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?
  - – Adresseringssätt
- Vilka typer/storlekar av operander kan hanteras ?
  - – Generella/speciella register, registerstorlek

Programmerarens bild – datatyper/storlek

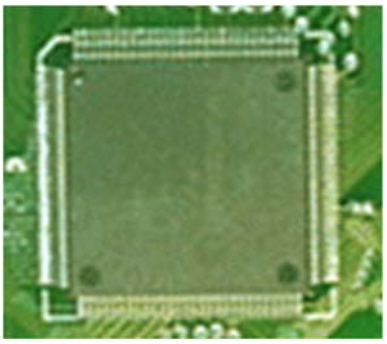
	char (8)	short int (16)	long int (32)	long int (64)	floating point (IEEE)	pointers
68HCS12	X	X				16/20 bit
Coldfire V1	X	X	X			32 bit
Coldfire V4	X	X	X		X	32 bit
PowerPC	X	X	X		X	32 bit
PowerPC (64)	X	X		X	X	64 bit
8086	X	X				16/20 bit
80386	X	X	X			32 bit
80486	X	X	X		X	32 bit
X86-32	X	X	X		X	32 bit
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 10^{19} byte = 16 Ebyte

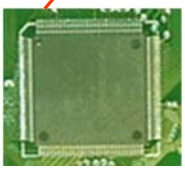
Freescale 68HCS12

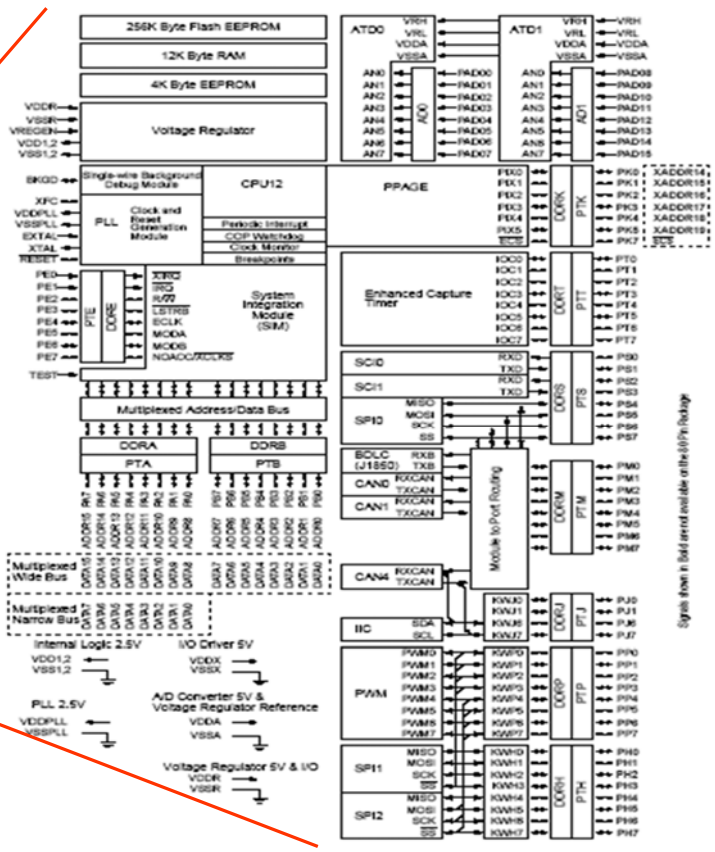
- HCS12 adressrum, IO och minne
- CPU12, klockor och räknare
- ”Random Access”- Minne
  - RWM, FLASH, EEPROM
- Periferienheter
  - Parallell Input/Output:
  - Seriell kommunikation
  - AD
  - PWM



