Programmering i maskinkode på *AMIGA*

A.Forness & N.A.Holten

Hæfte 1

Indhold

DATASKOLEN
Postboks 62
Nordengen 18
2980 Kokkedal

Telefon 42 24 96 57

Postgiro 7 24 23 44

KOKKEDAL, 17.02.1990

Kære kunde, studerende og elev på DATASKOLEN !

Velkommen på DATASKOLEN og tak for, at du valgte vort kursus.

Det er meget vigtigt at du ikke lader familie, venner og bekendte kopiere dette brevkursus. Hvis du gør det, bryder du for det første loven om ophavsret, og dernæst så medfører "piratkopiering" at DATASKOLEN ikke får råd til at udvikle og distribuere flere brevkurser - til skade for andre progammører over hele landet.

Derfor denne indtrængende opfordring:

KOPIER IKKE DETTE BREVKURSUS.

Med venlig hilsen
DATASKOLEN

Carsten Nordenhof

INDHOLDSFORTEGNELSE

Brev 8: Spiller en SAMPLE
Eksempel i maskinkode
Laver en enkel musikrutine
Eksempel i maskinkode
Forklarer MIDI
Maskinkode VII

Brev 9: Forklarer INTERRUPTS
Laver et KEYBOARD-INTERRUPT
Eksempel i maskinkode
Maskinkode VIII

Brev 10 Laver en SCROLL-tekst fra diskette
Eksempel i maskinkode
ALLOKERER hukommelse
En rutine som skriver/læser et spor på en diskette
Eksempel i maskinkode
Maskinkode IX

Brev 11: Laver en MUS-rutine
Eksempel i maskinkode
Laver en PRINTER-DRIVER
Eksempel i maskinkode
Maskinkode X

Brev 12: Maskinkode TIPS & TRICKS
Opsætter en HI-RES skærm (med billede)
Opsætter en HAM-skærm (med billede)
Forklarer INTERLACE
En TIDSTABEL for MC-68000
Laver en linjetegningsrutine
Bølger i dine billeder
Hvordan du laver din egen DEMO

INTRODUKTION

Velkommen til dette brevkursus i MASKINKODE-programmering på den fantastiske AMIGA fra COMMODORE. Der er mange verden over, som har anskaffet sig denne avancerede hjemmedatamat, men det er kun meget få, der kan udnytte alle dens indbyggede muligheder. COMMODORE har allerede solgt over 1.000.000 enheder, så hvis du beslutter dig til at skrive programmer til den, har du et stort marked at tjene penge på.

Når du har gennemgået samtlige breve - 12 ialt - vil du kunne skrive dine egne programmer på maskinen, hvad enten det er spil eller andre applikationer.

Men husk, det er ikke nok bare at sætte sig ned og læse dette brevkursus og så tro at kundskaberne sætter sig fast af sig selv. Du skal belave dig på at prøve dig frem, eksperimentere og øve meget for at beherske programmeringen godt.

Det er meget vigtigt at du lærer alt fra grunden. Jo bedre din grundviden er, desto lettere bliver det for dig senere at beherske AMIGA's avancerede sider.

Til brev 1-11 udgives en diskette, hvor alle eksempler fra brevkurset findes lagret både som KILDEKODE og OBJEKT KODE. Disse to udtryk hedder på engelsk "SOURCE CODE" og "OBJECT CODE" og vil blive forklaret mere udførligt senere i dette første brev.

I hele kurset vil vi forsøge at bruge både danske og engelske fagudtryk for at gøre det nemmere for dig at forstå terminologien, hvis du senere vil læse speciallitteratur på engelsk.

Til det sidste brev - brev nummer 12 - vil der blive også blive udgivet en diskette. Dette brev vil indeholde TIPS OG TRICKS både specielt for AMIGA og generelt for MASKINKODE-programmering. Du skal dog have gennemgået samtlige 12 breve for at kunne få fuldt udbytte af disketten.

Hvis du får vanskeligheder med at få en lille programdel til at fungere, kan du sende den til os på diskette sammen med en frankeret svarkuvert. Vi vil så gøre vort bedste for at finde fejlen og returnere disketten sammen med en forklaring på, hvad du gjorde forkert. Vi kan desværre ikke påtage os at rette programmeringsproblemer telefonisk.

Vi vil gerne afslutte denne introduktion med at ønske dig held og lykke. GIV IKKE OP! selv om noget til tider kan være vanskeligt at forstå, så er det trods alt lettere end du tror. Læs afsnittet endnu en gang, og skriv et lille program du kan teste det hele med.

HUSK, ØVELSE GØR MESTER !

DET BINÆRE TALSTYSTEM

Noget af det første man får brug for når man skal programmere i C er et talsystem, der ser lidt anderledes ud end det man er vant til. (Hvis du allerede er godt inde i både det BIN[RE og HEXADECIMALE talsystem, kan du springe dette over og gå direkte til næste kapitel.)

