysp</code>	5-bit constant offset from X or Y or SP or PC
<code>abd,xysp</code>	Accumulator A or B or D offset from X or Y or SP or PC

`opr3` — Any positive integer 1...8 for pre/post increment/decrement
`opr5` — Any value in the range -16...+15
`opr9` — Any value in the range -256...+255
`opr16` — Any value in the range -32,768...65,535

Basregister kan vara något av: X,Y,SP,PC

EXEMPEL:

```
LDAA 5, X
STAA 20, Y
LDAA sym, PC
STA off, SP
...
```

Specialfall: n, PCR

```
LDAA sym, PCR
```

Antag PC pekar på nästa instruktion.

Operanden är här PC-sym, jfr offsetberäkning för "BRA"-instruktioner

Observera, ingen syntaktisk skillnad.
Assemblator väljer effektivast kodning

Indexerade adresseringsätt:

- Auto pre- increment/decrement
- Auto post- increment/decrement

<code>LDAA #opr8</code>	(M) = A	<code>opr0_xysp</code> — Indexed addressing postbyte code:
<code>LDAA opr8a</code>	Load Accumulator A	
<code>LDAA opr16a</code>		<code>opr3,-xys</code> Predecrement X or Y or SP by 1...8
<code>LDAA opr0_xysp</code>		<code>opr3,+xys</code> Preincrement X or Y or SP by 1...8
<code>LDAA opr9_xysp</code>		<code>opr3,xys-</code> Postdecrement X or Y or SP by 1...8
<code>LDAA opr16_xysp</code>		<code>opr3,xys+</code> Postincrement X or Y or SP by 1...8
<code>LDAA [D,xysp]</code>		<code>opr5,xysp</code> 5-bit constant offset from X or Y or SP or PC
<code>LDAA [oopr16,xysp]</code>		<code>abd,xysp</code> Accumulator A or B or D offset from X or Y or SP or PC
		<code>opr3</code> — Any positive integer 1...8 for pre/post increment/decrement
		<code>opr5</code> — Any value in the range -16...+15
		<code>opr9</code> — Any value in the range -256...+255
		<code>opr16</code> — Any value in the range -32,768...65,535

Basregister kan vara något av: X,Y,SP

EXEMPEL:

```
LDAA 1, -X
STAA 4, Y-
STAB 8, +SP
LDAB 7, SP+
...
```

Indexerade adresseringsätt:

- Register relativ, offset i ackumulator

<code>LDAA #opr8</code>	(M) = A
<code>LDAA opr8a</code>	Load Accumulator A
<code>LDAA opr16a</code>	
<code>LDAA opr0_xysp</code>	
<code>LDAA opr9_xysp</code>	
<code>LDAA opr16_xysp</code>	
<code>LDAA [D,xysp]</code>	
<code>LDAA [oopr16,xysp]</code>	

`opr0_xysp` — Indexed addressing postbyte code:

<code>opr3,-xys</code>	Predecrement X or Y or SP by 1...8
<code>opr3,+xys</code>	Preincrement X or Y or SP by 1...8
<code>opr3,xys-</code>	Postdecrement X or Y or SP by 1...8
<code>opr3,xys+</code>	Postincrement X or Y or SP by 1...8
<code>opr5,xysp</code>	5-bit constant offset from X or Y or SP or PC
<code>abd,xysp</code>	Accumulator A or B or D offset from X or Y or SP or PC

`opr3` — Any positive integer 1...8 for pre/post increment/decrement
`opr5` — Any value in the range -16...+15
`opr9` — Any value in the range -256...+255
`opr16` — Any value in the range -32,768...65,535

Basregister kan vara något av: X,Y,SP,PC

EXEMPEL:

```
LDAA A, X
STAA B, Y
STAB D, SP
LDAB D, PC
...
```

Indexerade adresseringssätt:

