Programmering i maskinkode på *AMIGA*

A.Forness & N.A.Holten
Copyright 1989 ARCUS
Copyright 1989 DATASKOLEN

Hæfte 6

Indhold

DATASKOLEN
Postboks 62
Nordengen 18
2980 Kokkedal

Telefon 49 18 00 77

Postgiro 7 24 23 44

MASKINKODE

Følgende lille program ligger ikke på kursusdiskette # 1. Skriv det ind, assembler det og start det op ved at skrive "j". Bevæg derefter musen fra side til side, og se på POWER-lampen imedens.

```
1  move.w       #$4000,$dff09a
2
3  mainloop:
4  bset         #1,$bfe001
5  bsr wait1
6  bclr         #1,$bfe001
7  bsr wait2
8
9  btst         #6,$bfe001
10 bne          mainloop
11
12 move.w       #$c000,$dff09a
13 rts 14
15 wait1:
16 move.w       $dff00a,d0
17 and.l        #$ff,d0
18 waitloop1:
19 dbra         d0,waitloop1
20 rts 21
22 wait2:
23 move.w       $dff00a,d0
24 and.l        #$ff,d0
25 not.b        d0
26 waitloop2:
27 dbra         d0,waitloop2
28 rts
```

Her følger en forklaring til dette lille program:

Linie 1: Slukker alle INTERRUPTS.

Linie 4: Sætter BIT 1 i $BFE001 til "1". Dette slukker POWER-lampen.

Linie 5: Hopper til "wait1".

Linie 6: Sætter BIT 1 i $BFE001 til "0". Dette tænder POWER-lampen.

Linie 7: Hopper til "wait2".

Linie 9-10: Checker om musknappen er trykket ned. Hvis ikke, hop tilbage til "menuloop". Er den trykket ned, så udføres linierne 12 og 13.

Linie 12: Tænder alle INTERRUPTS igen.

Linie 13: Hopper tilbage til SEKA (afslut).

Linie 16: Henter musposition en og lægger den i D0.

Linie 17: Udfører en logisk AND med D0 således at vi atter kun får BIT 0-7 (BIT 8-31 sættes til "0"). Nu har vi luget de overflødige" BITs væk, således at kun musens x-position er tilbage.

Linie 18-19: Denne loop benyttes som en forsinkelse. Den går i loop så mange gange som det tal, som ligger i D0 repræsenterer (musposition en).

Linie 20: Hopper tilbage til programlinie 6.

Linie 23: Lægger musposition en ind i D0.

Linie 24: Sætter alle BIT undtagen BIT 0-7 til "0" (Husk at BIT 0-7 forbliver uberørte).

Linie 25: Inverterer (vender, dvs 0 bliver til 1 og omvendt) BIT 0-7 i D0. Det er kun BIT 0-7 som inverteres fordi vi bruger "b" i instruktionen (En BYTE er jo otte BIT).

Linie 26-27: Går i loop det antal gange, som tallet i D0 angiver.

Linie 28: Hopper tilbage til programlinie 9.

Eksemplet ovenfor fungerer således, at når du flytter musen fra side til side, kan POWER-lampen tændes og slukkes "trinløst". Hele hemmeligheden er at tænde og slukke POWER-lampen meget hurtigt.

Tænk efter før du læser videre om du kan finde ud af, hvorfor og hvordan det virker som det gør.

Fandt du ud af det? Sandsynligvis ikke. Det er ikke så let at komme på af sig selv. Men når du har læst forklaringen nedenfor, så forstår du det - helt sikkert!

Hvis vi lader lampen være tændt lige så længe som den er slukket, vil vi opleve det som om lampen lyser med halv styrke. Hvis vi så lader lampen være slukket i 90% af tiden, og tændt i 10% af tiden, oplever vi det som om lampen lyser meget svagt (10% af fuld styrke).

Denne regulering udføres med rutinerne "wait1" og "wait2".

LOGISK MATEMATIK

Nu skal vi se på et område indenfor programmering som mange synes er vanskeligt at forstå:

Regneoperationer, som udføres ved hjælp af de logiske operatorer:

AND, OR og NOT.

Der er dog ingen grund til at blive forskrækket. Det hele er meget logisk. Men - som med alt andet her i livet - så må man øve sig før man bliver mester.

Et udtryk kan f.eks. se således ud:

D=(A AND B) OR ((NOT A) AND C)

Akkurat som i almindelig matematik begynder man med at løse de inderste parenteser først. Det betyder at i udtrykket (NOT A), skal A inverteres. Hvis A altså var 1010, så bliver det efter at (NOT A) er udført 0101. Dette skriver man i logisk matematik med et A med en streg over for at markere invertering – sådan:

$\overline{A}$

Det nye reducerede udtryk vil se således ud:

D=(A AND B) OR ($\overline{A}$ AND C)

I logisk matematik skrives en AND som en multiplikation. Det betyder at parentesen (A AND B) kan skrives således:

A*B

I almindelig matematik udelader man ofte multiplikations-tegnet.

Det er der, men det skrives ikke. Matematikere er dovne folk.!