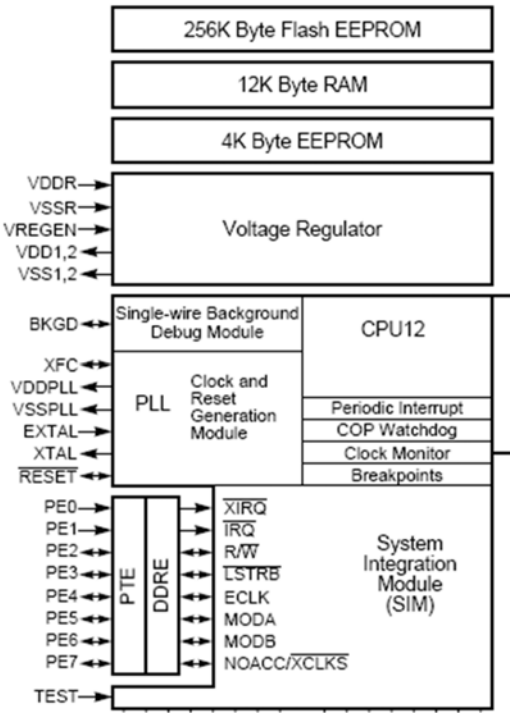
Historik

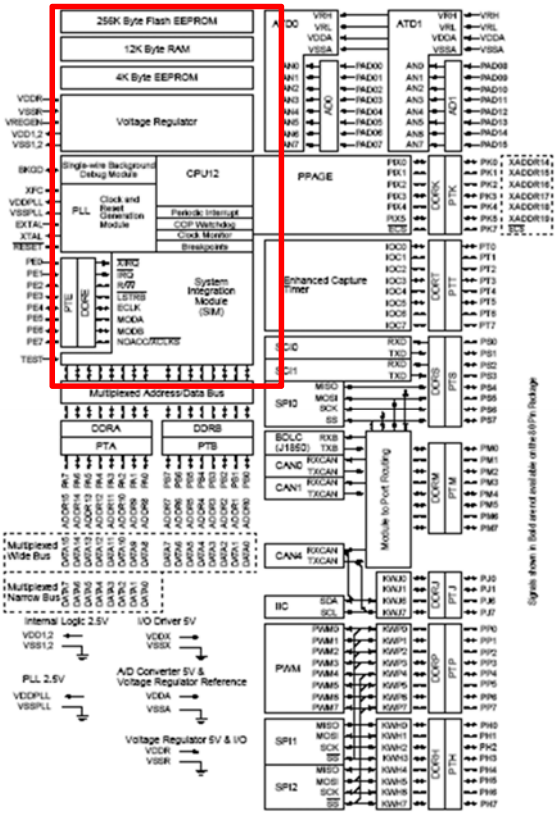
HCS12DG256, blockdiagram





HCS12DG256, ”core”





HCS12DG256, ”core”

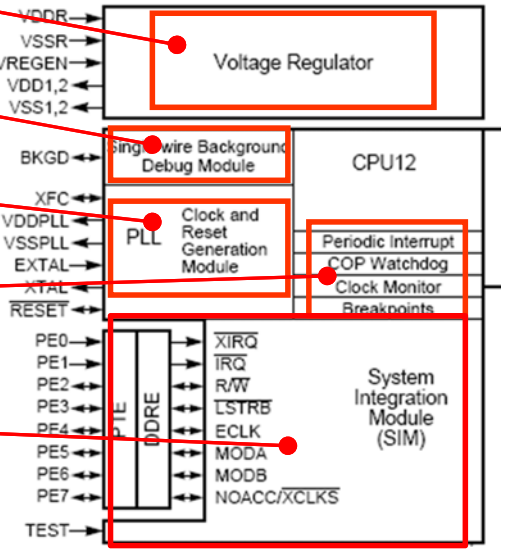
Spänningsregulatorer (flera olika spänningar används internt)

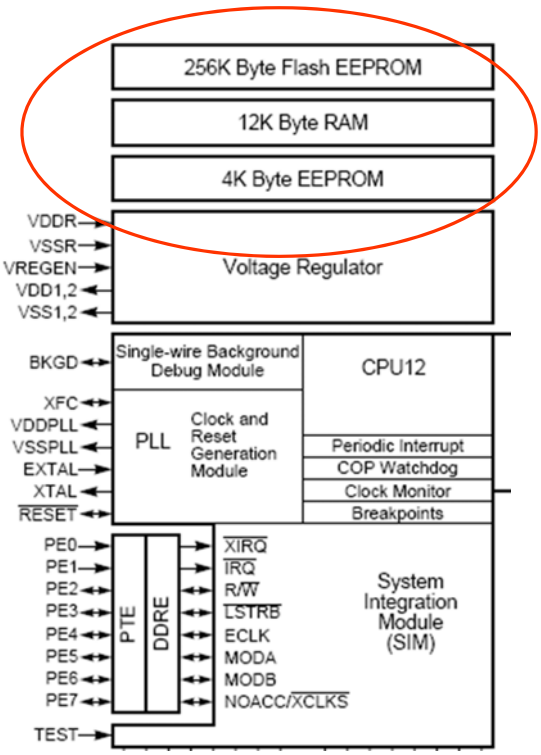
”Background Debug Mode” för test/avlusning

En kristall utgör bas för alla klockfrekvenser i systemet

Realtidsklocka och andra klockfunktioner

Programmerbara funktioner





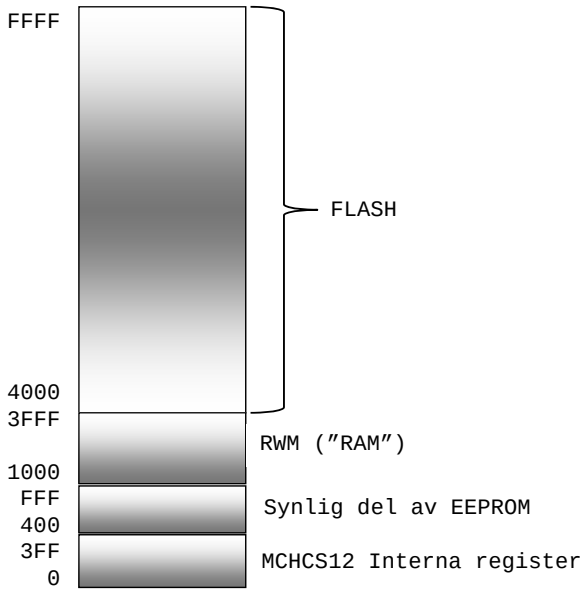
Primärminne

Icke flyktigt minne
- Upp till 256 Kbyte i ”minnesbankar”
- 48 kB utan användning av ”bankar”
- 4 kB EEPROM

