Programmering i maskinkode på *AMIGA*

A.Forness & N.A.Holten

Hæfte 2

Indhold

DATASKOLEN
Postboks 62
Nordengen 18
2980 Kokkedal

Telefon 42 24 96 57

Postgiro 7 24 23 44

DMA-KANALER

AMIGAen er opbygget af en række DMA-kanaler. "DMA" står for "DIRECT MEMORY ACCES". Disse kanaler håndteres af de tre CO-PROCESSORER: AGNUS, PAULA og DENISE. CO-PROCESSORERNE skriver til (og læser fra) hukommelsen uden hjælp fra hovedprocessoren MC-68000. Denne bruges kun til at opsætte og starte DMA'erne. Lad os tage et eksempel.

Dette eksempel er en enkel og ufuldstændig illustration af, hvordan det hele fungerer. Den fulde forklaring om diskhåndtering får du i et senere brev.

Vi ønsker at læse fra en diskette. Når diskette-data lægges ind på hukommelsen bestemmer MC-68000 placeringen af disse data. Denne del af hukommelsen bliver en opsamlingsplads - en BUFFER - i dette tilfælde en DISKBUFFER. Vi må fortælle hvor mange BYTEs vi skal indlæse, starte diskettestationens motor og til slut starte DISK-DMA'en (dvs. sætte co-processorerne igang med diskette-operationen).

Man plejer at kalde den del af co-processorerne, som udfører disse DMA-operationer, for en "DMA" - selvom de fysisk ikke eksisterer.

DISK-DMA'en vil læse fra disketten og af sig selv lagre dataene i vores DISKBUFFER. Selve hovedprocessoren (MC-68000) er imens fri til at fortsætte med noget andet.

OBS! De data, som man læser fra disketten, må naturligvis lagres i CHIP-RAM. Husker du hvorfor? Fordi det er AGNUS, PAULA og DENISE, som udfører operationerne, og de kan udelukkende benytte den hukommelse, som ligger i området $000000 til $07FFFF (Såkaldt CHIP-RAM). Se evt. BREV I.

Når DISK-DMA'en er færdig med at læse, vil den sende et signal til MC-68000, en INTERRUPT (eller på godt dansk: en afbrydelse, der giver direkte systemadgang).

AMIGAens DMA-kanaler:

TYPE	Antal KANALER
DISK	1
AUDIO	4 (fire lydkanaler)
SPRITE	8 (otte sprites)
BITPLANE	6
COPPER	1
BLITTER	4

Ialt findes der altså 24 DMA-kanaler på AMIGA. Alle DMA-kanaler kan startes og benyttes samtidigt, hvis det er nødvendigt.

Du har sikkert set dit WORKBENCH-skærmbillede mange gange uden at have tænkt over, hvad der egentlig sker når det er der. Det er nemlig mere end du tror! Vi forklarer det først lidt overfladisk, men senere uddyber vi det mere indgående, så bliv ikke nervøs, hvis du

ikke forstår så meget af det her i starten.

Skærmen (med et "fast" billede når du ikke foretager dig noget) benytter COPPER DMA-kanalen og 2 BITPLANE DMA-kanaler. Det er det, at den benytter sig af 2 BITPLANES, som gør at man maksimalt får 4 farver på WORKBENCH-skærmbilledet.

(begrebet BITPLANES vil blive grundigt gennemgået senere).

Markøren - den røde WORKBENCH-pil, som du flytter rundt med musen, bruger en SPRITE DMA-kanal. Når du åbner eller flytter på et vindue (eller en IKON) bruges der op til 4 BLITTER DMA-kanaler. Menuerne i vinduet bruger også BLITTER DMA'en. Når du læser eller skriver til disken bruges DISK DMA-kanalen.

Selve 68000-processoren er nok den, der er mindst belastet takket være alle DMA-kanalerne. Tænk hvilken trist maskine AMIGA havde været, hvis den ikke havde de 3 specielle chips: AGNUS, PAULA og DENISE.....

DMA TIME SLOT ALLOCATION PER HORIZONTAL LINE

Denne overskrift er jo til at få hikke af. At finde en god direkte oversættelse er vanskeligt. Udtalen er let nok. Vi begynder med den: "Di emm ei taim slått allokeishenn pør hårizon-tal lajn."

Og så er det oversættelsen: Det bør kunne oversættes med "DMA tidsforbrug per horison-tal linie". Eller lidt friere oversat: Hvordan tiden, som det tager at tegne en linie, deles op i mindre tidsrum og hvordan disse tidsrum udnyttes af AMIGAens forskellige processorer. Når du har læst dette kapitel, vil du forstå det hele bedre.

Men først må vi tage et sidespring fra AMIGA og forklare, hvordan en monitor - eller for den sags skyld en hvilken som helst tv-skærm - virker.