□ Indirekt

LDA #opr8	(M) ⇒ A
LDA opr8a	Load Accumulator A
LDA opr76a	
LDA opr0_xysp	
LDA opr9_xysp	
LDA opr16_xysp	
LDA [D_xysp]	
LDA [opr16_xysol]	

opr0_xysp — Indexed addressing postbyte code:

<i>opr3_xys</i>	Predecrement X or Y or SP by 1...8
<i>opr3_xys</i>	Preincrement X or Y or SP by 1...8
<i>opr3_xys-</i>	Postdecrement X or Y or SP by 1...8
<i>opr3_xys+</i>	Postincrement X or Y or SP by 1...8
<i>opr5_xysp</i>	5-bit constant offset from X or Y or SP or PC
<i>abd_xysp</i>	Accumulator A or B or D offset from X or Y or SP or PC

opr3 — Any positive integer 1...8 for pre/post increment/decrement
opr5 — Any value in the range -16...+15
opr9 — Any value in the range -256...+255
opr16 — Any value in the range -32,768...65,535

EXEMPEL:

LDA	[D, X]
STAA	[sym, PCR]
STAB	[2, SP]
LDAB	[D, Y]
...	

Instruktionsgrupper

LOAD-instruktioner

Mnemonic	Funktion	Operation
LDA	Load A	(M)→A
LDAB	Load B	(M)→B
LDD	Load D	(M:M+1) _i →A:B
LDS	Load SP	(M:M+1) _i →SP _i :SP _i
LDX	Load index register X	(M:M+1) _i →X _i :X _i
LDY	Load index register Y	(M:M+1) _i →Y _i :Y _i
LEAS	Load effective address into SP	Effective address→SP
LEAX	Load effective address into X	Effective address→X
LEAY	Load effective address into Y	Effective address→Y

STORE-instruktioner

Mnemonic	Funktion	Operation
STAA	Store A	(A)→M
STAB	Store B	(B)→M
STD	Store D	(A)→M _i :M _i →M+1
STS	Store SP	SP _i :SP _i →M:M+1
STX	Store X	X _i :X _i →M:M+1
STY	Store Y	Y _i :Y _i →M:M+1

MOVE-instruktioner

Mnemonic	Funktion	Operation
MOVB	Move byte (8 bitar)	(M _i)→M _i
MOVW	Move word (8 bitar)	(M:M+1) _i →M:M+1 _i

EXEMPEL: Kopiera byte

LDAB	\$3000
STAB	\$3001
eller	
LDA	\$3000
STAA	\$3001
eller	
MOVB	\$3000, \$3001

EXEMPEL: Kopiera word

LDD	\$3000
STD	\$3001
eller	
LDX	\$3000
STX	\$3001
eller	
LDY	\$3000
STY	\$3001
eller	
MOVW	\$3000, \$3001

Instruktioner för kopiering av registerinnehåll

Mnemonic	Funktion	Operation
TAB	Transfer A to B anm: Ekv. Med TFR A, B	(A)→B
TAP	Transfer A to CCR anm: Ekv. Med TFR A, CCR	(A)→CCR
TBA	Transfer B to A	(B)→A
TFR	Transfer register to register	(A,B,CCR,D,X,Y eller SP) → (A,B,CCR,D,X,Y eller SP)
TFA	Transfer CCR to A anm: Ekv. Med TFR CCR, A	(CCR)→A
TSX	Transfer SP to X anm: Ekv. Med TFR SP, X	(SP)→X
TSY	Transfer SP to Y anm: Ekv. Med TFR SP, Y	(SP)→Y
TXS	Transfer X to SP anm: Ekv. Med TFR X, SP	(X)→SP
TVS	Transfer Y to SP anm: Ekv. Med TFR Y, SP	(Y)→SP

← Använd denna

Övriga finns här av "kompatibilitetsskäl"

Instruktioner för växling av registerinnehåll

Mnemonic	Funktion	Operation
EXG	Exchange register to register	(A,B,CCR,D,X,Y eller SP) ↔ (A,B,CCR,D,X,Y eller SP)
XGDY	Exchange D with X anm: Ekv. Med EXG D, X - EXG X, D	(D) ↔ (X)
XGDX	Exchange D with Y anm: Ekv. Med EXG D, Y - EXG Y, D	(D) ↔ (Y)