Flyktigt minne
- 12 kB RAM (=RWM)

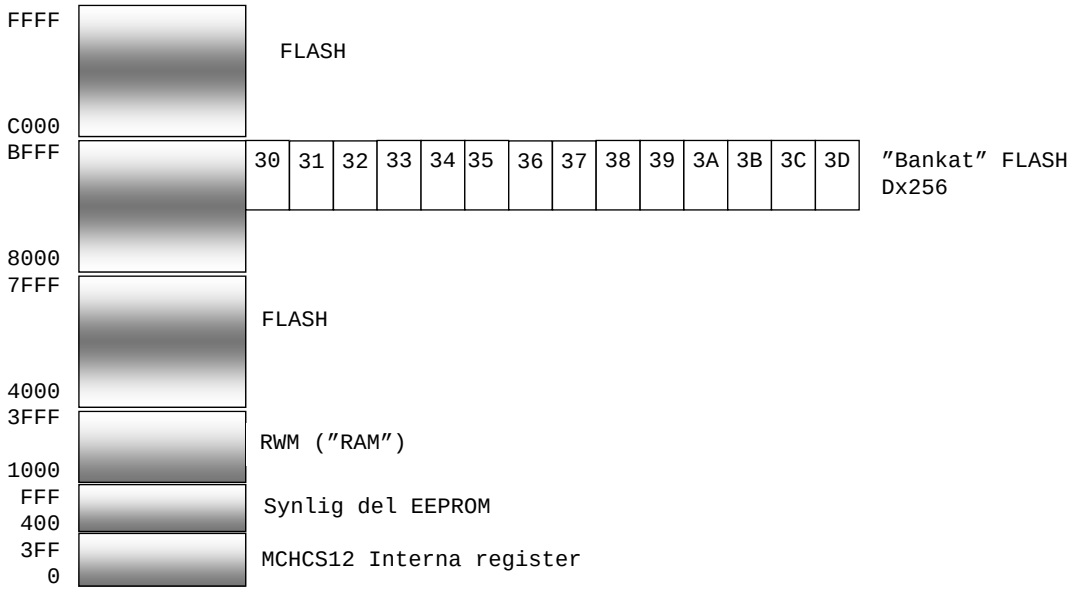
EXEMPEL, linjärt adressrum

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

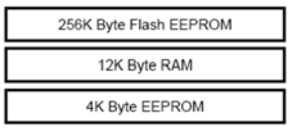


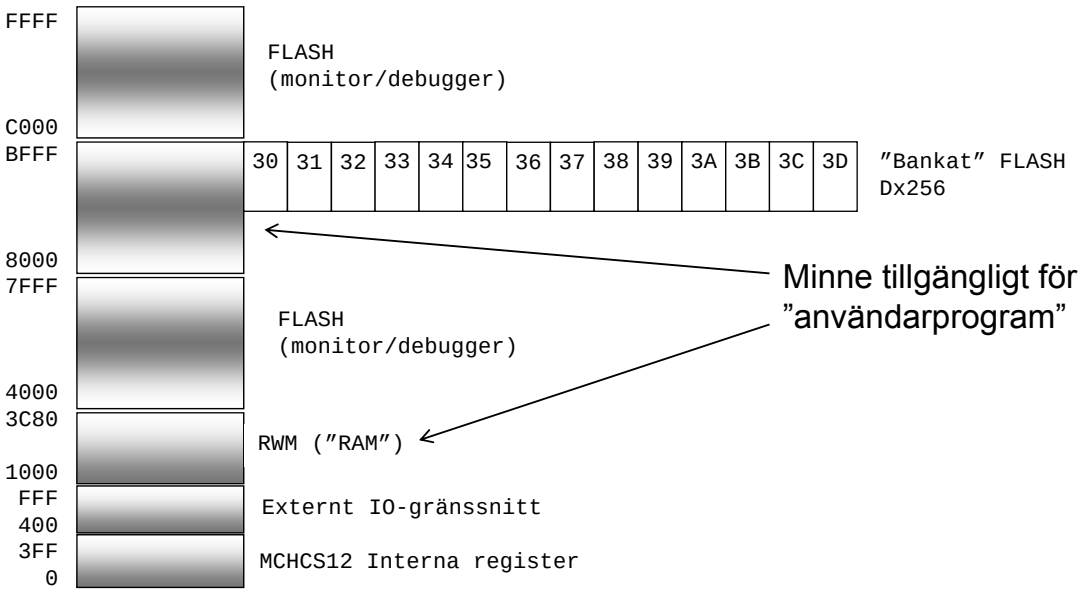
EXEMPEL, ”bankat” adressrum

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



EXEMPEL, i laborationsdator MC12