Inde i monitoren findes der en ELEKTRON-KANON (engelsk: Video beam, udtales video bim, med langt "i" i "bim"), der udsender en stråle af elektroner. I en farve-monitor eller farve-tv findes der til og med tre af slagsen - en for hver af grundfarverne RØD, GRØN og BLÅ. Det er denne elektronstråle, som tegner det du ser på skærmen. Strålen farer henover den fosforbelagte inderside af monitoren eller tv-et. Når dette lag med fosfor rammes af elektronerne, lyset det op et kort øjeblik i det lille punkt hvor elektronstålen rammer. Med de tre grundfarver kan man vise alle de fantastiske farver du f.eks ser i et naturprogram i tv.

Billedet er ikke et fast billede, som står der hele tiden. Det bliver tegnet op 50 gange i sekundet - men for dit øje er det næsten umuligt at opfatte. Du kan dog muligvis se et vist flimmer på skærmen ved visse farvekombinationer.

På europæiske AMIGA-skærme tegnes skærmbilledet af ialt 313 linier. Elektron-kanonen begynder med at tegne linie nummer 1 fra venstre mod højre. Derefter hopper den tilbage og tegner linie nummer 2. Så fortsætter den med linie 3, 4 og 5 osv. indtil den har udfyldt hele skærmen. Så hopper den op fra nederste højre hjørne til øverste venstre hjørne og begynder forfra. Og det som sagt vældig hurtigt - 50 gange i sekundet.

Som en kuriositet kan vi nævne at hver linie tager 0.000064 sekunder - eller 64 mikro-sekunder (milliontedels sekunder).

I løbet af den tid det tager at udfylde hver eneste linie på skærmen - de nævnte 64 mikrosekunder - sker der en hel del ting i AMIGA. Det er umiddelbart ikke så let at forså, men læs det nogle gange, så forstår du det sikkert.

I slutningen af brevet findes der et ark med en figur (FIGUR 1), som illustrerer, hvordan en linie på skærmen ser ud og hvad som sker - og kan ske - mens linien tegnes. Eller med andre ord: Hvad de forskellige DMA'er kan få udført i den tid det tager at tegne en linie på skærmen. Det er det overskriften til dette kapitel beskriver.

Det som du nu skal lære er ikke helt enkelt at forstå med det samme. Noget af det vil du med stor sandsynlighed ikke forstå helt, førend du er kommet længere i kurset.

Det er dog noget af det vigtigste som vi overhovedet kommer til at gennemgå, og du bør lægge noget arbejde i at forstå så meget af det som muligt. Figur 1 kommer du til at bruge ofte, når du programmerer selv, så selv om du ikke forstår alt nu, vil du få mange chancer til at forstå det senere.

Så her skal du begrænse dig til at forså princippet i det hele. Vi kommer lidt efter lidt tilbage til alle detaljerne i kapitlerne SPRITES, BITPLANES, LYD, DISKOPERATIONER osv i de kommende breve.

Se på figur 1 og forestil dig, at den linie du ser der, foregår samtidig med at elektron-kanonen tegner en horisontal linie. Eller med andre ord: Mens elektron-kanonen tegner en horisontal linie, udfører AMIGAen det du ser i figur 1.

DISK DMA'en, SPRITE DMA'en og BITPLANE DMA'en har altid en fast afsat "plads" på denne linie (eller i dette tidsrum) til at udføre deres opgaver på, mens BLITTER DMA'en, COPPER DMA'en og MC-68000 tager "hullerne" som bliver til overs. De hvide "spalter" er såkaldte frie klokkeimpulser som BLITTER, COPPER og MC-68000 også kan bruge.

Det skal også nævnes, at hvis du ikke bruger lydkanalerne (AUDIO) vil disse grå felter blive betragtet som hvide, og derfor være ledige for BLITTER, COPPER og MC-68000 istedet.

MASKINKODE-PROGRAMMERING

Nu skal vi til at se på, hvordan maskinkode-programmeringen foregår. Først beder vi dig lægge mærke til, at vi herefter vil benytte forkortelsen MC for "maskinkode-programmering".

Lad os starte med at se på to ofte benyttede udtryk i MC, nemlig WORD og LONGWORD. Det førstnævnte ord WORD er en betegnelse for to BYTEs (16 BIT). Her sættes altså to BYTEs sammen til en enhed.

Et LONGWORD er en samensætning af to WORD's, altså 32 BIT (eller 4 BYTEs om man vil).

