Programmering i maskinkode på *AMIGA*

A.Forness & N.A.Holten

Hæfte 11

Indhold

DATASKOLEN
Postboks 62
Nordengen 18
2980 Kokkedal

Telefon 49 18 00 77

Postgiro 7 24 23 44

CIA-CHIPS

I dette brev vil vi se nærmere på ind/ud-portene på AMIGA. Der findes to CHIPS inde i AMIGAen, som er med til at styre bl.a parallel-porten, serie-porten, joysticks-portene, osv. Disse CHIPs kaldes CIA-CHIPs. Forkortelsen CIA står ikke for "den amerikanske efterretningstjeneste" - "Central Intelligence Agency", men derimod for COMPLEX INTERFACE ADAPTER. Dette kan frit oversættes til Avanceret Grænseflade tilpasnings-enhed. De to CIA-chips hedder CIA-A og CIA-B.

Lad os nu fortsætte med en oversigt over adresserne som disse CHIPs benytter. Læg nøje mærke til at disse adresser kun kan adresseres med BYTE-længde (f.eks MOVE.**B**).

CIA-A:

ADRESSE	**NAVN**	**BETYDNING**
$BFE001	PRA	Dataregister A
$BFE101	PRB	Dataregister B
$BFE201	DDRA	Dataretning register A
$BFE301	DRB	Dataretning register B
$BFE401	TALO	TIMER A LOW-register
$BFE501	TAHI	TIMER A HIGH-register
$BFE601	TBLO	TIMER B LOW-register
$BFE701	TBHI	TIMER B HIGH-register
$BFE801		Vertikal-SYNC-tæller (BIT 0-7)
$BFE901		Vertikal-SYNC-tæller (BIT 8-15)
$BFEA01		Vertikal-SYNC-tæller (BIT 16-23)
$BFEB01		Ikke benyttet
$BFEC01	SDR	Serielt dataregister
$BFED01	ICR	INTERRUPT kontrolregister
$BFEE01	CRA	Kontrolregister A
$BFEF01	CRB	Kontrolregister B

CIA-B:

ADRESSE	NAVN	BETYDNING
$BFD000	PRA	Dataregister A
$BFD100	PRB	Dataregister B
$BFD200	DDRA	Dataretning register A
$BFD300	DDRB	Dataretning register B
$BFD400	TALO	TIMER A LOW-register
$BFD500	TAHI	TIMER A HIGH-register
$BFD600	TBLO	TIMER B LOW-register
$BFD700	THBI	TIMER B HIGH-register
$BFD800		Horisontal-SYNC-tæller (BIT 0-7)
$BFD900		Horisontal-SYNC-tæller (BIT 8-15)
BFDA000		Horisontal-SYNC-tæller (BIT 16-23)
$BFDB00		Ikke benyttet
$BFDC00	SDR	Serielt dataregister
$BFDD00	ICR	INTERRUPT kontrolregister
$BFDE00	CRA	Kontrolregister A
$BFDF00	CRB	Kontrolregister B

Dette var opsætningen for begge CIA-CHIPS. Vi vil ikke gennemgå alle funktionerne i dette brev.

Lad os tage fat på de fire første registre i hver CIA-CHIP, nemlig PRA, PRB, DDRA og DDRB.

Hver af disse CHIPs er fysisk udstyret med 16 elektriske "porte" som kan sættes til at være enten indgange eller udgange. Med registrene DDRA og DDRB bestemmes hvilke porte, som skal være indgange og hvilke som skal være udgange. I registrene PRA og PRB kan værdierne på de porte som er indgange aflæses og værdierne på udgangene sættes (bestemmes). Læg mærke til at dette er såkaldte "digitale porte". Dette betyder at de kun kan have to forskellige værdier, enten 0 eller 1, altså "ikke strøm" eller "strøm" (egentlig spænding).

Her kommer opsætningen af PRA- og PRB-registrene i begge CIA-CHIPs (sådan som de opfører sig i AMIGA).