← Använd denna

Övriga finns här av "kompatibilitetsskäl"

Instruktion för teckenutvidgning

Mnemonic	Funktion	Operation
SEX	Teckenutvidga 8 bitars operand	(A, B, CCR) → (D, X, Y eller SP)

Ovillkorlig programflödeskontroll

Mnemonic	Funktion	Operation
BSR	Anrop av subrutin. PC-relativ operand	SP-2 ⇒ SP RetAdrL:RetAdrH ⇒ M _(SP) :M _(SP+1) Adress ⇒ PC
BRA	"Hopp" till adress. PC-relativ operand	Adress ⇒ PC
CALL	Anrop av subrutin. Absolut operand (20 bitar) Anm: Användes vid programflödesändring mellan olika minnesbankar (\$8000-\$BFFF)	SP-2 ⇒ SP RetAdrL:RetAdrH ⇒ M _(SP) :M _(SP+1) Subrutinadress ⇒ PC SP-1 ⇒ SP (PPAGE) ⇒ M _(SP) PAGE ⇒ PPAGE Subrutinadress ⇒ PC
JMP	"Hopp" till adress. Absolut operand	Subrutinadress ⇒ PC
JSR	Anrop av subrutin. Absolut operand	SP-2 ⇒ SP RetAdrL:RetAdrH ⇒ M _(SP) :M _(SP+1) Subrutinadress ⇒ PC
RTC	Atervänd från subrutin. Returadress från STACK och PPAGE	M _(SP) ⇒ (PPAGE) SP+1 ⇒ SP M _(SP) :M _(SP+1) ⇒ PC _{hi} :PC _L SP+2 ⇒ SP
RTS	Atervänd från subrutin. Returadress från STACK	M _(SP) :M _(SP+1) ⇒ PC _{hi} :PC _L SP+2 ⇒ SP

Instruktioner för addition

Mnemonic	Funktion	Operation
ABA	Addera B till A	(A)+(B) → A
ABX	Addera B till X anm: Ekv. med LEAX B, X	(X)+(B) → X
ABY	Addera B till Y anm: Ekv. med LEAY B, Y	(Y)+(B) → Y
ADCA	Addition med carry till A	(A)+(M)+C → A
ADCB	Addition med carry till B	(B)+(M)+C → B
ADDA	Addition till A	(A)+(M) → A
ADDB	Addition till B	(B)+(M) → B
ADDD	Addition till D (A:B)	(D)+(M:M+1) → D

Mnemonic	Funktion	Operation
INC	Incrementera i minnet	(M)+\$01 → M
INCA	Incrementera A	(A)+\$01 → A
INCB	Incrementera B	(B)+\$01 → B
INS	Incrementera SP anm: Ekv. med LEAS 1, SP	(SP)+\$0001 → SP
INX	Incrementera X anm: Ekv. med LEAX 1, X	(X)+\$0001 → X
INY	Incrementera Y anm: Ekv. med LEAY 1, Y	(Y)+\$0001 → Y

Instruktioner för subtraktion

Mnemonic	Funktion	Operation
SBA	Subtrahera B från A	(A)-(B) → A
SBCA	Subtrahera med borrow från A	(A)-(M)-C → A
SBCB	Subtrahera med borrow från B	(B)-(M)-C → B
SUBA	Subtrahera från A	(A)-(M) → A
SUBB	Subtrahera från B	(B)-(M) → B
SUBD	Subtrahera från D (A:B)	(D)-(M:M+1) → D