Lad os tage nogle eksempler:

BYTEs:	$0A,$5C,$FF
WORDs:	$8020,$FC9B,$0001
LONGWORDs:	$00DFF180,$00BFE001,$8102FA88

Hovedprocessoren i AMIGA (MC-68000) er en såkaldt 16/32 BIT processor. Det betyder at DATABUSsen er 16 BIT bred. Det engelske ord BUS kan i datasammenhæng forklares som et bundt ledninger (baner) som processorerne i AMIGA benytter til at kommunikere med hinanden og med hukommelsen. Denne BUS gør at MC-68000 på en gang kan overføre 16 BIT til/fra hukommelsen (eller andre dele af maskinen). Betegnelsen 32 betyder, at processoren kan arbejde internt med 32 BITs samtidig. For at processoren skal kunne hente et LONGWORD (32 BIT), må den altså gå ud på DATABUSsen to gange (hente to WORDs, 2*16 BIT = 32 BIT).

Processoren består blandt andet af 16 registre. Disse bruges til at lagre instruktioner og data, som skal behandles. Otte af disse registre kaldes for DATAREGISTRE (D0,D1,D2.....D7). De otte andre kaldes ADRESSEREGISTER (A0, A1,A2.....A7). Alle registrene undtaget A7 (som også kaldes STACKPOINTER, SP), kan benyttes af brugeren. Vi kommer tilbage til brugen af DATAREGISTER efter næste afsnit.

Lad os se på nogle af de vigtigste maskinkode-instruktioner:
MOVE, ADD, SUB og LEA.

MOVE bruges til at flytte data fra et sted i hukommelsen til et andet. For eksempel fra hukommelsen til et register, fra et register til hukommelsen eller fra et register til et andet register.

MOVE-instruktionen har mange varianter, bl.a.:

MOVE.B
MOVE.W
MOVE.L

Når man efter MOVE skriver "**.B**", betyder det at man vil have flyttet et tal, som har størrelsen **BYTE** (8 BIT). Bruges "**.W**", vil man behandle tal, som har et **WORD**'s størrelse (16 BIT). Og som du sikkert har gættet allerede, så betyder "**.L**" efter en instruktion, at man ønsker at behandle et tal af **LONGWORDs** størrelse (32 BIT).

Læg også mærke til, at hvis man med MOVE-instruktionen vil flytte et tal et eller andet sted hen, så bliver det kopieret til det nye sted. Altså: Skriver man MOVE.L $05,D0 – så bliver det tal, som findes i hukommelsescellen med adressen $000005 kopieret ind i dataregister D0. Derefter indeholder både hukommelses-celle $000005 og dataregister D0 samme tal.

ADD-instruktionen bruges til at addere to tal (lægge sammen) SUB-instruktionen bruges til at subtrahere et tal fra et andet (trække fra).

Lad os se på et eksempel:

```
1  start:
2  move.l    #$50,D0
3  add.l     #$10,D0
4  move.l    #$100,D1
5  move.l    #$5,D2
6  sub.l     D2,D1
7  rts
```

Linie 1: Ordet "start" er en instruktion til din assembler – en såkaldt LABEL, (mærkat). Uden kolon "." efter ordet, vil assembleren forsøge at tolke ordet som en maskinkodeinstruktion. Vi forklarer det nærmere i et senere afsnit. I øjeblikket er det nok at huske, at assembleren noterer hvor i hukommelsen programmet lægges. Derefter tager den adressen og lægger den i en "æske", som den kalder "start". på denne måde har den lagret adressen, hvor programmet starter, så når du vil starte dit program skriver du bare (i K-SEKA): jstart.

Linie 2: Læg tallet $50 ind i dataregister D0.

Linie 3: Adder tallet $10 til tallet i dataregister D0.

Linie 4: Læg tallet $100 ind i dataregister D1.

Linie 5: Læg tallet $5 ind i dataregister D2.

Linie 6: Træk tallet i dataregister D2 fra det tal som findes i dataregister D1 **(OBS! svaret bliver liggende i D1)**.

Linie 7: Denne instruktion betyder: RETURN FROM SUBROUTINE, (dansk: returner til hovedprogrammet). Oversættelsen er ikke helt rigtig, men det kommer vi til senere. Vi skal ikke kaste os ud i for meget på en gang –

det bliver rigeligt alligevel. Vi siger bare her, at når du starter programmet fra K-SEKA med ordre "jstart", så vil programmet afsluttes når denne instruktion (RTS) nås og kontrollen over maskinen overtages igen af K-SEKA.

Efter at ovenstående programdel er udført, sidder man med dette facit:

D0 indeholder $60 ($50+$10)
D1 indeholder $FB ($100-$05)
D2 indeholder $05

Her må det være på sin plads at diskutere MC-68000 og nogle af dens såkaldte "adresseringsmodes" - adresseringsmåder.