CIA-A PRA ($BFE001):

BIT	FUNKTION
7	Fire-knap joystickport 1
6	Fire-knap joystickport 0 (venstre musknap)
5	RDY (disk)
4	TK0 (disk)
3	WPRO (disk)
2	CHNG (disk)
1	LED (styring af POWER-lampen)
0	OVL (system)

CIA-A PRB ($BFE101):

BIT	FUNKTION
0-7	Parallelport data 0-7

CIA-B PRA ($BFD000):

BIT	FUNKTION
7	DTR (serieport)
6	RTS (serieport)
5	CD (serieport)
4	CTS (serieport)
3	DSR (serieport)
2	SEL (parallelport)
1	POUT (parallelport)
0	BUSY (parallelport)

CIA-B PRB ($BFD100):

BIT	FUNKTION
7	MTR (disk)
6	SEL3 (disk)
5	SEL2 (disk)
4	SEL1 (disk)
3	SEL0 (disk)
2	SIDE (disk)
1	DIR (disk)
0	STEP (disk)

Vi har nu vist adresserne på de forskellige registre i CHIPene og en dybere specifikation af opsætningen af PRA- og PRB-registrene. I næste kapitel fortsætter vi med forklaringen af programeksempel MC1101.

AFLÆSNING AF MUSEN

I dette kapitel gennemgås programeksempel MC1101, nemlig aflæsning af musens position.

Når vi skal læse musens position (dens X- og Y-koordinater), benyttes register $DFF00A. Dette register indeholder to 8-BITs værdier. BIT 0-7 indeholder X-positionen, mens BIT 8-15 indeholder Y-positionen. Hver position kan altså indeholde værdier fra 0 til 255 (8 BITs).

Når du flytter musen til højre, vil X-positionen "blive større". Flytter du musen til venstre, vil X-positionen mindskes.

Ulempen ved at bruge disse værdier direkte er, at når musen flyttes så langt at X-positionen bliver over 255, vil registret "gå rundt" og starte forfra fra 0 igen. Vi bliver derfor nødt til at vi lave en rutine, som udregner afstanden mellem den nye position og den forrige.

Studer eksemplerne nedenfor:

GAMMEL POSITION	NY POSITION	FORSKYDNING
130	150	+20
100	75	-25
250	10	+16
50	255	-51
180	180	0
0	255	-1
100	200	+100
100	250	-106

Når vi skal udregne dette gøres følgende:

Lad os sige at "NY POSITION" er 250, og "GAMMEL POSITION" er 100. Først trækkes den gamle position fra den nye.

250 - 100 = 150

Hvis resultatet er større end 127, TRÆKKES 256 FRA.

Hvis resultatet er mindre end -128, LÆGGES DET TIL 256.

I dette tilfælde blev værdien større end 127, og vi skal derfor trække 256 fra.

150 - 256 = -106

Altså: Forskydningen blev -106.

Lad os tage nogle flere eksempler:

```
NY POSITION     = 22
GAMMEL POSITION = 200

22 - 200 = -178

værdien er mindre end -128....

-178 + 256 = 78

FORSKYDNING = +78


NY POSITION     = 160
GAMMEL POSITION = 200

160 - 200 = -40

værdien er hverken for lille eller for stor....

FORSKYDNING = -40
```

Det skulle være til at forstå nu, ikke sandt?

Nu forklarer vi programeksempel MC1101:

Linie 1: Her begynder hovedrutinen.

Linie 2: Hopper til rutinen "mouse".

Linie 4: Lægger adressen på "mousex" ind i A1.

Linie 5: Lægger adressen på "mousey" ind i A2.

Linie 7-8: Lægger X-positionen ind i D1, og Y-positionen ind i D2. Værdierne bliver ikke benyttet til noget som helst i dette programeksempel, men du kan jo selv prøve at lave et program, som styrer f.eks en SPRITE ved hjælp af musen.