Det almindelige talsystem regner i grupper på 10. Hvis vi for eksempel skriver tallet 4362, og mener kroner, så ved alle, at det betyder vi har 2 1-kroner, 6 10'ere, 3 100-kronesedler og 4 1000-kronesedler. Man kan jo for den sags skyld have beløbet i bar 10-ører (eller rettere kunne), men den totale sum kan beskrives som nævnt.

Rent matematisk svarer de forskellige positioner af cifrene i tallet 4362 til potenser af 10:

$4*10^3 + 3*10^2 + 6*10^1 + 2*10^0 = 4000 + 300 + 60 + 2 = 4362$

Læg mærke til at eksponenten til 10 stiger med 1 for hver plads man rykker fra højre mod venstre. Det var altså vores "almindelige" talsystems opbygning;

Men hvad er et BINÆRT talsystem, og hvorfor skal man bruge det?

Når man skriver på tastaturet (engelsk : KEYBOARD, udtales kibård), så registrerer maskinen hvilken tast du har trykket på. Dermed ved den også hvilket tegn den skal skrive ud på skærmen. Dette lagrer den i sin hukommelse i en lille "kasse" - en såkaldt bitgruppe eller en **BYTE** (udtales bajt).

Denne BYTE består af 8 dele som hver kaldes en BIT. En BIT er i princippet en ledning som der kan være spænding på - eller ikke spænding på. Den kan altså være enten tændt eller slukket. I datasammenhæng betegnes disse to tilstande som 1 når der er strøm i ledningen og 0 (nul) når der ikke er strøm i ledningen.

En BYTE (bitgruppe) består altså af 8 BIT. Hvis man nu skriver 100 1100 for at beskrive tilstanden i BYTE'en, så betyder det, at der er strøm i 3 ledninger og at 5 er uden strøm.

Den BIT længst til højre kaldes BIT nummer 0. Den til venstre for nummer 0 kaldes BIT 1, den næste kaldes BIT 2 og sådan fortsætter det til den sidste BIT, som får betegnelsen BIT 7. Altså ialt 8 BIT's som nummereres fra 0 til 7.

Det kan være hensigtsmæssigt at dele en BYTE i to: fire BIT's i hver gruppe. Dette kaldes en halv bitgruppe (engelsk: NIBBLE, udtales nibl - med kort "i"). Vi får brug for denne NIBBLE i det HEXADECIMALE talsystem, men den bruges også ofte i det BINÆRE talsystem - som du snart får at se.

Det store bogstav "A" lagres som tallet 65 i en BYTE. Lad os se, hvordan det gøres.

Som nævnt benytter maskinen sig af 8 BIT's (ledninger) med spænding på nogle af ledningerne for at markere en værdi. Værdien 65 (for "A") skrives på denne måde: 01000001

BITen længst til højre (BIT 0) tæller enere (0'te potens). BIT 1 lige til venstre herfor tæller grupper med 2'ere. Næste BIT mod venstre tæller grupper med 4'ere, næste igen (BIT 3) grupper med 8'ere og sådan fordobles værdien hver gang vi flytter en plads mod venstre.

Studer nøje denne opstilling:

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	0	0	0	0	0	1

Det binære tal (0100001) vi bruger som eksempel står i nederste linie. Som du kan se får de to 1-taller værdierne 64 og 1. Summen af de to er 65 og det repræsenterer her bogstavet "A".

Lad os se nærmere på BIT 0 og BIT 1. BITen længst mod højre repræsenterer som nævnt antal enere. I det BINÆRE talsystem er det 2 som er grundenheden på samme måde som 10 er det i det DECIMALE talsystem, vi bruger til dagligt. Tallet 2 i 0'te potens (2^0) er 1. Ethvert tal der opløftes til 0'te potens er lig med 1. Altså $2^0 = 1$, $10^0 = 1$, $16^0 = 1$

Ethvert tal der opløftes til 1. potens er lig med tallet selv. Altså 2^1 (to i første) = 2, $10^1 = 10$, $16^1 = 16$

Vi kan nu skrive vort BINÆRE tal 0100001 ud matematisk som vi gjorde i det DECIMALE talsystem:

$0*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 1*2^0 =$

$0*128 + 1*64 + 0*32 + 0*16 + 0*8 + 0*4 + 0*2 + 1*1 =$

$0 + 64 + 0 + 0 + 0 + 0 + 0 + 1 = 65$

Studer grundigt tabellen på næste side, hvor vi har oversat de BINÆRE tal fra 1 til 12 til DECIMALE tal.