Mnemonic	Funktion	Operation
DEC	Dekrementera i minnet	(M)-\$01 → M
DECA	Dekrementera A	(A)-\$01 → A
DECB	Dekrementera B	(B)-\$01 → B
DES	Dekrementera SP anm: Ekv. med LEAS -1, SP	(SP)-\$0001 → SP
DEX	Dekrementera X anm: Ekv. med LEAX -1, X	(X)-\$0001 → X
DEY	Dekrementera Y anm: Ekv. med LEAY -1, Y	(Y)-\$0001 → Y

Instruktioner för logikoperationer

Mnemonic	Funktion	Operation
ANDA	Bitvis "och" A med minnesinnehåll	(A)•(M) ⇒ A
ANDB	Bitvis "och" A med minnesinnehåll	(B)•(M) ⇒ B
ANDCC	Bitvis "och" CC med minnesinnehåll	(CCR)•(M) ⇒ CCR
EORA	Bitvis "exklusivt eller" A med minnesinnehåll	(A)⊕(M) ⇒ A
EORB	Bitvis "exklusivt eller" B med minnesinnehåll	(B)⊕(M) ⇒ B
ORAA	Bitvis "eller" A med minnesinnehåll	(A)+(M) ⇒ A
ORAB	Bitvis "eller" B med minnesinnehåll	(B)+(M) ⇒ B
ORCC	Bitvis "eller" CCR med minnesinnehåll	(CCR)+(M) ⇒ CCR

EXEMPEL: Nollställ bit 7-bit 4 på adress \$3000

```
LDAB $3000
ANDB #00001111
STAB $3000
```

EXEMPEL: Ettställ bit 7 och bit 0 på adress \$3000

```
LDAB $3000
ORAB #10000001
STAB $3000
```

Ynära operationer

Mnemonic	Funktion	Operation
CLC	Nollställ carryflaggan i CCR	0 ⇒ C
CLI	Nollställ avbrottsmask i CCR	0 ⇒ I
CLR	Nollställ minnesinnehåll	\$00 ⇒ M
CLRA	Nollställ A	\$00 ⇒ A
CLRB	Nollställ B	\$00 ⇒ B
CLV	Nollställ överflövsflaggan i CCR	0 ⇒ V
COM	Ettkomplementera minnesinnehåll	\$FF-(M) ⇒ M
COMA	Ettkomplementera A	\$FF-(A) ⇒ A
COMB	Ettkomplementera B	\$FF-(B) ⇒ B
NEG	Tvåkomplementera minnesinnehåll	\$00-(M) ⇒ M
NEGA	Tvåkomplementera A	\$00-(A) ⇒ A
NEGB	Tvåkomplementera B	\$00-(B) ⇒ B

EXEMPEL: Invertera bit 2 och bit1 på adress \$3000

```
LDAB $3000
EORB #00000110
STAB $3000
```

Logiska skiftoperationer

Mnemonic	Funktion	Operation
LSL	Logiskt vänsterskift i minnet	
LSLA	Logiskt vänsterskift A	
LSLB	Logiskt vänsterskift B	
LSLD	Logiskt vänsterskift D	
LSR	Logiskt högerskift i minnet	
LSRA	Logiskt högerskift A	
LSRB	Logiskt högerskift B	
LSRD	Logiskt högerskift D	

Exempel på användning:

Multiplikation med 2, tal utan tecken.
Division med 2, tal utan tecken.

Aritmetiska skiftoperationer

Mnemonic	Funktion	Operation
ASL	Aritmetiskt vänsterskift i minnet (ekv. med LSL)	
ASLA	Aritmetiskt vänsterskift A (ekv. med LSLA)	
ASLB	Aritmetiskt vänsterskift B (ekv. med LSLB)	
ASLD	Aritmetiskt vänsterskift D (ekv. med LSLD)	
ASR	Aritmetiskt högerskift i minnet	
ASRA	Aritmetiskt högerskift A	
ASRB	Aritmetiskt högerskift B	

Exempel på användning, högerskift:

Division med 2, tal med tecken.