Linie 12-13: Checker om venstre musknap er trykket ned. Hvis ikke, hop tilbage til "main".

Linie 17: Her begynder selve mus-rutinen.

Linie 18: Lagrer alle registre på STACKen.

Linie 19: Lægger værdien som ligger på adresse $DFF00A ind i D0.

Linie 20: Udfører en AND så det kun er de første 8 BITs (BIT 0-7) som bliver tilbage i D0.

Linie 21: Lægger værdien 0 ind i D2. Denne værdi angiver den mindste X-position vi vil have.

Linie 22: Lægger værdien 639 ind i D3. Denne værdi angiver den højeste X-position. Vi har nu sat området for X-positionen til at have værdier mellem 0 og 639.

Linie 23: Lægger adressen på "oldx" ind i A1.

Linie 24: Lægger adressen på "mousex" ind i A2.

Linie 25: Hopper til rutinen "calcmouse".

Linie 26-33: Her gentages det hele for Y-positionen. Området for Y-positionen har vi sat til værdierne mellem 0 og 511.

Linie 34: Lægger de gamle registerværdier tilbage.

Linie 35: Afslutter rutinen.

Linie 36: Her begynder den rutine, som udregner forskydningen og checker at musen holdes indenfor det opsatte område (X = 0 - 639, og Y = 0 - 511).

Linie 37: Lægger værdien 0 ind i D1.

Linie 38: Lægger værdien i "oldx/oldy" ind i D1.

Linie 39: Lægger værdien som ligger i D0 ind i "oldx/oldy".

Linie 40: Lægger værdien i D0 ind i D5.

Linie 41: Lægger værdien i D1 ind i D6.

Linie 42: Trækker D1 fra i D0. Her trækkes den gamle position fra den nye.

Linie 43: Sammenligner D1 med -128.

Linie 44: Hvis D1 er mindre end -128, hop til "mc_less".

Linie 45: Sammenligner D1 med 127.

Linie 46: Hvis D1 er større end 127, hop til "mc_more".

Linie 47: Sammenligner D1 med 0.

Linie 48: Hvis D1 er mindre end 0, hop til "mc_chk2".

Linie 49: Herfra fortsættes der, hvis D1 er større end (eller lig med) 0.

Linie 50: Sammenligner D6 med D5.

Linie 51: Hvis D6 er større end D5, hop til "mc_chk1ok".

Linie 52: Bytter fortegn i D1. Altså: Hvis D1 var 100 bliver D1 -100, og omvendt.

Linie 54: Hopper til "mc_storem".

Linie 56: Sammenligner D6 med D5.

Linie 57: Hvis D6 er mindre end D5, hop til "mc_chk2ok".

Linie 58: Bytter fortegn i D1.

Linie 60: Hopper til "mc_storem".

Linie 62: Lægger 256 til i D1.

Linie 63: Hopper til "mc_storem".

Linie 65: Trækker 256 fra i D1.

Linie 67: Bytter fortegn i D1.

Linie 68: Lægger D1 til i "mousex/mousey".

Linie 69: Lægger "mousex/mousey" ind i D0.

Linie 70: Sammenligner D0 med D2.

Linie 71: Hvis D0 er mindre end D2, hoppes der til "mc_tosmall". Altså: Hvis man prøver at bevæge musen udenfor det opsatte område.

Linie 72: Sammenligner D0 med D3.

Linie 73: Hvis D0 er større end D3, hop til "mc_tolarge". Dette sker, hvis man flytter musen udenfor området.

Linie 74: Afslutter rutinen ("calcmouse").

Linie 76: Lægger D2 ind i (A2). Hvis værdien er for lille til at være indenfor området, lægges den mindst lovlige værdi direkte ind i "mousex/mousey".

Linie 77: Afslutter rutinen.

Linie 79: Lægger den højeste lovlige værdi direkte ind i "mousex/mosey".

Linie 80: Afslutter rutinen.