Når man vil vise, at et tal (f.eks. 65) er skrevet i det DECIMALE talsystem skriver man : 65_{10}

Vil man derimod angive at det er et BINÆRT tal (f.eks. 0100 0001) skriver man: 01000001_2

Her er de første tolv tal i det BINÆRE talsystem - det system der har tallet 2 som basis (grundtal):

$0000\ 0000_2 = 0_{10}$

$0000\ 0001_2 = 1_{10}$

$0000\ 0010_2 = 2_{10}$

$0000\ 0011_2 = 3_{10}$

$0000\ 0100_2 = 4_{10}$

$0000\ 0101_2 = 5_{10}$

$0000\ 0110_2 = 6_{10}$

$0000\ 0111_2 = 7_{10}$

$0000\ 1000_2 = 8_{10}$

$0000\ 1001_2 = 9_{10}$

$0000\ 1010_2 = 10_{10}$

$0000\ 1011_2 = 11_{10}$

$0000\ 1100_2 = 12_{10}$

Ved hjælp af disse otte BITs i en BYTE kan man skrive 256 tal: fra 0 til 255. Det kan man fordi summen af alle tallene som de 8 BITs repræsenterer hvis de alle er sat (har værdien 1) netop giver 255:

BIT 7 BIT 6 BIT 5 BIT 4 BIT 3 BIT 2 BIT 1 BIT 0

128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255

BINÆRT : 1111 1111

Det er meget vigtigt at du forstår dette. Ja det er så vigtigt, at vi gerne skriver det en gang til: DET ER MEGET VIGTIGT AT DU FORSTÅR DETTE. Læs det igennem så mange gange at du forstår systemet uden nogen form for tvivl. Øv dig derefter på at oversætte tal fra DECIMALT til BINÆRT.

Du vil spare meget programmeringstid senere, hvis denne del af de kundskaber du skal have for at programmere AMIGAen sidder på rygmarven. Opgiv ikke selvom du måske ikke forstår det hele på en gang. Læg det lidt til side og prøv igen lidt senere !

Løs nu opgave 0101, som du eventuelt kan sende til os sammen med de andre opgaver i brev 1. Glem ikke at vedlægge en adresseret og frankeret svarkuvert. Løsningen kommer i næste brev, så det er ikke nødvendigt at sende opgaverne ind. Det gør du kun, hvis du har lyst til det, men vi retter gerne !

OPGAVE 0101:

Omskriv de DECIMALE TAL 0, 14, 15, 16, 27, 45, 63, 64, 65, 98, 99, 100, 101, 133, 147, 155, 156, 157, 200, 213, 228, 229, 240, 245, 253, 254, 255 til BINÆRE TAL.

DET HEXADECIMALE TALSYSTEM

Nu håber vi du kan det BINÆRE talsystem på fingrene, og vi fortsætter derfor med det HEXADECIMALE talsystem.

Det BINÆRE talsystem havde basistallet 2. Det system vi bruger til dagligt er det DECIMALE talsystem med basistallet 10. Det HEXADECIMALE talsystem, som vi skal lære nu, har basistal 16.

I den del af dette brev, der omhandlede BINÆRE tal, benyttede vi os af et "forsænket" 2-tal for at vise, at det var et BINÆRT tal, og efter det DECIMALE tal satte vi tilsvarende et "forsænket" 10-tal. I næsten al datalitteratur bruger man en skrivemåde, der passer bedre til skrivemaskiner. Vi indfører her denne skrivemåde i resten af kurset:

Foran et BINÆRT tal skrives % foran et HEXADECIMALT tal skrives $, og foran et decimalt tal ingenting.

Tallet 65 som vi brugte før kan da skrives på følgende måder:

BINÆRT:	%010001
DECIMALT:	65
HEXADECIMALT:	$41

Når man udtaler det BINÆRE tal nævnes hvert ciffer for sig: nul-en-nul-nul-nul-nul-nul-en og ikke en million og en. På samme måde siges HEXADECIMALT: fire-en og ikke enogfyrre.

De som har arbejdet med programmering i et stykke tid bryder ofte disse regler og siger hex-enogfyrre om det DECIMALE tal 65. Det vigtigste i denne sammenhæng er jo, at andre forstår hvad du mener. Det korrekte er altså: Hex-fire-en når det siges og $41 når det skrives på papir eller skærm. I noget litteratur vil du finde et "&" for at angive et HEXADECIMALT tal. På AMIGA bruger alle dog et dollar-tegn.

Lad os nu se på hvordan det HEXADECIMALE talsystem opfører sig.

I det DECIMALE system bruger vi tallene fra 0 til 9 (basis er 10), og i det BINÆRE system bruger vi tallene 0 og 1 (basis er 2). Det HEXADECIMALE system har 16 som basis og skal bruge tal fra 0 til 15 for at kunne skrives. Problemet er, at vi kun har 10 forskellige tal til rådighed, nemlig 0, 1, 2, 3, 4, 5, 6, 7, 8 og 9. Det har man løst på den måde, at man bruger bogstaver når tallet 9 passeres.