Nuvel, det gør at udtrykket (A AND B) - som vi kunne skrive som A*B - kan forenkles yderligere til AB.

Hele udtrykket kan nu skrives således:

D=(AB) OR ($\overline{A}$ AND C)

Vi håber du er med så langt. Lad os gå igang med den sidste parentes. Som nævnt skrives end AND i logisk matematik som en multiplikation. Det resulterer i at udtrykket...

$(\overline{A}$ AND C) kan skrives: $(\overline{A}C)$
udtrykket er nu blevet reduceret til:

$D=(AB)$ OR $(\overline{A}C)$
Så langt burde alt være klart som blæk. Men vi har en logisk operation tilbage, nemlig en OR. I logisk matematik skrives den som en addition, altså med et plustegn (+). Vi kan nu reducere udtrykket endnu en gang:

$D=(AB)+(\overline{A}C)$
Parenteserne kan nu fjernes helt og udtrykket ser i sin enkleste form således ud (næsten ikke til at kende igen, vel):

$D=AB+\overline{A}C$

Når man skal læse det færdige udtryk, så siger man:

D er lig med A gange B plus A-invers gange C.

Man bruger samme regnerækkefølge som i almindelig matematik: En multiplikation (og division) udføres før en eventuel addition (eller subtraktion), så udregningen udføres sådan:

A og B skal ANDes med hinanden
A skal inverteres for så at ANDes med C
De to resultater skal ORes med hinanden
Resultatet skal havne i D.

OBS: I logisk matematik bruges ikke division og subtraktion.

I dette kursus behøver du ikke vide mere om logisk matematik end det vi her har gennemgået. Men du kan senere have stor gavn af at læse speciallitteratur om logisk matematik. Det er uvurderlig og nødvendig viden, hvis man vil være en god programmør.

BLITTER 1

Ordet BLITTER er en forkortelse for BLOCK IMAGE TRANSFERRER (dansk: Blok-grafik-flytter). BLITTERen i AMIGA kan bruges til mange ting. Lad os begynde med at nævne at den bl.a. kan:

- kopiere store mængder data fra et sted i hukommelsen til et andet.

- flytte data BIT-vis i hukommelsen (bruges bl.a. i SCROLLing).

- bevæge figurer på en grafik-skærm.

- flytte figurer rundt på skærmem, uden at ødelægge eventuel bagved liggende grafik (baggrund).

- tegne linier (streger) på skærmen.

Altså, BLITTERen kan hente data fra hukommelsen og f.eks. udføre en logisk operation for så at lægge dataene tilbage igen på samme sted (eller et andet sted) i hukommelsen.

BLITTERen kan benytte sig af op til 4 DMA-kanaler på en gang. Disse DMA-kanaler kaldes A, B, C og D. Kanalerne A, B og C kan kun bruges til at hente data fra hukommelsen, mens kanal D kun bruges til at skrive data til hukommelsen. DMA-kanalerne A, B og C kaldes ofte for SOURCE-kanaler (oversat: kilde-kanaler) fordi de kun kan hente data. Kanal D kaldes for DESTINATION-kanal fordi den kun kan skrive data til hukommelsen.

Forestil dig at vi skal kopiere data som ligger i hukommelsesområdet $10000 - $100FF (tilsammen 256 BYTEs) ind på adresse $20000 - $200FF.

Vi kan da bruge operationen D=A (D-kanalen er lig A-kanalen) Derefter sætter vi A-kanalen til at pege på adresse $10000, og kanalen på $20000. Længden sættes til $100, og til sidst startes BLITTERen. Den vil så arbejde på egen hånd, således at hovedprocessoren kan fortsætte med sine opgaver.

Vi kan også udføre logiske operationer med BLITTERen. De tre funktioner er: AND, OR, NOT. Disse kan kombineres på forskellige måder. For eksempel som i det foregående kapitel:

$D=AB+\overline{A}C$ (reduceret fra D=(A AND B)OR((NOT A)AND C))

Vi kommer tilbage til dette på et senere tidspunkt når vi får brug for det i praksis. Lad os se på de registre BLITTERen bruger.

Tabel over adresse-pointers til A, B, C og D:

Navn		BITS	Adresse
BLTAPTH	(A)	16-31	$DFF050
BLTAPTL	(A)	0-15	$DFF052
BLTBPTH	(B)	16-31	$DFF04C
BLTBPTL	(B)	0-15	$DFF04E
BLTCPTH	(C)	16-31	$DFF048
BLTCPTL	(C)	0-15	$DFF04A
BLTDPTH	(D)	16-31	$DFF054
BLTDPTL	(D)	0-15	$DFF056

Modulo-registrene for A, B, C og D:

Navn	Adresse
BLTAMOD	$DFF064
BLTBMOD	$DFF062
BLTCMOD	$DFF060
BLTDMOD	$DFF066

BLITTER kontrol-register 0 og dets opsætning:

BLTCONO - $DFF040

BIT	Funktion
15	ASH3
14	ASH2
13	ASH1
12	ASH0
11	USE A
10	USE B
9	USE C
8	USE D
7	LF7
6	LF6
5	LF5
4	LF4
3	LF3
2	LF2
1	LF1
0	LF0

ASHO - ASH3 = Roteringsværdi for A-kanalen (A SHIFT). Kan indeholde en værdi fra 0 til 15.