På en eller anden måde må man jo have en mulighed for at kunne fortælle MC-68000, hvordan den skal behandle de forskellige tal, som den får tildelt. Det gøres ved at skrive tallene (dataene), som skal bruges på forskellige måder. Nedenfor kan du se nogen få eksempler på dette:

```
move.l   #10,D0
move.l   #$10,D0
move.l   $10,D0
```

Den første instruktion flytter det decimale tal 10 ind i register D0. Denne måde at flytte tal ind i et dataregister kaldes "IMMIDIATE ADRESSING" (udtales: immidiejt adressing) og kan oversættes med "øjeblikkelig flytning af data".

Den anden instruktion flytter tallet $10 (HEXADECIMALT) ind i register D0 på samme måde. Navnet på denne data-flytning er derfor den samme som i eksemplet ovenfor.

Den sidste instruktion flytter tallet, som er i hukommelses-celle $000010 ind i D0. Dette er såkaldt "DIREKCT ADRESSING" - udtales dairekt adressing, og vi oversætter det med "direkte adressering".

Du skal ikke bekymre dig om, hvad de forskellige adresserings-måder hedder. Det vigtige er, at du bliver i stand til at se og forstå forskellen når du ser instruktionerne. Og tro os, det bliver du i stand til.

Tegnet "#" angiver, at det er det efterfølgende tal, som skal bruges. Når tegnet "#" mangler foran et tal, betyder det at processoren skal hente et tal fra en hukommelses-celle, og at tallet repræsenterer den hukommelses-celle hvor tallet, som skal bruges, befinder sig. Således betyder MOVE.L $10,D0 som forklaret ovenfor: Gå til hukommelses-celle $000010 og hent tallet som ligger der. Læg det derefter ind i dataregister D0. Denne sidste instruktion kaldes "LOAD EFFECTIVE ADDRESS", frit oversat til dansk: "hent egentlig adresse". Dette forklares senere.

Der findes flere adresseringsmåder (ADRESSERINGSMODES), men dem studerer vi efterhånden som vi får brug for dem senere i dette kursus.

Et andet register i processoren er PROGRAMCOUNTERen (PC – eller på dansk: Programtælleren). Dette register indeholder adressen på den næste instruktion/data, som processoren skal hente ind og udføre/bearbejde. hver gang den henter en instruktion eller data, så tæller PC'eren op, så den ved hvor den skal hente instruktioner eller data næste gang. Dette register holder altså rede på, hvor i programmet processoren befinder sig.

Du kan benytte programtælleren når du programmerer, men vi tror ikke, du kommer til at gøre det. Der findes andre og bedre måder at aktivere dette register på. Mere om dette senere.

Vi tager et eksempel til (ikke "kørbart")

```
1   lea.l   copperlist,a1
2   rts

3   copperlist:
4   dc.w  $2C01,$FFFE
5   dc.w  $00E0,$0000
6   dc.w  $00E2,.....
```

Linie 1: Hent egentlig adresse (.l = LONGWORD) til copper–listen, og læg den i adresseregister nummer 1.

Linie 2: Slut. Gå tilbage til stedet, hvor programmet blev startet.

Linie 3: "Copperlist:" er ikke nogen instruktion. Den bruges bare til at angive et punkt i programmet. Altså, den angiver adressen på den første BYTE i copperlisten ($2C).

Linie 4,5 og 6: Her begynder den del af et større program, som skulle kunne udføres af COPPER.

Programmer, som du skriver i K–SEKA, kan lægge sig hvor som helst der er ledig hukommelsesplads. Det er operativsystemet i AMIGA, som sørger for dette helt automatisk. Vi forklarer mere om dette senere. Altså: Den adresse, som LABELen repræsenrer kan derfor havne hvor som helst i maskinens hukommelse.

Instruktionen "LEA" i vores eksempel finder adessen på copperlisten ved at tælle sig fremad fra instruktionen "LEA" til labelen "copperlist". Det tal den finder, viser hvor mange BYTEs den skal flytte sig fremad for at finde copperlisten. Tallet kaldes en OFFSET.

Forestil dig programmet som en lineal og at instruktionen LEA ligger på 16 cm mærket. I stedet for at sige at copperlisten ligger på 20 cm mærket, kan vi sige at det ligger 4 cm foran den position, hvor programmet er igang lige netop nu.

På samme måde fungerer "LEA". Den nuværende position i programmet findes i program–

tælleren – PC. Det den egentlig gør, er at den tager adressen, som ligger i program-tælleren, lægger afstanden imellem de to instruktioner til, og så lægger den resultatet i A1.

Altså: Afstanden imellem to instruktioner kaldes en OFFSET. Den bruges til at finde instruktions adresse, ved at addere til en anden adresse.

Forstod du det? Hvis ikke, så læs det en gang til. Det er lettere end du tror.