Instruktioner för rotation (carry-skift)

Mnemonic	Funktion	Operation
ROL	Rotation vänster via carry i minnet	
ROLA	Rotation vänster via carry A	
ROLB	Rotation vänster via carry B	
ROR	Rotation höger via carry i minnet	
RORA	Rotation höger via carry A	
RORB	Rotation höger via carry B	

EXEMPEL: Skifta ett 32-bitars tal på adress \$3000, 1 steg åt höger

```
LSR $3000
ROR $3001
ROR $3002
ROR $3003
```

Exempel på användning:

Skiftoperationer på tal större än 8 bitar.

Instruktioner för jämförelser och test

Mnemonic	Funktion	Operation
CBA	Jämför B med A	(A)-(B)
CMPA	Jämför A med minne	(A)-(M)
CMPB	Jämför B med minne	(B)-(M)
CPD	Jämför D med minne	(A,B)-(M:M+1)
CPS	Jämför SP med minne	(SP)-(M:M+1)
CPX	Jämför X med minne	(X)-(M:M+1)
CPY	Jämför Y med minne	(Y)-(M:M+1)

JÄMFÖRELSE
Två operander
BINÄR operation

Mnemonic	Funktion	Operation
TST	Testa minnesinnehåll	(M)-\$00
TSTA	Testa register A	(A)-\$00
TSTB	Testa register B	(B)-\$00

TEST
En operand
UNÄR operation

Villkorlig programflödeskontroll

Mnemonic	Funktion	Villkor
Enkla flaggtest		
BCS	"Hopp" om carry	C=1
BCC	"Hopp" om ICKE carry	C=0
BEQ	"Hopp" om zero	Z=1
BNE	"Hopp" om ICKE zero	Z=0
BMI	"Hopp" om negative	N=1
BPL	"Hopp" om ICKE negative	N=0
BVS	"Hopp" om overflow	V=1
BVC	"Hopp" om ICKE overflow	V=0
Test av tal utan tecken		
BHI	Villkor: R>M	C + Z = 0
BHS	Villkor: R≥M	C=0
BLO	Villkor: R<M	C=1
BLS	Villkor: R≤M	C + Z = 1
Test av tal med tecken		
BGT	Villkor: R>M	Z + (N ⊕ V) = 0
BGE	Villkor: R≥M	N ⊕ V = 0
BLT	Villkor: R<M	N ⊕ V = 1
BLE	Villkor: R≤M	Z + (N ⊕ V) = 1

Används typiskt tillsammans med jämförelse/test instruktioner.

EXEMPEL

```
LDAB $3000
CMPB $3001
BEQ L1
...
```

Sammansatta instruktioner.
EXEMPEL
DBEQ B,L2
samma sak som
DECB
BEQ L2

Instruktioner för räkande programlingor

Mnemonic	Funktion	Villkor
DBEQ	Dekrementera innehåll i register. "Hoppa" om resultatet = 0. (register: A,B,D,X,Y,SP)	(register) - 1 ⇒ register om(register)=0; "hoppla"; annars: nästa instruktion
DBNE	Dekrementera innehåll i register. "Hoppa" om resultatet ≠ 0. (register: A,B,D,X,Y,SP)	(register) - 1 ⇒ register om(register)≠0; "hoppla"; annars: nästa instruktion
IBEQ	Inkrementera innehåll i register. "Hoppa" om resultatet = 0. (register: A,B,D,X,Y,SP)	(register) + 1 ⇒ register om(register)=0; "hoppla"; annars: nästa instruktion
IBNE	Inkrementera innehåll i register. "Hoppa" om resultatet ≠ 0. (register: A,B,D,X,Y,SP)	(register) + 1 ⇒ register om(register)≠0; "hoppla"; annars: nästa instruktion
TBEQ	Testa innehåll i register. "Hoppa" om resultatet = 0. (register: A,B,D,X,Y,SP)	om(register)=0; "hoppla"; annars: nästa instruktion
TBNE	Testa innehåll i register. "Hoppa" om resultatet ≠ 0. (register: A,B,D,X,Y,SP)	om(register)≠0; "hoppla"; annars: nästa instruktion