USE A - USE D = Her vælges hvilke DMA-Kanaler som skal bruges.

LF0 - LF7 = Bruges til at sætte den logiske operation som BLITTERen skal benytte (LF = LOGIC FUNKCTION).

BLITTER kontrol-register 1 og dets opsætning:

BIT	Funktion
15	BSH3
14	BSH2
13	BSH1
12	BSH0
11	-
10	-
9	-
8	-
7	-
6	-
5	-
4	EFE
3	IFE
2	FCI
1	DESC
0	LINE

BSH0 - BSH3 = Roteringsværdi for B-Kanalen (B SHIFT). Kan indeholde en værdi fra 0 til 15.

EFE, IFE, FCI = Bruges i forbindelse med udfyldning af figurer. Vi går ikke nærmere ind på det nu - men vær bare rolig - vi kommer tilbage til det!

DESC = Bruges til at sætte BLITTERen i "bakgear" (kaldes på datasprog: DESCENDING, udtales dis9e9nding og betyder nedstigende).

Denne metode kommer vi tilbage til i BREV VII.

BLITTER størrelse (BLITTER SIZE) register og dets opsætning:

BLTSIZE - $DFF058

BIT	Funktion
15	H9
14	H8
13	H7
12	H6
11	H5
10	H4
9	H3
8	H2
7	H1
6	H0
5	W5
4	W4
3	W3
2	W2
1	W1
0	W0

H0 - H9 = Højden (HEIGHT) på "BLITen". Kan indeholde en værdi fra 0 til 1023.

W0 - W5 = Bredden (WIDTH) på "BLITen". Kan være en værdi fra 0 til 63.

Læg mærke til at alle registrene, som vi har vist her, er registre, som du kun kan skrive til (WRITE ONLY).

BEGREBET MODULO
Vi fortsætter nu med at forklare hvordan MODULO-registrene (BLTxMOD) fungerer.

Studer FIGUR 1 bag i dette brev. Figuren viser et udsnit af en grafikskærm. En rude i figuren svarer til 16 PIXELs i bredden, og 1 linie i højden. Tallene i ruderne er skærm-adresser. Som du ser, består en linie af 40 BYTEs, hvilket er en bredde på 320 PIXELs.

På skærmudsnittet er der trukket en firkant op. Denne firkant skal fyldes med de data, som ligger i bufferen. Dette gøres med en "BLIT".

Som du kan se er den første adresse (øverste venstre hjørne) i firkanten 126, mens den første adresse i bufferen er 0. Forestil dig at skærmhukommelsen ligger på adresse $10000, og at bufferen ligger på adresse $20000. Da bliver den første adresse inde i firkanten $10000 + 126 = $1007E ($7E=126), og den første adresse i bufferen bliver $20000 + 0 = $20000.

Den logiske operation som BLITTERen skal udføre er D=A.

Størrelsen af en "BLIT" specificeres i højde og bredde. Højden angives i antal linier, mens bredden angives i blokke på 16 PIXELs. Bredden på "BLITen" i dette tilfælde bliver 4 (4 * 16 = 64 PIXELs), og højden bliver 3 (3 linier).

Adresserne inde i firkanten ligger ikke i rækkefølge sådan som i bufferen. Lad os se hvordan dataene skal flyttes således at det bliver rigtigt.

BUFFER		SKÆRM
0	->	126
2	->	128
4	->	130
6	->	132
8	->	166
10	->	168
12	->	170
14	->	172
16	->	206
18	->	208
20	->	210
22	->	212

Som du ser er der et "hop" i rækkefølgen på skærmadresse 132-166 og 172-206. Dette "hop" sættes i MODULO-registrene. Den værdi som skal bruges for at få korrekt "hoplængde", kaldes for en MODULO.
MODULO bliver i vores tilfælde 0 for A-kanalen (som peger på bufferen), fordi alle data i bufferen ligger lige efter hinanden.

MODULO for D-kanalen (som peger på skærmhukommelsen) udregnes således:

Bredden på "BLITen" er 4 WORDS (eller 4 ruder i figuren). Et WORD indeholder 16 BITs, altså 16 PIXELs. Vi ved også at skærmbredden (linielængden) er 40 BYTEs, Vi regner så MODULO ud således:

40 -(64/8) = 40 - 8 = 32.

Vi har nu følgende:

- Startadressen for begge DMA-kanaler (A = $20000 og D = $1007E).

- MODULO for begge kanaler (A = 0 og D = 32).

- Den logiske operation BLITTERen skal udføre (D=A), direkte kopiering fra A til D9.