Periferikretsar i HCS12DG256

AD – Analog till Digital omvandling

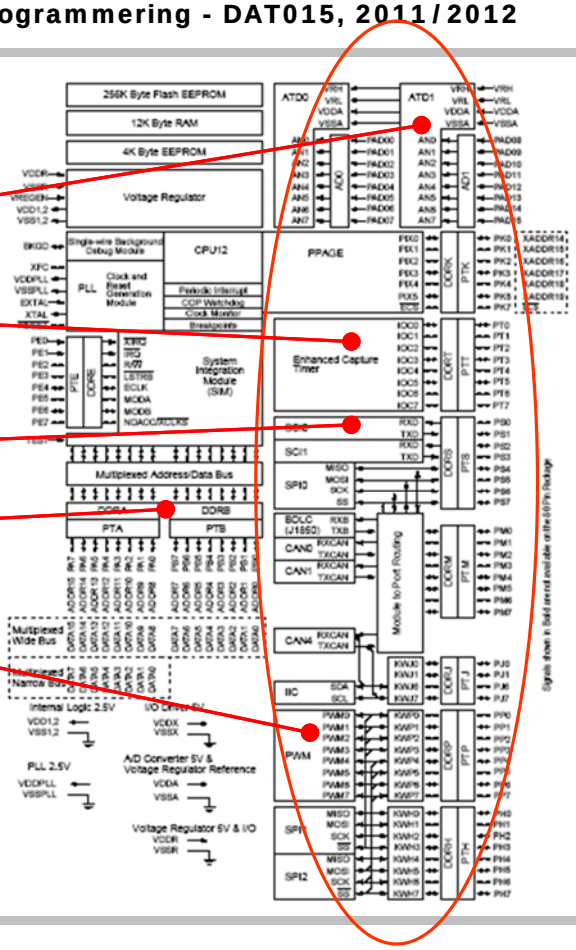
ECT- Räknarkretsar för noggrann tidmätning

SCI – Asynkron seriekommunikation

Parallell In-Utmatning

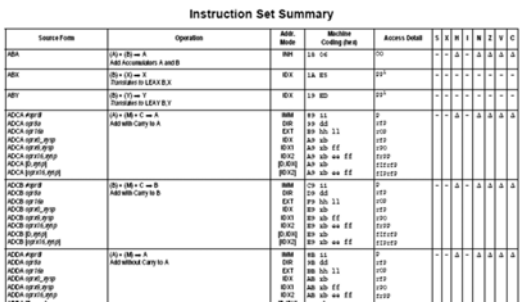
PWM – Pulsbreddsmodulering

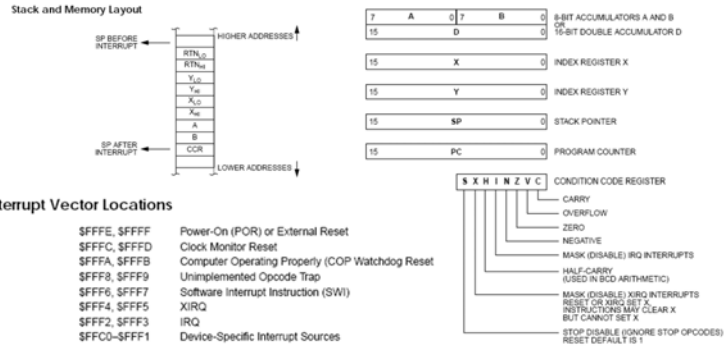
Etc…



HCS12DG256, ”core”