Linie 81-88: Her ligger variablerne deklareret.

I det næste kapitel skal vi se nærmere på AMIGAens printerport.

PARALLELPORTEN

I dette kapitel gennemgås printer-porten (parallelporten) på AMIGA.

Parallelporten er en grænseflade, som kan benyttes til at styre en printer. Parallel-grænsefladen overfører data parallelt. Dette vil sige at der overføres 8 BITs (1 BYTE) ad gangen (seriel-porten overfører derimod kun 1 BIT ad gangen). Teknisk set fungerer parallelporten således:

- Først sender AMIGA en BYTE til printeren.

- AMIGAen venter til printeren har bearbejdet dataene. Når printeren har udført sit, vil den sende et "klar"-signal tilbage til AMIGAen. Når AMIGAen har modtaget "klar"-signalet sendes eventuelt en BYTE til.

Printeren har også mulighed for at sende andre beskeder til AMIGAen. Udover "klar"-signalet (READY), kan printeren give besked om at den er tom for papir, eller at printeren ikke er "ONLINE"". Vi kan også checke om der findes printer tilkoblet, eller om printeren ikke er tændt.

.... og så til programeksempel MC1102:

Linie 1: Lægger adressen på "buffer" ind i A0. I "buffer" ligger tegnene (som skal sendes til printeren) deklareret.

Linie 3: Slukker alle INTERRUPTs. Dette skal gøres for at undgå problemer med operativsystemet (AMIGA-DOS). Hvis ikke det gøres, vil du få en "PRINTER TROUBLE"- besked op på skærmen efter at have printet et kort stykke tid.

Linie 4: Hopper til rutinen "print". Det er denne rutine som udfører selve printningen.

Linie 5: Tænder for alle INTERRUPTs igen.

Linie 7: Afslutter programmet.

Linie 9-11: Her ligger tegnene som skal printes. Værdien 10 i slutningen af sætningen, angiver et linieskift (LINEFEED,LF). Værdien 0 lægges altid i slutningen af teksten således at "print"-rutinen finder slutningen på dataene.

Linie 14: Her starter "print"-rutinen.

Linie 15: Lægger værdien $FF ind på $BFE301. Dette sikrer at alle linier på PRB-porten bliver udgange (der skal jo sendes data til printeren).

Linie 18: Lægger værdien som ligger på adresse $BFD000 ind i D0. På denne adresse ligger de signaler som printeren sender til AMIGAen.

Linie 19: Udfører en logisk AND. Dette resulterer i at kun BIT 0-2 bliver stående tilbage. Læg mærke til at vi opgiver værdien BINÆRT (#%111 = #7).

Linie 20: Sammenligner D0 med %100 (4 DECIMALT).

Linie 21: Hvis D0 er lig %100, hoppes der til "ready". Altså: Printeren er klar til at tage imod et tegn.

Linie 22: Sammenligner D0 med %001.

Linie 23: Hvis D0 er lig %001, hop til "offline". Dette betyder at printerens "ONLINE"-knap er slukket.

Linie 24: Sammenligner D0 med %111.

Linie 25: Hvis D0 er lig %111, hoppes der til "poweroff". Altså: Printeren er ikke tændt. Læg mærke til at det samme sker hvis ingen printer er tilkoblet AMIGAen.

Linie 26: Sammenligner D0 med %001.

Linie 27: Hvis D0 er lig %001, hoppes der til "wait".

Linie 28: Sammenligner D0 med %011.

Linie 29: Hvis D0 er lig %011, hop til "paperout". Altså: Hvis printeren løber tør for papir, vil der blive hoppet til "paperout".

Linie 30: Hopper op igen til "wait".

Linie 32: Her begynder rutinen, som sender et tegn til printeren.

Linie 33: Lægger værdien, som findes på adressen, som A0 peger på ind i D0, og lægger 1 til i A0. Altså: Lægger tegnet, som skal sendes, ind i D0.