- Størrelsen på "BLITen" (højden = 3 og bredden = 4)

Den komplette opsætning i maskinkode vil se således ud:

```
1  MOVE.L #$20000,$DFF050
2  MOVE.L #$1007E,$DFF054
3  MOVE.W #0,$DFF064
4  MOVE.W #32,$DFF066
5  MOVE.W #$09F0,$DFF040
6  MOVE.W #$000, $DFF042
7  MOVE.L #$FFFFFFFF,$DFF044
8  MOVE.W #$00C4,$DFF058
```

Linie 1: Lægger værdien af #$20000 (adressen på bufferen) ind i A-kanal-pointeren

Linie 2: Lægger værdien #$1007E "(adressen på skærm-hukommelsen) ind i D-kanal-pointeren.

Linie 3: Sætter MODULO til 0 (nul) for A-kanalen.

Linie 4: Sætter MODULO til 32 for D-kanalen.

Linie 5: Den er sikkert lidt indviklet: Se i tabellen til registret (BLTON0). Der vil du se at dette register blandt andet sætter den logiske funktion som BLITTERen skal udføre. Derudover angives der hvilke DMA-kanaler, som skal bruges (i dette tilfælde A og D). Når det gælder den logiske operation, så har vi lavet en tabel over de mest benyttede operationer bagest i brevet. Værdien for operationen A=D, som bliver brugt i dette tilfælde, er $F0.

Linie 6: Dette register sættes som regel til 0. Vi vil komme tilbage til dette register senere når vi får brug for det. Se også i tabellen over registeret (BLTC0N1).

Linie 7: Dette register sættes til $FFFFFFFF. Vi vil ikke forklare hvad dette register gør i denne omgang, men hvis du vil eksperimentere med BLITning selv, kan du trygt altid sætte dette register til $FFFFFFFF (for den nysgerrige hedder dette register BLITTER A MASK).

Linie 8: Her kommer det register som bestemmer, hvor stor BLITen skal være (højde og bredde). Den udregnes ved hjælp af en enkel formel:

(højde * 64) + bredde

Altså: 3 (linier) * 64 + 4 (WORDs, som svarer til 64 PIXELS) = 196 (DECIMALT) = $00C4 (HEXADECIMALT)

Lad os nu se på eksempel MC0601. På grund af eksemplets længde har vi ikke trykt det i brevet, (du finder det på kursusdiskette #1).

Først en enkel forklaring af programmet:

Linie 1-28: Slukker INTERRUPTS og diskettestation. Sætter en skærm op med en BITPLANE på 320 * 256 PIXELs i LORES, og starter COPPERen.

Linie 30-42: Dette er hovedprogrammet (hovedLOOPen).

Linie 44-53: Henter den gamle skærm tilbage og afslutter programmet.

Linie 55-67: Dette er en underrutine (SUBRUTINE), som ved hjælp af BLITTERen renser skærmen (lægger 0 i alle skærmadresserne).

Linie 72 -103: "Denne underrutine "BLITTER" figuren ind på skærmen. Dette foregår ved hjælp af BLITTERen.

Linie 105-120 Her er vores COPPER-liste.

Linie 122-126: Den første "BLK"-kommando sætter 10240 BYTEs af til skærmhukommelsen. Den anden sætter 640 BYTEs af til figurdata (som også er på diskette #1).

Her følger en grundigere forklaring på de nye og ukendte dele i programmet:

Linie 1-28: Dette burde være kendt stof for alle nu.

Linie 31-35: Denne rutine venter til elektronstrålen har nået skærmlinie 300.

Linie 37: Her hoppes der til rutinen "clear".

Linie 39: Her hoppes der til rutinen "blitin".

Linie 41-41: Checker om venstre mus-knap er trykket ned. Hvis ikke - så hop tilbage til "mainloop".

Linie 44-53: Her afsluttes programmet.

Linie 55: Denne rutine renser skærmen (lægger værdien 0 i alle skærmadresserne). Vi bruger kun D-kanalen i denne BLIT (altså, ingen SOURCE-kanaler). Dette resulterer i at der kun bliver lagt nuller i hukommelsen.

Linie 56: Læg adressen på skærmhukommelsen (screen)ind i A1.

Linie 58-60: Denne rutine venter med at gå videre til BLITTERen er færdig med sit forrige job (eller "BLIT").

Linie 62: Læg adressen på skærmhukommelsen ind i D-kanal-pointeren.

Linie 63: Sæt MODULO for D-kanalen til 0.

Linie 64: Vælg kun D-kanalen og ingen logisk operation.

Linie 65: Læg værdien 0 ind i BLTCONl.

Linie 66: Angiv størrelse på BLITen. Højden sættes til 256, og bredden til 20 WORDs (20 * 16 = 320). Når du skriver til dette register, starter BLITTERen (DMA-kanalerne) af sig selv. Derfor skal du sætte størrelsen på BLITen ($DFF058) til sidst. Du skal altså ikke sætte nogen BITs i DMACON-registret ($DFF096) for at BLITTERen skal starte.

Linie 67: Hop tilbage til hovedrutinen.

Linie 69-70: Her har vi deklareret et LONGWORD der skal indeholde positionen til figuren på skærmen.

Linie 72: Denne rutine BLITer figuren ind på skærmen.

Linie 73: Hent adressen på "pos" ind i Al.

Linie 74: Læg indholdet fra adressen, som Al peger på, ind i D1.