MOST SIGNIFICANT BIT OG LEAST SIGNIFICANT BIT

"MOST SIGNIFICANT BIT" eller MSB betyder "den mest vigtige BIT". "LEAST SIGNIFICANT BIT" eller LSB betyder "den mindst vigtige BIT. Udtrykkene viser altså hvilken BIT i en gruppe af BITs, som har den højeste værdi. I det decimale tal 5005 er det 5-tallet til venstre, som er mest "værd", mens 5-tallet til højre har mindst "værdi"- selv om begge angiver antallet 5 stykker. Se også afsnittet om positions-værdi i BREV i under "DET HEXADECIMALE TAL-SYSTEM". Følgende opstilling forklarer yderligere:

	MSB						LSB	
BIT NR. :	7	6	5	4	3	2	1	0
BYTE :	1	0	1	0	1	0	1	0 (170 eller AA)
BIT-VÆRDI :	128	54	32	16	8	4	2	1

Dette viser en BYTE og dens 8 BIT. Som du ser har BIT nr. 7 værdien 128. Værdien 128 er den største i denne BIT-gruppe, og derfor kalder vi BIT nr. 7 for MSB. BIT 0 har værdien 1, og er altså den mindste i BIT-gruppen og får betegnelsen LSB.

Disse betegnelser bruges som regel i tabeller eller skemaer for lettere at kunne henvise til hvilken BIT vi snakker om. Vi vil give dig flere eksempler på dette i et senere brev.

STANDARDER I MASKINKODE MED K-SEKA

Før vi kaster os ud i maskinkode-progammering, må vi se lidt på, hvordan K-SEKA organiserer det hele og hvordan man skriver sine programmer korrekt.

Nedenfor har vi sat en linie med en typisk maskinkode-instruktion op. Vi vil nu vise hvad de forskellige dele i opsætningen betyder.

FELT 0	FELT 1	FELT 2	FELT 3
23	move.l	$04,a6	;EXEC-adresse ind i a6

FELT 0: I dette felt skriver assembleren linienummeret. Det gør den helt automatisk, så det behøver du ikke tænke på. Linienumrene gør det lettere for dig at finde rundt i større programmer.

FELT 1: I dette felt placeres selve maskinkode-instruktionen. LABELS placeres også i dette felt.

FELT 2: Her placeres de data eller ordrer, som instruktionen i FELT 1, skal bearbejde eller udføre. Kommaet skiller feltet i to dele. Første del kaldes operatøren og anden del kaldes operanden, Venstre del kan man også betegne som FRA-delen og højre del som TIL-delen. I eksemplet bliver det til : Hent fra $04 og læg ind i a6.

FELT 3: I dette felt kan man skrive kommentarer. For at det skal opfattes som en kommentar, skal man starte med et semikolon (;). Hvis du ikke gør det, får du en fejlmelding fra assembleren, når du prøver at assemble dit program.

Man bruger som regel små bogstaver og tal når man skriver sine programmer i K-SEKA. Man kan udmærket bruger versaler (store bogstaver) men vi synes det er for tæt en skrift og derfor uoverskuelig. Du prøver dig naturligvis frem for at finde ud af, hvad der passer dig bedst.

I vores program-eksempler vil vi i begyndelsen bruge både store og små bogstaver for lettere at kunne fremhæve specielle ting. Langsomt, men sikkert vil vi dog gå over til udelukkende at bruge små bogstaver.

I begyndelsen af din programmør-karriere bør du skrive mange kommentarer i dine maskinkode-programmer. Hvis du ikke gør det, vil det blive meget tidskrævende senere at finde ud af, hvad de forskellige rutiner udfører.

Efter et stykke tid vil du sandsynligvis (ganske sikkert) have glemt, hvor alle de forskellige funktioner i dit program ligger. Et kommentar-rigt program er let at finde rundt i.

Desuden kan en anden fortsætte ud fra dine ideer og udvikle dit program videre, hvis

han/hun har muligheden for at finde ud af, hvilke dele i programmet der udfører hvad.

Når man i maskinkode-programmering snakker om rutiner, mener man følgende:

Programmet du skriver, kan betragtes som et eneste stort værksted, hvor hver arbjder er specialist i at udføre et bestemt job. Lad os for enkelthedens skyld tænke os et postkontor med sådanne special-uddannede arbejdere. En person har til opgave at åbne postkontoret hver dag. En anden henter posten, som er lagt i postkassen, en tredie igen sortererer post, en fjerde bærer den ud til kunderne - og så videre.

Alle som arbejder på postkontoret har hver sin egen specielle opgave at udføre. Således er det også i dit program. En programdel tager imod de data du skriver på skærmen. En anden del sorterer dem - en del lagrer dem - en del holder orden på, hvor mange data du har skrevet ind - og så videre - alt efter hvad dit program egentlig skal gøre.