Linie 34: Sammenligner D0 med 0.

Linie 35: Hvis D0 er lig 0, hoppes der til "stop". Dette betyder at teksten er slut.

Linie 36: Lægger værdien som ligger i D0 ind på adresse $BFE101. Dette medfører, at der bliver sendt et tegn til printeren.

Linie 37: Hopper tilbage til "wait".

Linie 39: Her starter rutinen, som udføres når printningen er færdig.

Linie 40: Lægger værdien 0 ind i D0. Dette gøres fordi du skal kunne checke om der er opstået en fejl ved printningen. Altså: Når "print"-rutinen returnerer 0 i D0, var printningen fejlfri.

Linie 41: Afslutter rutinen.

Linie 43: Her ligger rutinen som udføres når printeren ikke er tændt (eller ingen printer er tilkoblet).

Linie 44: Lægger værdien 1 ind i D0. Altså: Hvis der returneres 1 i D0, ved vi, at enten er printeren ikke tændt, eller der er ikke tilkoblet en printer.

Linie 45: Afslutter rutinen.

Linie 47: Her ligger rutinen, som udføres når printeren ikke er "ONLINE" (altså "OFFLINE").

Linie 48: Lægger værdien 2 i D0. Altså: Hvis D0 returnerer 2, ved vi at printeren er "OFFLINE".

Linie 49: Afslutter rutinen.

Linie 51: Her ligger den rutine, som udføres når printeren er tom for papir.

Linie 52: Lægger værdien 3 ind i D0. Altså: Når D0 returnerer 3, ved vi at printeren løb tør for papir.

Linie 53: Afslutter rutinen.

Funktionen af denne printerrutine kan kort sammenfattes på denne måde:

IND: **A0 = adressen på teksten som skal printes.**

UD: **D0 = status**
0 = OK
1 = POWER OFF
2 = OFFLINE
3 = PAPER OUT

Programeksemplet MC1102 kommunikerer som nævnt direkte med printeren. Det kan både være en fordel og en ulempe. Fordelen er, at det går hurtigt, enkelt og problemfrit. Ulempen er, at hvis du laver et program som benytter kontrolkoder, vil programmet ikke virke med alle typer printere. Kontrolkoder benyttes for at styre f.eks **fed skrift**, understregning, skriftstørrelse, osv.

Som du sikkert ved, har AMIGAen et såkaldt PREFERENCES-program, hvor du kan angive hvilken type printer du har. Det har ingen indflydelse på dette programeksempel. Hvis du vil benytte AMIGAens printer-rutiner skal du bruge en anden metode. Det kan gøres ved at bruge "writefil"-eksemplet i BREV X. Filnavnet sættes til "prt:", og bufferen skal indeholde teksten som skal sendes. Hvis du har en printer, kan du prøve dette som et lille eksperiment.

I næste kapitel fortsætter vi med MASKINKODE X, hvor vi blandt andet skal se lidt nærmere på en programrutine, som sætter en PIXEL ud på en grafikskærm.

MASKINKODE X

Dette maskinkodekapitel indeholder en del programmeringstips.

Det første vi skal se på er en rutine, som sætter en PIXEL ud på en grafikskærm. X- og Y-positionen angives i henholdsvis D0 og i D1. Her kommer rutinen:

```
1  pixel:
2  mulu        #40,D1
3  move.w      D0,D2
4  lsr.w       #3,D0
5  not.b       D2
6  andi.w      #7,D2
7  add.w       D1,D0
8  lea.l       screen,A1
9  bset        D2,(A1,D0.w)
10 rts
```

Her kommer så forklaringen:

Linie 2: Multiplicerer D1 (Y-positionen) med 40. Vi antager at skærmbredden er 320 PIXELS (320/8 = 40).

Linie 3: Lægger værdien som ligger i D0 ind i D2.

Linie 4: Skifter de 3 nederste BITs i D0. Dette er det samme som at dividere med 8. Vi har nu fået BYTE-positionen for X-positionen.