Linie 75: Læg 1 til den værdi, som adressen i A1 peger på. Det vil sige at 1 lægges til i vores LONGWORD, og ikke i register D1.

Linie 77: Sammenlign D1 med 216.

Linie 78: Hvis D1 er ulig 216, hop til "notbottom". Altså, check om figuren er nået til bunden af skærmen.

Linie 80: Sæt alle BITs i D1 til 0.

Linie 81: Sæt alle BITs i adressen, som A1 peger på, til 0.

Linie 84: Læg adressen på "screen" ind i A1.

Linie 85: Multiplicer D1 med 40. Det gøres fordi en skærmlinie indeholder 40 BYTEs. Denne instruktion er vi ikke stødt på før, men vi synes den er selvforklarende. Vi vil dog tage bl.a. denne instruktion op i forbindelse med såkaldte signerede værdier i et senere brev. For den nysgerrige har MC-68000 også en instruktion, som kan udføre en division. Den kan f.eks. se således ud: DIVU #5,D1

Linie 86: Adder værdien som ligger i D1 til værdien i A1.

Linie 87: Adder 12 til værdien i A1. Dette gør at figuren centreres horisontalt på skærmen.

Linie 89: Læg adressen på figuren ind i A2.

Linie 91-93: Vent med at gå videre til et eventuel andet BLITTER-job er færdigt.

Linie 95: Læg værdien (skærmadressen), som ligger i A1, ind i pointerne for D-kanalen.

Linie 96: Læg værdien (figuradressen), som ligger i A2, ind i pointerne for A-kanalen.

Linie 97: Sæt D-kanalens MODULO til 24.

Linie 98: Sæt A-kanalens MODULO til 0.

Linie 99: Sæt A MASK til SFFFFFFFF. Brugen af dette register vil vi ikke forklare lige nu (men vi kommer selvfølgelig tilbage til det senere).

Linie 100: Vælg kanalerne A og D, og sæt den logiske operation til: D=A.

Linie 101: Læg værdien 0 ind i BLTC0N1 ($DFF042).

Linie 102: Sæt BLITens størrelse. Højde = 40 (linier) og bredde = 8 WORDs (8*16=128 PIXELs).

Linie 103: Hop tilbage til hovedrutinen.

Linie 105-120: Her deklareres COPPER-listen

Linie 122-126: Her deklareres skærmbuffer og figurbuffer.

For at kunne køre dette program skal du først assemble det og derefter hente filen "OBJECT ind med kommandoen "ri". Filen ligger i samme katalog (DIRECTORY) som selve kilde-koden (SOURCE CODE).

Husk at BLITTER ligesom COPPER, SPRITE etc. kun kan arbejde i CHIP-RAM.

BLITTER II

I dette kapitel forklares hvordan BLITTERen kan flytte en figur på skærmen uden at berøre (eller ødelægge) de grafikdata, som i forvejen findes på skærmen.

Studer FIGUR 2 bag i brevet. Objektet er figuren, som skal BLITes ind på skærmen. MASKen er en figur som dækker over objektet.

Altså: Den har samme form og størrelse som objektet, men har kun en BITPLANE.

Det virker sådan:

- Først læses musens position.

- Derefter kopieres den del (bestemt af muspositionen) af skærmhukommelsen, som figuren skal lægges ind i, over i baggrundsbufferen.

- Nu BLITes figuren ind. Hvis vi kopierede hukommelsen direkte fra objektbufferen ind i skærmhukommelsen (BITMAPen), ville vi få en sort ramme rundt om den runde figur. Dette undgås således:

Den logiske operation for denne BLIT er $D=AB+\overline{A}C$ eller skrevet på en anden måde:

D = (mask * objekt) + (mask * baggrund)

Stregen over MASK (A) betyder at de BITs som indlæses i BLITTERen fra MASKbufferen vil blive inverteret. (Husker du? : et nul (0) bliver en (1), og en (1) bliver nul (0).) Det resulterer i at:

De steder MASKen indeholder nuller (udenfor cirklen), vil dataene som ligger i baggrundsbufferen blive kopieret (BLITet) ind på skærmen, og der hvor MASKen indeholder enere (inde i cirklen) vil dataene, som ligger i objektbufferen (figuren) blive kopieret ind på skærmen.

- Til sidst kopieres baggrunden ind på skærmen således at den bliver som den var fra begyndelsen.

Hele denne rutine gentages kontinuerligt således at det vil se "flydende" ud når figuren flyttes rundt på skærmen.

Vi fortsætter nu med en enkel forklaring af programeksemplet MC0602:

Linie 1-17: Sætter en skærm op med opløsningen 320 * 256 PIXELs og med 5 BITPLANEs (32 farver).

Linie 19-25: Her lægges farverne (som ligger i begyndelsen af skærmbufferen) ind i farveregistrene.

Linie 27-40: Her lægges adresserne på de 5 BITPLANEs ind i COPPER-listen.

Linie 42-44: Læg adressen på COPPER-listen ind i COPPER-pointeren.