Disse forskellige dele i et program kaldes for RUTINER. Hovedprogrammet kan hoppe til en RUTINE, som beder dig (for eksempel) bekræfte, at du vil slette noget på disketten. Når du bekræfter at det er det du ønsker, går programudførelsen tilbage til det punkt, hvorfra denne UNDERRUTINE (engelsk: SUBROUTINE, oversat: SUB- eller UNDER-RUTINE) blev kaldt frem. Når man programmerer MC68000 gøres det med kommandoen RTS (engelsk: RETURN FROM SUB-ROUTINE, oversat: RETURNER FRA SUB-RUTINE).

Det er meget vigtigt at denne instruktion ikke udelades når du skriver dine programmer. Og du må passse på at de placeres på det rigtige sted, ellers bryder dit program sammen. Men det skal vi sørge for, at du gør, da vi længere fremme tager os specielt af denne kommando i detaljer.

HAR DU EKSTRA RAM I AMIGA?

Hvis du har ekstra hukommelse i din AMIGA, må du koble den ud for at kunne køre program-eksemplerne i dette kursus (undtagen eksempel nummer 1 i BREV I). Grunden er, at hvis du har en copperliste i dit MC-program, vil copperlisten sammen med resten af programmet havne i FAST-RAM. Som du sikkert husker er COPPEREN en del af AGNUS, PAULA og DENISE, og de kan kun arbejde i CHIP-RAM.

For at koble ekstra-hukommelsen ud, kan du starte AMIGA op med WORKBENCH-disketten - sæt disketten ind og RESET. Når den er indlæst, åbner du SYSTEM-skuffen og starter programmet NOFASTMEM. Derefter skal du gå ud i CLI. (K-SEKA kan kun startes derfra).

Sæt derefter K-SEKA-disketten ind og skriv "df0:seka" (Hvis du har en ekstra disket-testation kan du skrive "df1:seka").

COPPER

COPPER står for CO-PROCESSOR (Hjælpe- eller samarbejdsprocessor). COPPEREN er en processor, som kun har 3 instruktioner (Hovedprocessoren 68000 har over 100).

Instruktionerne til COPPEREN er: **wait** (vent), **SKIP** (hop over) og **MOVE** (flyt data).

COPPEREN kan kun programmeres til at udføre specielle ting. For at skrive et program til COPPEREN (en såkaldt "COPPERLISTE") i et maskinkode-program, skal vi bruge maskinkode-instruktionen "**DC.**" (engelsk: Declare word, oversat: definer to BYTES). Lad os tage et eksempel i et lille maskinkode-program:

COPPERen bruges mest sammen med BITPLANEs (meget mere om det i BREV III og IV), altså grafik-skærmen.

```
1   move.w      #$01A0,$DFF096
2   lea.l       copperlist,A1
3   move.l      A1,$DFF080
4   move.w      #$8080,$DFF096
5
6   wait:
7   btst        #6,$BFE001
8   bne         wait
9
10  move.w      #$0080,$DFF096
11  move.l      $04,A6
12  move.l      156(A6),A1
13  move.l      38(A1),$DFF080
14  move.w      #$81A0,$DFF096
15  rts
16
17  copperlist:
18  dc.w        $9001,$FFFE
19  dc.w        $0180,$0F00
20  dc.w        $A001,$FFFE
21  dc.w        $0180,$0FFF
22  dc.w        $A401,$FFFE
23  dc.w        $0180,$000F
24  dc.w        $AA01,$FFFE
25  dc.w        $0180,$0FFF
26  dc.w        $AE01,$FFFE
27  dc.w        $0180,$0F00
28  dc.w        $BE01,$FFFE
29  dc.w        $0180,$0000
30  dc.w        $FFFF,$FFFE
```

En foreløbig forklaring:

Linienummer uden tekstlinie er kun indsat som tomrum for at programmet skal blive mere overskueligt.

Linie 1–4: Blanker (renser) skærmen, afbryder WORKBENCH–markøren (den røde pil) og starter vores egen COPPERLISTE.

Linie 6–8: Denne lille rutine går i en loop (uendelig cirkel) og venter på, at du trykker på venstre mus–knap, når du vil afslutte programmet.

Linie 10–15: Henter den gamle COPPERLISTE tilbage (den som bruges til at tegne WORKBENCHen med) og starter den.

Linie 17–30: Dette er vores egen COPPERLISTE, som former en rød, hvid, blå, hvid og rød linie (felt).

Der er en hel del i dette program, som behøver en nærmere forklaring. Det er dog nogle kundskaber vi endnu ikke har formidlet, så vi vælger at tage dette op igen når vi forklarer BITPLANEs i BREV III og IV.