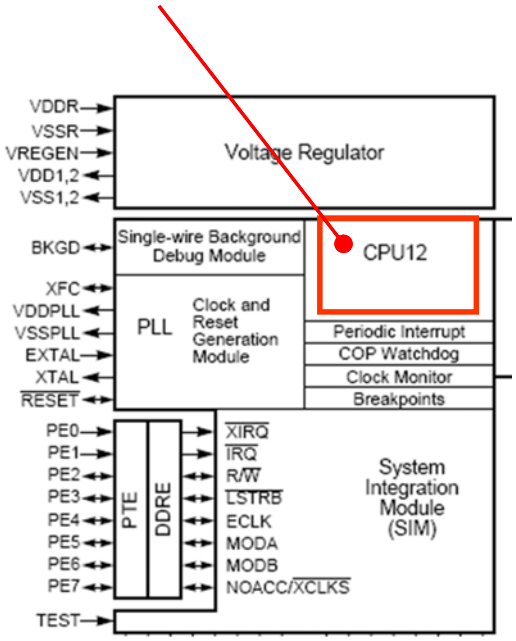
Centralenhet CPU12



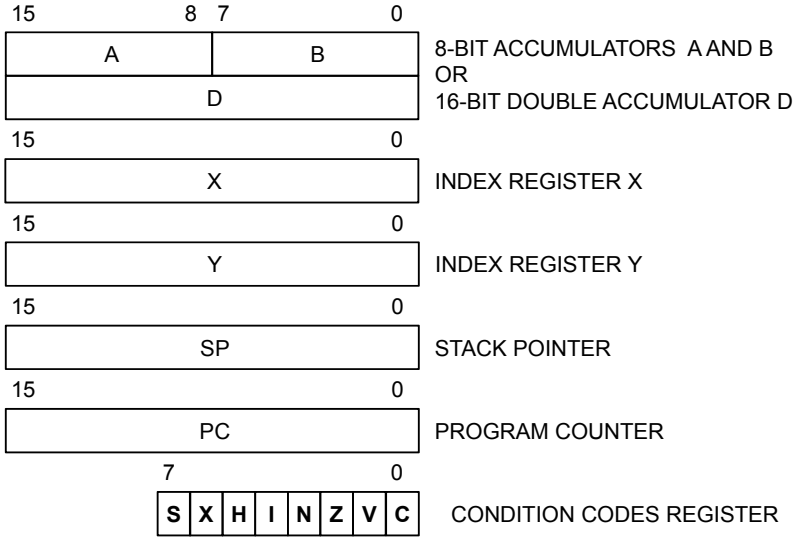


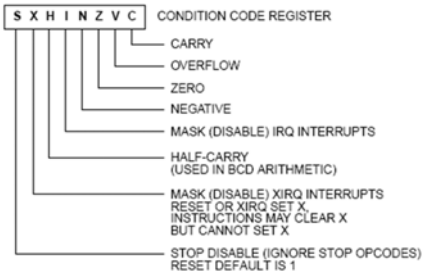
Interrupt Vector Locations

$FFFE, $FFFF	Power-On (POR) or External Reset
$FFFC, $FFFD	Clock Monitor Reset
$FFFA, $FFFB	Computer Operating Properly (COP Watchdog Reset
$FFF8, $FFF9	Unimplemented Opcode Trap
$FFF6, $FFF7	Software Interrupt Instruction (SWI)
$FFF4, $FFF5	XIRQ
$FFF2, $FFF3	IRQ
$FFC0–$FFF1	Device-Specific Interrupt Sources



Registeruppsättning CPU12





Adresseringssätt

Vi känner igen de flesta adresseringssätten i från FLEX.

Indexerade adresseringssätt 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,xysp*	IDX	5-bit signed constant offset from x, y, sp, or pc
Indexed (pre-decrement)	**INST** *oprx3,–xys*	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8
Indexed (pre-increment)	**INST** *oprx3,+xys*	IDX	Auto pre-increment x, y, or sp by 1 ~ 8
Indexed (post-decrement)	**INST** *oprx3,xys–*	IDX	Auto post-decrement x, y, or sp by 1 ~ 8
Indexed (post-increment)	**INST** *oprx3,xys+*	IDX	Auto post-increment x, y, or sp by 1 ~ 8
Indexed (accumulator offset)	**INST** *abd,xysp*	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,xysp*	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,xysp*	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,xysp*]	[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,*xysp*]	[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	∞	–	–	–	–	Δ	Δ	Δ	Δ

Maskinkod för instruktionen

Cykel för cykel beskrivning

Flaggpåverkan

Omedelbar (Immediate) 8-bit/16-bit

LDAA #*opr8i*	(M) ⇒ A	IMM	86 ii	P	–	–	–	–	Δ	Δ	0	–
LDAA *opr8a*	Load Accumulator A	DIR	96 dd	rfP								
LDAA *opr16a*		EXT	B6 hh ll	rOP								
LDAA *oprx0_xysp*		IDX	A6 xb	rfP								
LDAA *oprx9,xysp*		IDX1	A6 xb ff	rPO								
LDAA *oprx16,xysp*		IDX2	A6 xb ee ff	frPP								
LDAA [D,*xysp*]		[D,IDX]	A6 xb	fIfrfP								
LDAA [*oprx16,xysp*]		[IDX2]	A6 xb ee ff	fIPrfP								
LDD #*opr16i*	(M:M+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 ll	ROP								
LDD *oprx0_xysp*		IDX	EC xb	RfP								
LDD *oprx9,xysp*		IDX1	EC xb ff	RPO								
LDD *oprx16,xysp*		IDX2	EC xb ee ff	fRPP								
LDD [D,*xysp*]		[D,IDX]	EC xb	fIfRfP								
LDD [*oprx16,xysp*]		[IDX2]	EC xb ee ff	fIPRfP								

opr8i, 8-bitars konstant om 8-bitars register

Opr16i, 16-bitars konstant om 16-bitars register

Direkt (Direct Page)

Absolut (Extented)