Linie 46: Start BITPLANE- og COPPER-DMAerne. Som du ser har vi sat en ekstra bit i dette register. Dette er BIT 10 i DMACON-registret. Hvis man sætter denne BIT vil BLITTERen i enkelte tilfælde blive noget hurtigere. Hvorfor? Det forklarer vi i et senere brev.

Linie 52-65: Dette er hovedrutinen, som skulle være til at forstå.

Linie 67-77: Henter den oprindelige COPPER-liste tilbage og afslutter programmet.

Linie 79-121: Denne rutine BLITer figuren ind.

Linie 123-171: Denne rutine drejer figuren og MASKen (BIT-vis) således at figuren også kan flyttes PIXEL-vis i horisontalretningen.

Linie 173-185: Her aflæses musens position.

Linie 187-222: Denne rutine kopierer skærmens baggrund (skærmhukommelsen) ind i baggrundsbufferen ("BACKBUFFER").

Linie 224-259: Denne rutine tilbage-kopierer baggrunden fra baggrundsbufferen til skærmen.

Linie 261-280: Her er vores COPPER-liste deklareret.

Linie 282-298: Her er alle de nødvendige buffere deklareret.

Linie 300-303: Her har vi deklareret to LONGWORDs til at holde orden på muspositionerne (x- og y-koordinaterne).

Her følger en kort forklaring på alle SUBRUTINER (under-rutiner):

Linie 80: Læg adressen på "maskbuffer" i A1.

Linie 81: Læg adressen på "backbuffer" i A3.

Linie 82: Læg adressen på "figbuffer" i A2.

Linie 83: Læg skærmhukommelsens ("screen") adresse i A4.

Linie 84: Læg 64 til værdien i A4. Dette gøres for at springe over de farvedata, som ligger i begyndelsen af skærmbufferen.

Linie 86: Hent adressen på "mousex" og læg den i A0.

Linie 87: Læg værdien, som findes på den adresse A0 peger på, ind i D0.

Linie 88-89: Udfør det samme for musens y-positionen.

Linie 91: Skift indholdet i D0 fire BITs til højre. Denne instruktion udfører egentlig en division med 16 (2^4 eller 2*2*2*2).

Linie 92: Skift indholdet i D0 en (1) BIT til venstre. Denne instruktion udfører egentlig en multiplikation med 2 (2^1).

Linie 93: Multiplicer Dl med 40. Dette gør at y-positionen bliver rigtig i forhold til skærmen (1 linie på skærmen er jo 40 BYTEs).

Linie 94: Adder indholdet i D0 til værdien i A4.

Linie 95: Adder indholdet i D1 til værdien i A4.

Linie 97: Læg værdien 4 ind i D7.

Linie 99-101: Vent til BLITTERen er ledig.

Linie 103: Læg værdien som ligger i A4 ind i D-pointeren.

Linie 104: Læg værdien som ligger i A1 ind i A-pointeren.

Linie 105: Læg værdien som ligger i A2 ind i B-pointeren.

Linie 106: Læg værdien som ligger i A3 ind i C-pointeren.

Linie 107: Sæt MODULO for D-kanalen til 32 (figuren er 64 PIXELs bred: 40-(64/8)=32).

Linie 108-110: Sæt MODULO for A, B, og C-kanalerne til 0.

Linie 111: Læg værdien SFFFFFFFF ind i A MASK-registret.

Linie 112: Her sættes operationen $D=AB+\overline{A}C$, og alle kanalerne vælges. Linie 113: Læg værdien 0 ind i BLTCON1.

Linie 114: Her sættes BLITens størrelse. Højde: 45 (linier eller PIXELs) og bredde: 4 WORDs (64 PIXELs).

Linie 116: Læg 360 til i A2, således at A2 peger på næste bitplane i figuren (45*(64/8)=360).

Linie 117: Læg 360 til i A3 (baggrund).

Linie 118: Læg 10240 til i A4 (skærm). MASK-bufferen bliver den samme fordi den kun indeholder en BITPLANE.

Linie 120: Gentag denne BLIT 5 gange (en gang for hvert af de 5 BITPLANEs).

Linie 121: Hop tilbage til hovedrutinen.

Linie 124: Læg adressen på "fig" ind i A1.

Linie 125: Læg adressen på "figbuffer" ind i A2.

Linie 127-128: Hent musens x-position ind i D1.

Linie 130: Sæt alle BITs (undtagen BIT 0-3) i D1 til 0.

Linie 131-132: Indholdet i Dl skiftes ialt 12 BITs til venstre. Man skal skifte to gange fordi MC-68000 (med denne metode) kun tillader 8 skift ad gangen.

Linie 133: Læg værdien $09F0 til i D1. D1 vil nu indeholde $x9F0, hvor "x" er det antal BIT data i BLITTERen skal skiftes. I brev VII kommer vi ind på, hvordan BLITTERen udfører sådanne skift. Denne BLIT henter alle data i "fig", skifter dem "x" gange til højre, og lægger resultatet i "figbuffer".

Den logiske operation er D=A, og D og A-kanalerne er valgt.

Linie 135: Læg værdien 4 ind i D7 (bruges som tæller).