Der forklarer vi dette eksempel detaljeret, beskriver instruktionerne som RTS, og fænomener som forgreninger (hop) til andre steder i programmet (BRANCHing), BIT–testing og STATUS–registret.

Som du sikkert har på fornemmelsen, har vi kun skrabet på overfladen af alle de muligheder, som COPPERen giver os i programmerings–sammenhæng. I de to næste breve, hvor vi som nævnt vil beskrive BITPLANEs, skal vi også beskæftige os mere indgående med COPPERen. Det bliver lettere for dig at forstå både BITPLANE og COPPER, når vi kan forklare disse to fænomener samtidig.

LØSNINGER TIL OPGAVER I BREV I

OPGAVE 0101:

Decimalt	Binært	Decimalt	Binært
0	0000 0000	147	1001 0011
14	0000 1110	155	1001 1011
15	0000 1111	156	1001 1100
16	0001 0000	157	1001 1101
27	0001 1011	158	1001 1110
45	0010 1101	200	1100 1000
63	0011 1111	213	1101 0101
64	0100 0000	228	1110 0100
65	0100 0001	229	1110 0101
98	0110 0010	240	1111 0000
99	0110 0011	245	1111 0101
100	0110 0100	253	1111 1101
101	0110 0101	254	1111 1110
133	1000 0101	255	1111 1111

OPGAVE 0102:

Decimalt	HEX	Decimalt	HEX
30	1E	255	FF
31	1F	256	100
32	20	257	101
33	21	2747	ABB
63	3F	2748	ABC
64	40	2749	ABD
65	41	4095	FFF
66	42	4096	1000
127	7F	4097	1001
128	80	4098	1002
129	81	4099	1003
130	82	5000	1388
170	AA	10000	2710
171	AB	20000	4E20
172	AC	43980	ABCC
254	FE	43981	ABCD

OPGAVE 0103:

BINÆRT	**HEX**
0000001001101000	0268
1010101111001101	ABCD

HEX	**BINÆRT**
ABCD	1010101111001101
FFFF	1111111111111111
D5C6	1101010111000110

OPGAVER TIL BREV II

Opgave 0201: Forklar hvorfor vi kan påstå, at der findes 24 DMA-kanaler i AMIGA

Opgave 0202: Hvad er en SUBROUTINE i et program?

Opgave 0203: Hvor mange BYTEs er der i et WORD, og hvor mange BIT?

Opgave 0204: Hvor mange BYTEs er der i et LONGWORD, og hvor mange BIT?

Opgave 0205: Hvor mange ADRESSERINGSREGISTRE findes der i MC-68000? og hvad er deres betegnelse?

Opgave 0206: Hvor mange DATAREGISTRE findes der i MC-68000? og hvad er deres betegnelse?

Opgave 0207: Hvad gør følgende instruktion MOVE.L $10,A3?

Opgave 0208: Hvad menes med MSB og LSB?

Opgave 0209: Hvor mange instruktioner har COPPERen og hvilke er det?

Opgave 0210: Hvad er en ELEKTRONKANON?

KORT DATA-ORDBOG

ADRESSEREGISTER Betegnelsen på et af de 8 interne registre i MC-68000 - a0 til a7.

COPPER Forkortelse for CO-PROCESSOR. PROCESSOR med kun tre instruktioner. Bruges ofte i grafik-sammenhæng.

DMA "DIRECT MEMORY ACCES". En funktion i AMIGA, som selv henter data fra hukommelsen - uden hjælp af andre processorer.

LONGWORD En sammensætning af 4 BYTEs (32 BIT)

LSB LEAST SIGNIFICANT BIT. Betegnelse på den BIT i en BIT-gruppe, som har den mindste positionsværdi.

MSB MOST SIGNIFICANT BIT. Betegnelse på den BIT i en BIT-gruppe, som har den største positionsværdi.

PROGRAMCOUNTER Forkortes PC. Det register i MC-68000, som holder orden på hvilken instruktion i et program, som skal udføres næste gang.

SUBROUTINE Underprogram i et større program. Ofte en speciel måde at kunne udføre en speciel opgave på.

VIDEO BEAM Den stråle af elektroner, som kommer fra monitorens elektron-kanon og som tegner billedet på skærmen, punkt for punkt.

WORD En sammensætning af to BYTEs (16 BIT).

KOMMENTARER TIL BREV II

Dette var BREV II. Det er et lidt vanskeligere brev end det første, men det burde ikke være så indviklet at lære det som står heri.

Opgaverne i BREV II, kan du som tidligere nævnt sende ind til os og få rettet, Det er dog ikke nødvendigt, fordi løsningerne til et brev altid findes i det følgende brev. Så i BREV III finder du løsningerne til opgaverne i dette brev.