Linie 5: Inverterer D2.

Linie 6: Isolerer Bit 0-2 i D2. Dette register vil nu indeholde BIT-positionen i BYTEn.

Linie 7: Lægger værdien, som ligger i D1, sammen med værdien i D0. Registret D0 indeholder nu BYTE-positionen på skærmen.

Linie 8: Lægger adressen på "screen" ind i A1.

Linie 9: Tænder BITen som D2 angiver i den BYTE som A1+D0 peger på. Det vil resultere i at en PIXEL bliver sat på skærmen.

Linie 10: Afslutter rutinen.

Du ved sikkert, at instruktionernes tidsforbrug er forskelligt. Der findes specielt to instruktioner som tager ekstra lang tid, nemlig multiplikation (MULU eller MULS) og division (DIVU eller DIVS). MULU bruger ca. 15-20 gange længere tid end f.eks "MOVE.L D0,D1", mens DIVU bruger op til 45 gange længere tid end "MOVE.L D0,D1".

I programeksemplet ovenfor benyttede vi en MULU. Når man skal programmere hurtige rutiner prøver man at undgå MULU, MULS, DIVU og DIVS-instruktioner. At udføre en multiplikation med 40 ved at rotere BITvis er faktisk muligt. Hvis vi ønsker at multiplicere med 32 ville sagen jo være meget enkel (LSL.W #5,D1). Tallet 40 er som bekendt ikke helt så "BINÆRT" som 32 (1,2,4,8,16,32,64,128 osv.). Vi vil vise hvordan dette gøres med et programeksempel (tallet som skal multipliceres ligger i D0):

```
1  move.l      D0,D1
2  lsl.w       #5,D0
3  lsl.w       #3,D1
4  add.w       D1,D0
5  rts
```

Linie 1: Lægger værdien som ligger i D0 ind i D1. Altså: Laver en kopi af D0 i D1.

Linie 2: Roterer D0 fem BITs til venstre. Altså: Multiplicerer med 32.

Linie 3: Roterer D1 tre BITs til venstre. Altså: Multiplicerer med 8.

Linie 4: Lægger værdien som ligger i D1 ind i D0.

Linie 5: Afslutter programmet.

Lad os vise nogle eksempler på hvordan det virker:

D0 = 1

1 * 32 = 32
1 * 8 = 8

32 + 8 = 40 (1 * 40 = 40)

Vi prøver et til...

D0 = 3

3 * 32 = 96
3 * 8 = 24
96 + 24 = 120 (3 * 40 = 120)

og et til...

D0 = 25

25 * 32 = 800
25 * 8 = 200

800 + 200 = 1000 (25 * 40 = 1000)

Det fungerer jo helt perfekt! Vi har på denne måde sløjfet en MULU. Hvis man skal være endnu mere avanceret, så bruger f.eks LSL.W #5,D0 længere tid end LSL.W #3,D0. Altså: Desto flere BITs man roterer, jo længere tid tager det at udføre instruktionen. Vi kan endda spare mere tid hvis vi omskriver det forrige programeksempel således:

```
1  lsl.W    #3,D0
2  move.l   D0,D1
3  lsl.w    #2,D0
4  add.w    D1,D0
5  rts
```

Linie 1: Roterer D0 tre gange til venstre. Altså: Multiplicerer med 8.

Linie 2: Lægger værdien som ligger i D0 ind i D1.

Linie 3: Roterer D0 to gange til venstre således at D0 nu er roteret 5 gange ialt.

Linie 4: Lægger værdien som ligger i D1 ind i D0.

Linie 5: Afslutter programmet.