Linie 137-139: Vent til BLITTERen er ledig.

Linie 141: Læg værdien som ligger i A2 ind i D-pointeren.

Linie 142: Læg værdien som ligger i Al ind i A-pointeren.

Linie 143-144: Sæt MODULO for A og D-kanalen til 0.

Linie 145: Sæt A MASK-registret til $FFFFFFFF.

Linie 146: Læg værdien som ligger i D1 ind i BLTCON0.

Linie 147: Læg værdien 0 ind i BLTCON1.

Linie 148: Størrelsen på denne BLIT er også 45 linier * 4 WORDs (8 BYTEs eller 64 PIXELs).

Linie 150-151: Læg 360 til i både A1 og A2 så der peges på næste BITPLANE.

Linie 153: Dette repeteres for alle 5 BITPLANEs.

Linie 155-169: Det foregår på samme måde for MASKen også. Den eneste forskel er at MASKen kun har en BITPLANE.

Linie 171: Hop tilbage til hovedrutinen.

Linie 174-185: Denne rutine aflæser musens position og lægger værdierne i "mousex" og "mousey". Vi forklarer ikke dette register ($DFF00A) nu. Vi kommer tilbage til musrutiner i BREV XI.

Linie 188: Læg adressen på "screen" ind i A1.

Linie 189: Læg 64 til værdien i A1. Det får A1 til at springe over de farvedata, som ligger i begyndelsen af skærmbufferen.

Linie 190: Læg adressen på "backbuffer" ind i A2.

Linie 192-195: Hent muspositionerne (x- og y-koordinaterne) og læg dem i henholdsvis D0 og D1.

Linie 197-201: Find første BLIT-position på skærmen.

Linie 203: Læg værdien 4 ind i D7.

Linie 205-207: Vent til BLITTERen er ledig.

Linie 209: Læg værdien i A2 ind i D-pointeren.

Linie 210: Læg værdien i A1 ind i A-pointeren.

Linie 211: Sæt MODULO for D-kanalen til 0.

Linie 212: MODULO for A-kanalen sættes til 32.

Linie 213: A MASK-registret sættes til SFFFFFFFF.

Linie 214: Sæt den logiske operation D=A, og vælg A- og D-kanalen.

Linie 215: Læg værdien 0 ind i BLTCON1.

Linie 216: Sæt størrelsen til 45 linier * 4 WORDs.

Linie 218-219: Adder 10240 til værdien i A1, og adder 360 til værdien i A2.

Linie 221: Denne BLIT repeteres 5 gange.

Linie 222: Hop tilbage til hovedrutinen.

Linie 224-259: Denne rutine er næsten magen til "storeback"-rutinen. Forskellen er kun, at dataene BLITes i modsat retning (fra baggrundsbufferen til skærmhukommelsen).

For at kunne køre dette program skal du først assemble det, og derefter skal du indlæse filerne SCREEN, FIG og MASK (med kommandoen "ri"). De ligger i samme DIRECTORY som SOURCE-filen).

Vi håber at du har forstået, hvordan BLITTER virker og opfører sig. Vi vil ikke lægge skjul på at BLITTER er noget af det mest komplicerede i AMIGAen, men den er mulig at lære ved at eksperimentere selv. Så selv om du måske synes dette kapitel er helt uforståeligt, så giv ikke op. Kast dig over det en gang til!

TABEL OVER BLITTERENS MEST BRUGTELOGISKE FUNKTIONER

LF	Værdi	LF	Værdi
D=A	$F0	D=AB	$C0
D=$\overline{A}$	$0F	D=A$\overline{B}$	$30
D=B	$CC	D=$\overline{A}$B	$0C
D=$\overline{B}$	$33	D=$\overline{AB}$	$03
D=C	$AA	D=$\overline{B}$C	$88
D=$\overline{C}$	$55	D-B$\overline{C}$	$44
D=AC	$A0	D=$\overline{BC}$	$22
D=A$\overline{C}$	$50	D=BC	$11
D=$\overline{A}$C	$0A	D=A+$\overline{B}$	$F3
D=$\overline{AC}$	$05	D=$\overline{A}$+$\overline{B}$	$3F
D=A+B	$FC	D=A+$\overline{C}$	$F5
D=$\overline{A}$+B	$CF	D=$\overline{A}$+$\overline{C}$	$F5
D=A+C	$FA	D=B+$\overline{C}$	$DD
D=$\overline{A}$+C	$AF	D=$\overline{B}$+$\overline{C}$	$77
D=B+C	$EE	D=AB+A$\overline{C}$	$CA
D=$\overline{B}$+C	$BB	D=$\overline{A}$B+AC	$AC

LØSNINGER TIL OPGAVER I BREV V

Opgave 0501: Position x bliver 363, position y bliver 238 og højden bliver 35, Alle tal er angivet DECIMALT.

Opgave 0502: En SPRITE kan være 16 PIXELs bred.

Opgave 0503: I AMIGA er der 8 SPRITEs.