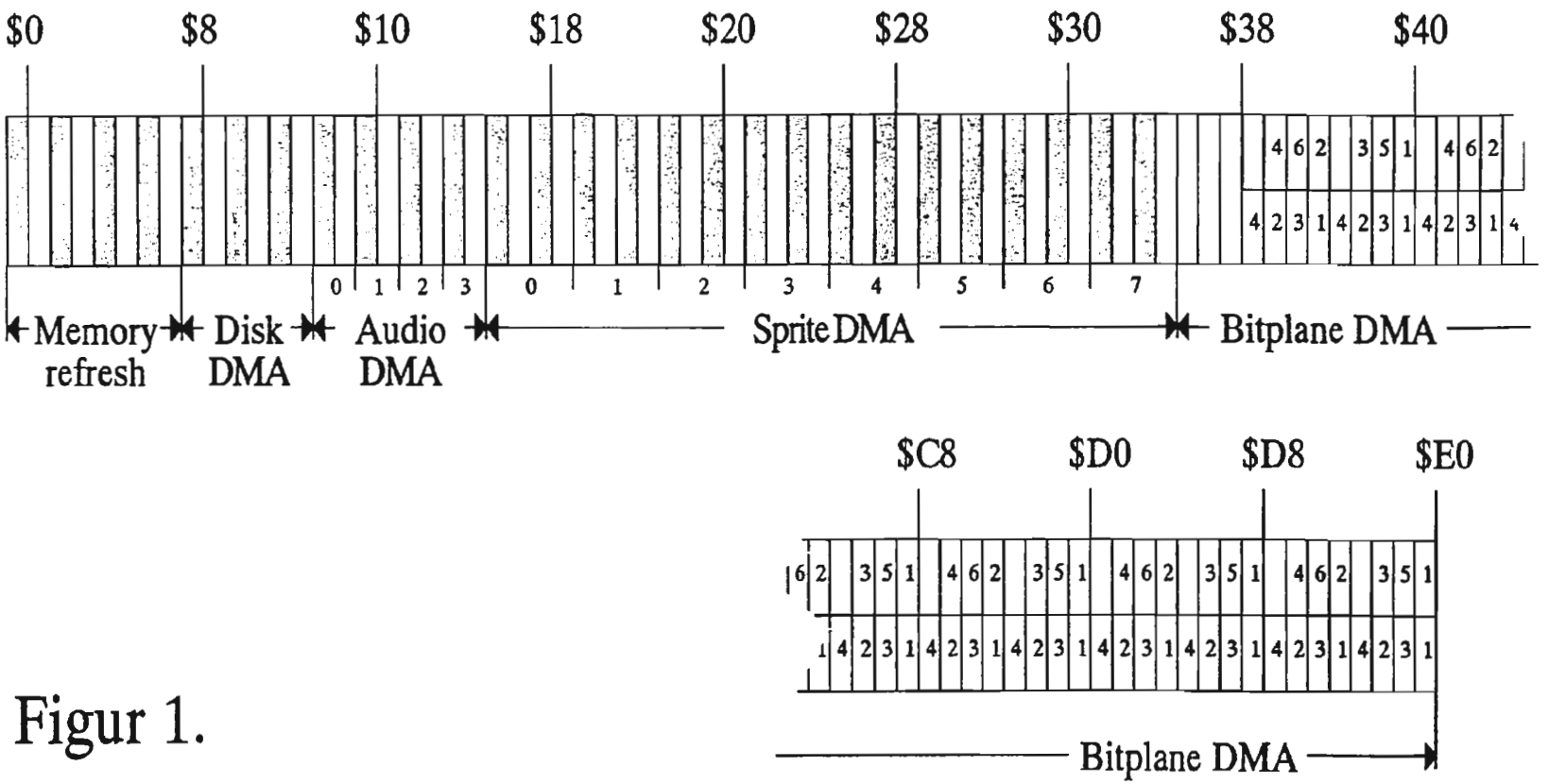
Efter den 20. i måneden udsendes det næste brev "automatisk" pr. efterkrav medmindre vi har modtaget din forudbetaling på vedlagte girokort. Hvis vi modtager en forudbetaling udsættes næste "automatiske" udsendelse til måneden efter, så du ikke risikerer at få et brev til sendt pr. efterkrav i samme måned som du har forudbetalt.

Hvis du forudbetaler sparer du efterkravsgebyret og portoen.
På denne måde bliver kurset væsentlig billigere for dig.

Fortsat god fornøjelse!

DATASKOLEN

Carsten Nordenhof


$0
$8
$10
$18
$20
$28
$30
$38
$40
0 1 2 3 0 1 2 3 4 5 6 7
Memory refresh
Disk DMA
Audio DMA
Sprite DMA
Bitplane DMA
$C8
$D0
$D8
$E0
Bitplane DMA

Figur 1.