LDAA #*opr8i*	(M) ⇒ A	IMM	86 ii	P	–	–	–	–	Δ	Δ	0	–
LDAA *opr8a*	Load Accumulator A	DIR	96 dd	rfP								
LDAA *opr16a*		EXT	B6 hh ll	rOP								
LDAA *oprx0_xysp*		IDX	A6 xb	rfP								
LDAA *oprx9,xysp*		IDX1	A6 xb ff	rPO								
LDAA *oprx16,xysp*		IDX2	A6 xb ee ff	frPP								
LDAA [D,*xysp*]		[D,IDX]	A6 xb	fIfrfP								
LDAA [*oprx16,xysp*]		[IDX2]	A6 xb ee ff	fIPrfP								
LDD #*opr16i*	(M:M+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 ll	ROP								
LDD *oprx0_xysp*		IDX	EC xb	RfP								
LDD *oprx9,xysp*		IDX1	EC xb ff	RPO								
LDD *oprx16,xysp*		IDX2	EC xb ee ff	fRPP								
LDD [D,*xysp*]		[D,IDX]	EC xb	fIfRfP								
LDD [*oprx16,xysp*]		[IDX2]	EC xb ee ff	fIPRfP								

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)

BRA *rel8*	Branch Always (if 1 = 1)	REL	`20 rr`

IBEQ *abdxys, rel9*	(cntr) + 1⇒ cntr If (cntr) = 0, then Branch else Continue to next instruction Increment Counter and Branch if = 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	`04 lb rr`
IBNE *abdxys, rel9*	(cntr) + 1⇒ cntr if (cntr) not = 0, then Branch; else Continue to next instruction Increment Counter and Branch if ≠ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	`04 lb rr`

LBCC *rel16*	Long Branch if Carry Clear (if C = 0)	REL	`18 24 qq rr`

Indexerade adresseringssätt:

- Register relativ, konstant offset

LDAA #*opr8i*
LDAA *opr8a*
LDAA *opr16a*
LDAA *oprx0_xysp*
LDAA *oprx9,xysp*
LDAA *oprx16,xysp*
LDAA [D,*xysp*]
LDAA [*oprx16,xysp*]
	(M) ⇒ A
	Load Accumulator A

oprx0_xysp — Indexed addressing postbyte code:

oprx3,–xys	Predecrement X or Y or SP by 1 . . . 8
oprx3,+xys	Preincrement X or Y or SP by 1 . . . 8
oprx3,xys–	Postdecrement X or Y or SP by 1 . . . 8
oprx3,xys+	Postincrement X or Y or SP by 1 . . . 8
oprx5,xysp	5-bit constant offset from X or Y or SP or PC
abd,xysp	Accumulator A or B or D offset from X or Y or SP or PC

oprx3 — Any positive integer 1 . . . 8 for pre/post increment/decrement
oprx5 — Any value in the range –16 . . . +15
oprx9 — Any value in the range –256 . . . +255
oprx16 — 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
...
```

Observera, ingen syntaktisk skillnad.
Assemblator väljer effektivast kodning

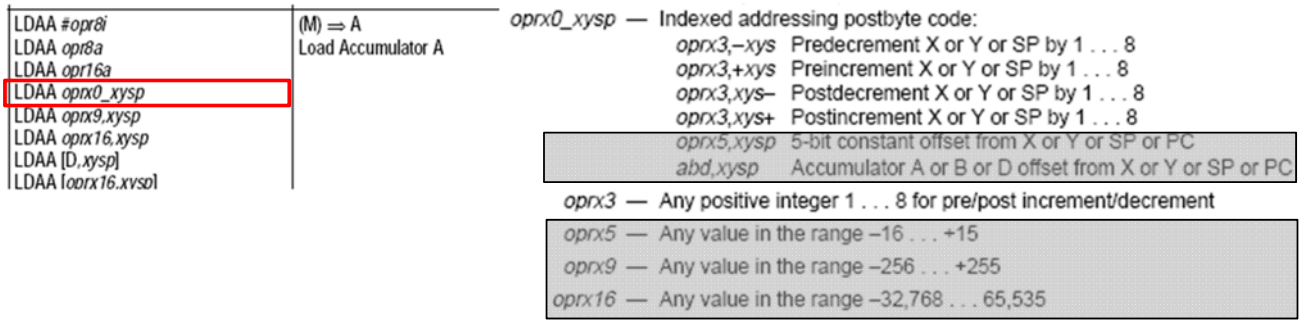
Specialfall: `n,PCR`