Opgave 0504: D0 vil se således ud:
10110010110100110101101110001001O

Opgave 0505: D0 vil få sin værdi fra adresse $000554

Opgave 0506: STACKen i AMIGA bliver bl.a. brugt til at:

- Lagre data midlertidigt
- Lagre returnerings-adresser (for BSR og RTS) til processoren.

OPGAVER TIL BREV VI

Opgave 0601: Prøv at oversæt MODULO til et godt dansk ord.

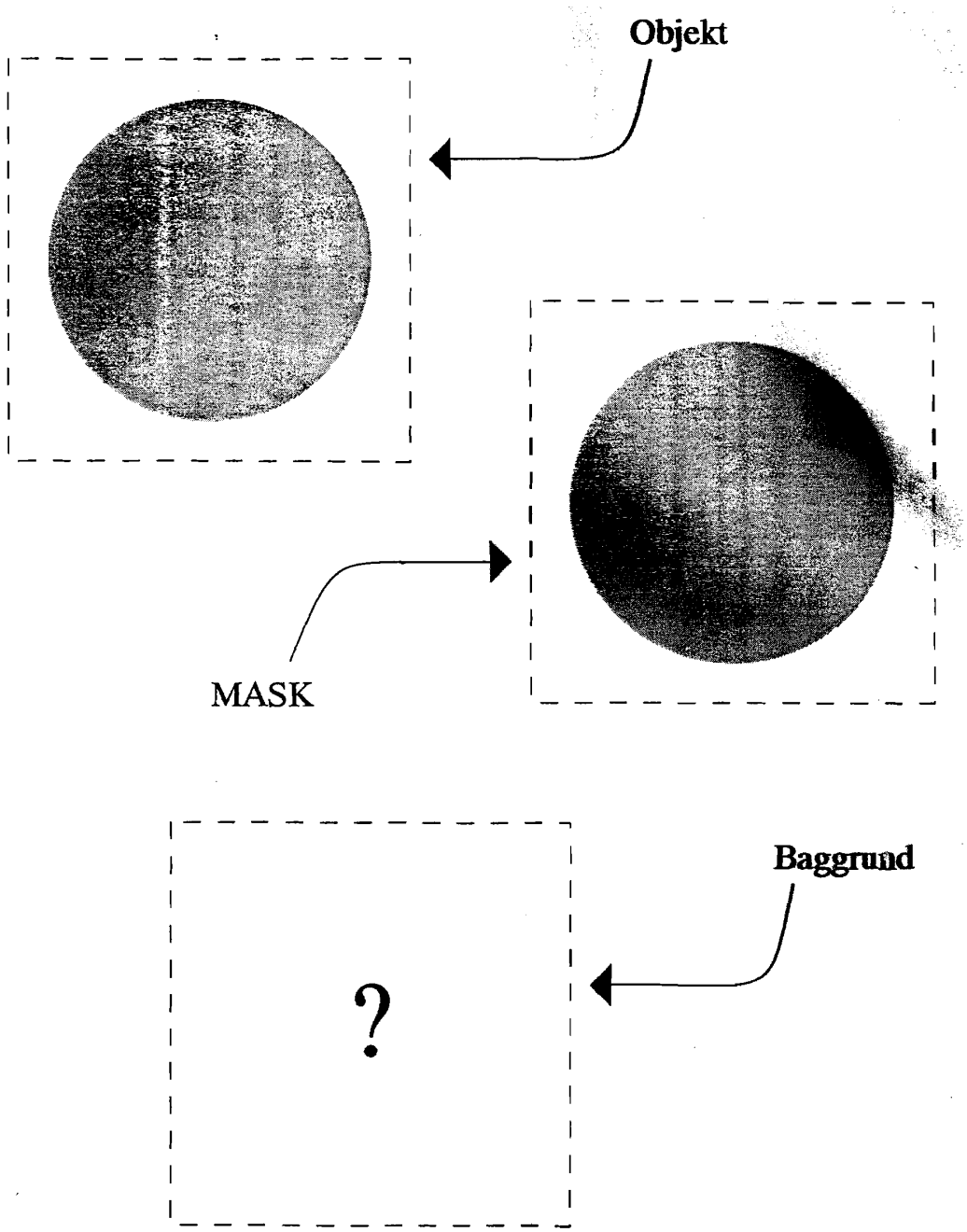
Opgave 0602: Hvor høj og hvor bred bliver en BLIT med denne BLITSIZE: $1A69

Opgave 0603: Hvad sker der med A i dette tilfælde: A

Opgave 0604: Hvor mange PIXELs horisontalt er det mindste som kan defineres i BLTSIZE (undtaget 0).

0	2	4	6	8	10	12	14	16	
40	42	44	46	48	50	52	54	56	
80	82	84	86	88	90	92	94	96	
120	122	124	126	128	130	132	134	136	
160	162	164	166	168	170	172	174	176	
200	202	204	206	208	210	212	214	216	
240	242	244	246	248	250	252	254	256	
280	282	284	286	288	290	292	294	296	
320	322	324	326	328	330	332	334	336	
360	362	364	366	368	370	372	374	376	
400	402	404	406	408					
440	442	444							

Figur 1.



Figur 2.