I det første programeksempel udføres ialt 8 roteringer (5 + 3), mens der i dette programeksempel kun udføres 5 ialt (3 + 2). Altså: Rutinen er blevet endnu hurtigere. Vi viser nu en komplet "pixel"-rutine med den nye multiplikationsmetode:

```
1     pixel:
2     lsl.w    #3,D1
3     move.w   D1,D3
4     lsl.w    #2,D1
5     add.w    D3,D1
6     move.w   D0,D2
7     lsr.w    #3,D0
8     not.b    D2
9     andi.w   #7,D2
10    add.w    D1,D0
11    lea.l    screen,A1
12    bset     D2,(A1,D0.w)
13    rts
```

Denne rutine kan du bruge i dine egne programmer. Ydelsen på rutinen er ca. 50000 PIXELs i sekundet. At gøre rutinen hurtigere er vanskelig. Det eneste du kan gøre er at lægge programlinie 11 udenfor rutinen. Hvis rutinen udføres mange gange efter hinanden, er der jo ingen grund til at hente skærmadressen hver gang.

For en god ordens skyld vises opsætningen til rutinen:

IND: **D0 = X-position**
D1 = Y-position

UD: **Ingenting**

Det næste vi skal se lidt nærmere på, er hvordan man kan aflæse joystick-positioner. Da vi aflæste muspositionerne, benyttede vi registret $DFF00A. Dette register tilhører JOYSTICK-port 1. I og med at musen ofte er tilkoblet JOYSTICK-port 1 hele tiden, laver vi en rutine som aflæser JOYSTICK-port 2. Registret for JOYSTICK-port 2 er $DFF00C. Her kommer rutinen:

```
1     readjoy:
2     move.w   D1,-(A7)
3     move.w   $DFF00C,D0
4     move.w   D0,D1
5     andi.w   #3,D0
6     lsr.w    #6,D1
7     andi.w   #12,D1
7     add.w    D1,D0
9     move.w   (A7)+,D1
10    rts
```

Linie 1: Rutinen hedder "readjoy".

Linie 2: Lagrer D1 på STACKen. Læg mærke til at vi kun lagrer BITene 0-15 (WORD) fra D1. Vi behøver ikke at lagre registret som LONGWORD fordi vi kun benytter BITene 0-15 i register D1 i programrutinen. Dette gør at rutinen bliver lidt hurtigere og vi sparer 2 BYTEs (et WORD) i STACKen.

Linie 3: Lægger værdien som ligger på adresse $DFF00C ind i D0.

Linie 4: Lægger værdien som ligger i D0 ind i D1. Altså: Lægger en kopi af D0 ind i D1.

Linie 5: Udfører en logisk AND sådan at det kun er BIT 0 og 1 som står tilbage i D0 (#3 = %0000000000000011).

Linie 6: Roterer indholdet i D1 seks BIT til højre.

Linie 7: Udfører en logisk AND sådan at BIT 2 og 3 står tilbage (#12 = %0000000000001100).

Linie 8: Lægger værdien i D1 til i D0.

Linie 9: Henter den gamle værdi på D1 ud fra STACKen.

Linie 10: Afslutter rutinen.

Som du ser er det BIT 0, 1 8 og 9 vi skal samle på fra registret $DFF00C. Alle andre BITs behøves ikke. Dette program resulterer i at vi har fået en værdi i D0's BIT 0-3.

I figur 1 bagest i brevet ser du hvilke værdier de forskellige retninger på JOYSTICKet har. Du kan jo som et eksperiment prøve at lave et program som f.eks styrer en SPRITE (eller en BOB) på skærmen ved at bruge et JOYSTICK.