```
LDAA    sym,PCR
```

Antag `PC` pekar på nästa instruktion.
Operanden är här `PC-sym`, jfr offsetberäkning för "BRA"-instruktioner

Indexerade adresseringssätt:

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



LDAA #opr8i	(M) ⇒ A
LDAA opr8a	Load Accumulator A
LDAA opr16a	
LDAA oprx0_xysp	
LDAA oprx9,xysp	
LDAA oprx16,xysp	
LDAA [D,xysp]	
LDAA [oprx16,xysp]	

oprx0_xysp — Indexed addressing postbyte code:

oprx3,–xys	Predecrement X or Y or SP by 1 . . . 8
oprx3,+xys	Preincrement X or Y or SP by 1 . . . 8
oprx3,xys–	Postdecrement X or Y or SP by 1 . . . 8
oprx3,xys+	Postincrement X or Y or SP by 1 . . . 8
oprx5,xysp	5-bit constant offset from X or Y or SP or PC
abd,xysp	Accumulator A or B or D offset from X or Y or SP or PC

oprx3 — Any positive integer 1 . . . 8 for pre/post increment/decrement

oprx5 — Any value in the range –16 . . . +15

oprx9 — Any value in the range –256 . . . +255

oprx16 — Any value in the range –32,768 . . . 65,535

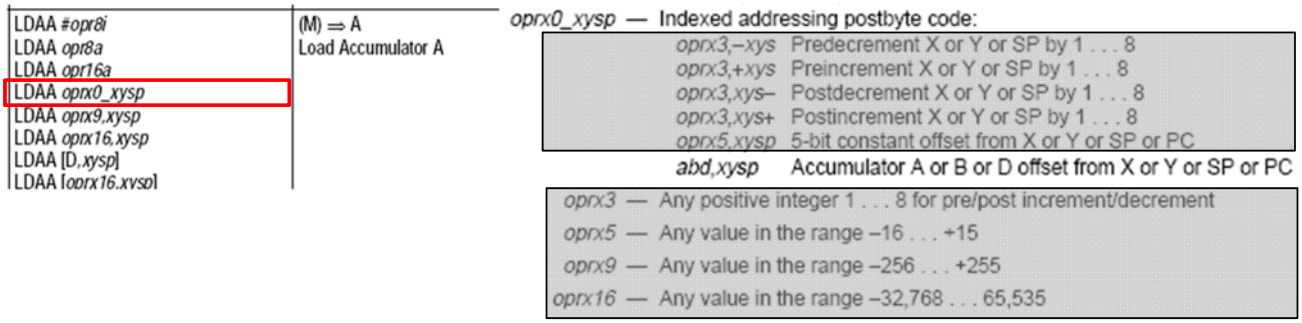
Basregister kan vara något av: X,Y,SP

EXEMPEL:

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


Indexerade adresseringssätt:

- Register relativ, offset i ackumulator



LDAA #opr8i	(M) ⇒ A
LDAA opr8a	Load Accumulator A
LDAA opr16a	
LDAA oprx0_xysp	
LDAA oprx9,xysp	
LDAA oprx16,xysp	
LDAA [D,xysp]	
LDAA [oprx16,xysp]	

oprx0_xysp — Indexed addressing postbyte code:

oprx3,–xys	Predecrement X or Y or SP by 1 . . . 8
oprx3,+xys	Preincrement X or Y or SP by 1 . . . 8
oprx3,xys–	Postdecrement X or Y or SP by 1 . . . 8
oprx3,xys+	Postincrement X or Y or SP by 1 . . . 8
oprx5,xysp	5-bit constant offset from X or Y or SP or PC
abd,xysp	Accumulator A or B or D offset from X or Y or SP or PC

oprx3 — Any positive integer 1 . . . 8 for pre/post increment/decrement

oprx5 — Any value in the range –16 . . . +15

oprx9 — Any value in the range –256 . . . +255

oprx16 — 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

LDAA #opr8i	(M) ⇒ A
LDAA opr8a	Load Accumulator A
LDAA opr16a	
LDAA oprx0_xysp	
LDAA oprx9,xysp	
LDAA oprx16,xysp	
LDAA [D,xysp]	
LDAA [oprx16,xysp]	

oprx0_xysp — Indexed addressing postbyte code:

oprx3,–xys	Predecrement X or Y or SP by 1 . . . 8
oprx3,+xys	Preincrement X or Y or SP by 1 . . . 8
oprx3,xys–	Postdecrement X or Y or SP by 1 . . . 8
oprx3,xys+	Postincrement X or Y or SP by 1 . . . 8
oprx5,xysp	5-bit constant offset from X or Y or SP or PC
abd,xysp	Accumulator A or B or D offset from X or Y or SP or PC

oprx3 — Any positive integer 1 . . . 8 for pre/post increment/decrement

oprx5 — Any value in the range –16 . . . +15

oprx9 — Any value in the range –256 . . . +255

oprx16 — Any value in the range –32,768 . . . 65,535

EXEMPEL:

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


Instruktionsgrupper

LOAD-instruktioner

Mnemonic	Funktion	Operation
LDAA	Load A	(M)→A
LDAB	Load B	(M)→B
LDD	Load D	$(M:M+1)_1 \rightarrow A:B$
LDS	Load SP	$(M:M+1)_1 \rightarrow SP_H:SP_L$
LDX	Load index register X	$(M:M+1)_1 \rightarrow X_H:X_L$
LDY	Load index register Y	$(M:M+1)_1 \rightarrow Y_H:Y_L$
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, (B)→M+1
STS	Store SP	$SP_H:SP_L \rightarrow M:M+1$
STX	Store X	$X_H:X_L \rightarrow M:M+1$
STY	Store Y	$Y_H:Y_L \rightarrow M:M+1$

MOVE-instruktioner

Mnemonic	Funktion	Operation
MOVB	Move byte (8 bitar)	$(M_1) \rightarrow M_2$
MOVW	Move word (8 bitar)	$(M:M+1)_1 \rightarrow M:M+1_2$

EXEMPEL: Kopiera *byte*