Kommentarer til BREV XI

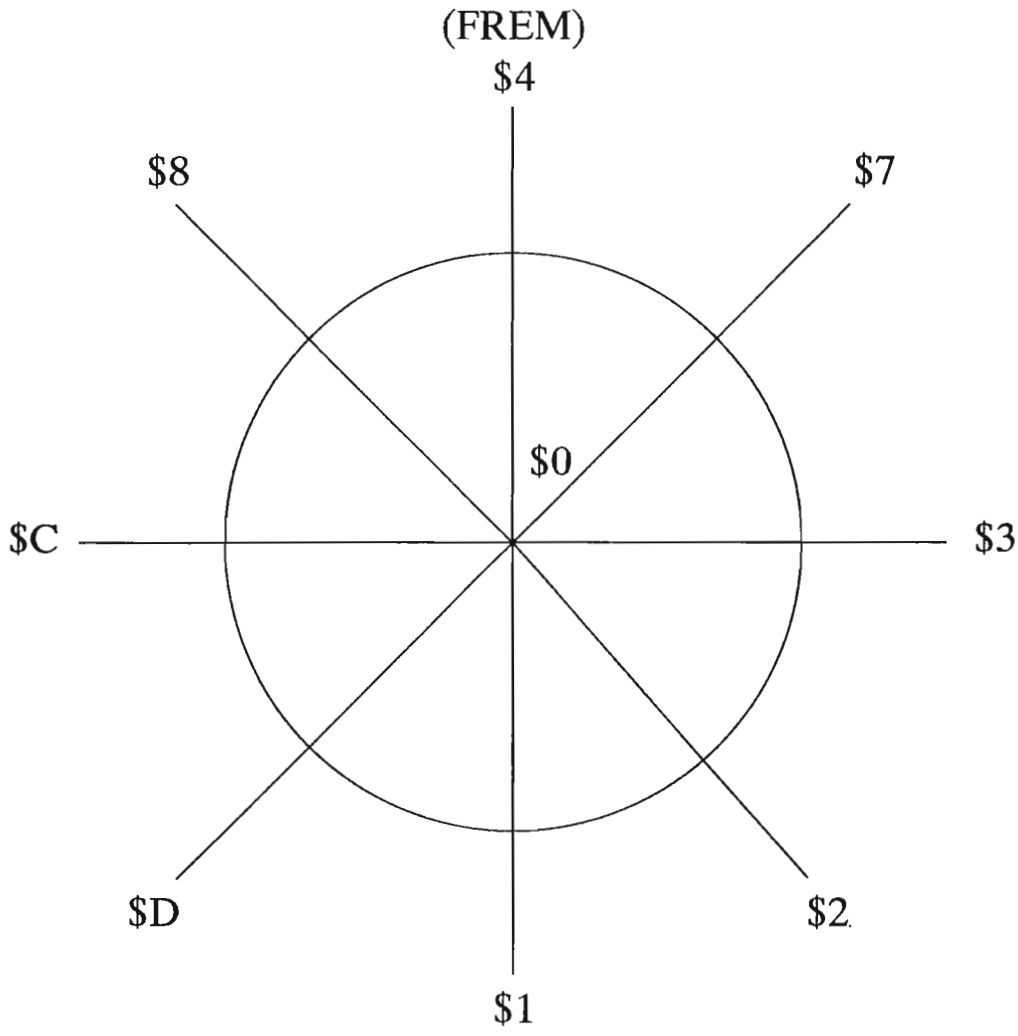
Dette er jo det næstsidste brev. Vi er kommet ganske langt omkring i gennemgangen af de forskellige maskinkode-instruktioner siden vi startede med gennemgangen af talsystemerne. Nogle instruktioner har vi behandlet grundigt, og nogle har vi kun lige set på. Det er altid vanskeligt at vælge hvilke emner man skal behandle, men vi håber, at du på dette tidspunkt har fået en så bred viden om maskinkodeprogrammering, at du selv kan arbejde videre med emnet.

I næste brev, som er det sidste, vil vi komme med forskellige tips og tricks, som er gode at have kendskab til, når du på egen hånd begiver dig ud i programmeringens store verden. **Sammen med BREV XII er der udgivet en diskette. Det vil være vanskeligt at få fuldt udbytte af dette brev uden at have den diskette ved hånden.** Vi anbefaler dig derfor at bestille den, så du kan få den sammen med det sidste brev.

Vi ses i brev 12.

Med venlig hilsen
DATASKOLEN

Carsten Nordenhof



Figur 1.