```
      LDAB   $3000
      STAB   $3001
eller
      LDAA   $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)**
TPA	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
TYS	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)**
XGDX	Exchange D with X anm: Ekv. Med `EXG D,X - EXG X,D`	(D) ↔ (X)
XGDY	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)

CHALMERS Chalmers University of Technology
Maskinorienterad Programmering - DAT015, 2011/2012

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 address. Absolut operand	Subrutinadress ⇒ PC
JSR	Anrop av subrutin Absolut operand	SP-2 ⇒ SP RetAdrL:RetAdrH ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Subrutinadress ⇒ PC
RTC	Återvänd från subrutin. Returadress från STACK och PPAGE	$M_{(SP)}$ ⇒ (PPAGE) SP+1 ⇒ SP $M_{(SP)}$:$M_{(SP+1)}$ ⇒ PC_H:PC_L SP+2 ⇒ SP
RTS	Återvänd från subrutin. Returadress från STACK	$M_{(SP)}$:$M_{(SP+1)}$ ⇒ PC_H: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	Inkrementera A	(A)+ $01 → A
INCB	Inkrementera B	(B)+ $01 → B
INS	Inkrementera SP anm: Ekv. med LEAS 1,SP	(SP)+ $0001 → SP
INX	Inkrementera X anm: Ekv. med LEAX 1,X	(X)+ $0001 → X
INY	Inkrementera 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

Unära 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 overflowflaggan I CCR	0 ⇒ V
COM	Ettkomplementera minnesinnehåll	$FF-(M) ⇒ M
COMA	Ettkomplementera A	$FF-(A) ⇒ A
COMB	Ettkomplementera B	$FF-(B) ⇒ A
NEG	Tvåkomplementera minnesinnehåll	$00-(M) ⇒ M
NEGA	Tvåkomplementera A	$00-(A) ⇒ A
NEGB	Tvåkomplementera B	$00-(B) ⇒ B

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

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

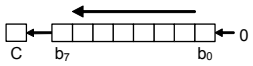
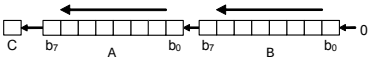
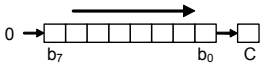
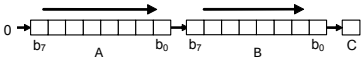
EXEMPEL: Ettställ bit 7 och bit 0 på adress $3000

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

EXEMPEL: Invertera bit 2 och bit1 på adress $3000

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

Logiska skiftoperationer

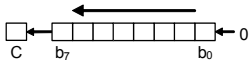
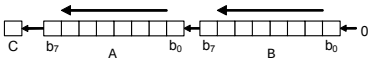
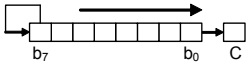
Mnemonic	Funktion	Operation
LSL	Logiskt vänsterskift i minnet	
LSLA	Logiskt vänsterskift A	
LSLB	Logiskt vänsterskift B	
LSLD	Logiskt vänsterskift D	
LSR	Logiskt högerskift i minnet	
LSRA	Logiskt högerskift A	
LSRB	Logiskt högerskift B	
LSRD	Logiskt högerskift D	

Exempel på användning:

Multiplikation med 2, tal utan tecken.

Division med 2, tal utan tecken.

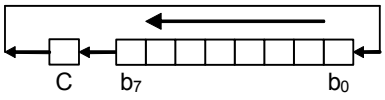
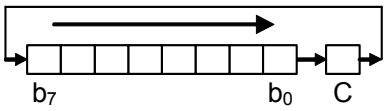
Aritmetiska skiftoperationer

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
      ....
```


Instruktioner för räknande programslingor

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; "hoppa"; 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; "hoppa"; 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; "hoppa"; 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; "hoppa"; annars: nästa instruktion
TBEQ	Testa innehåll i register. "Hoppa" om resultatet = 0. (register: A,B,D,X,Y,SP)	om(register)=0; "hoppa"; annars: nästa instruktion
TBNE	Testa innehåll i register. "Hoppa" om resultatet ≠ 0. (register: A,B,D,X,Y,SP)	om(register)≠0; "hoppa"; annars: nästa instruktion

Sammansatta instruktioner.
EXEMPEL

```
      DBEQ    B,L2
```

samma sak som

```
      DECB
      BEQ     L2
```