Man benytter sig af de seks første bogstaver i alfabetet : A, B, C, D, E og F.(Du må godt bruge små bogstaver, men i dette brev vil vi bruge store for tydelighedens skyld.) Når vi skal tælle HEXADECIMALT fra 0 til 15 må vi sige: 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Det betyder at i dette talsystem er A det samme som 10_{10}, B bliver 11_{10}, C svarer til 12_{10}, D modsvarer 13_{10}, E er 14_{10} og F repræsenterer 15_{10}.

Nu er vi i stand til at lave en oversigt over opbygningen af tallet. Det foregår på samme måde som vi viste under BINÆRE og DECIMALE tal.

I selve tallet er der kun 6 cifre, for eksempel: $DFF180

Cifret "0" i $DFF180 viser hvor mange enere (16^0), der er i tallet. I dette tilfælde: ingen. Cifret til venstre for "0" er "8". Det viser hvor mange grupper der er med 16 (16^1), her altså 8 grupper. Det næste ciffer i rækken er "1", det viser hvor mange grupper med 256 (16^2) der er i tallet. Til venstre herfor står "F" Dette bogstav repræsenterer 15 HEXADECIMALT, og viser hvor mange grupper der er med 4096 (16^3), her er der altså 15 grupper.

Lad os vise hvad vi mener før det bliver for vanskeligt at følge med:

POS-VÆRDI	16^5	16^4	16^3	16^2	16^1	16^0
HEX	D	F	F	1	8	0

POS-VÆRDI (POSITIONS-VÆRDI) er den værdi hvert ciffer tillægges, afhængigt af dets placering (position) i tallet. I de to DECIMALE tal 400 og 4000 har 4-tallet som sådan samme værdi hele tiden nemlig fire. Men positionsværdien er mindst i 400, fordi 4-tallet der viser grupper med ethundrede enheder.

I det andet tal 4000 er tallet fire stadig fire, men dets positionsværdi er 10 gange større. Nu viser det antallet af grupper med ettusinde enheder.

HEXADECIMALE tal fungerer på samme måde som både BINÆRE og DECIMALE tal. Jo længere til højre i et tal et ciffer (eller bogstav) befinder sig, desto lavere er dets positionsværdi.

8-tallet i $DFF180 har altså lavere POS-VÆRDI end 1-tallet, selvom det naturligvis er rigtigt at 8-tallet som sådan er større end 1-tallet.

De forskellige positioner i et HEXADECIMALT tal, har følgende værdier:

Position 6:	16^5	=	1.048.576
Position 5:	16^4	=	65.536
Position 4:	16^3	=	4.096
Position 3:	16^2	=	256
Position 2:	16^1	=	16
Position 1:	16^0	=	1

Lad os så regne ud, hvor stort det HEXADECIMALE tal $DFF180 er, når det omregnes til et DECIMALT tal.

POS-VÆRDI	16^5	16^4	16^3	16^2	16^1	16^0
HEX	D	F	F	1	8	0

DECIMALT 1048576*13 + 65536*15 + 4096*15 + 256*1 + 16*8 + 1*0

Position 6:	1.048.576	* 13	=	13.631.488
Position 5:	65.536	* 15	=	983.040
Position 4:	4.096	* 15	=	61.440
Position 3:	256	* 1	=	256
Position 2:	16	* 8	=	128
Position 1:	1	* 0	=	0
IALT	DECIMALT			14.676.352

Allerede her kan man se en af fordelene ved HEXADECIMALE tal. De er meget kortere og derfor meget lette at huske. $DFF180 er for øvrigt farveregister nummer 0 (den såkaldte baggrunds-farve) på AMIGA. Vi vender tilbage til dette tal mange gange, så vi vil ikke forklare mere om det lige nu; men det er i hvert fald meget lettere at huske $DFF180 end 14.676.353, ikke sandt?

Her er de tyve første DECIMALE tal i talrækken oversat til HEXADECIMALT:

1	$0001	11	$000B
2	$0002	12	$000C
3	$0003	13	$000D
4	$0004	14	$000E
5	$0005	15	$000F
6	$0006	16	$0010
7	$0007	17	$0011
8	$0008	18	$0012
9	$0009	19	$0013
10	$000A	20	$0014

Altså: når man i HEX har talt til 15 ($F) og adderer 1, så begynder man på en ny gruppe med 16-ere. Denne ener placeres i værdiposition 2 og i værdiposition 1 begynder man forfra med 0. Eller med andre ord: så snart en værdiposition er kommet til cifret "F" og man adderer 1, så nulstilles denne position og man forhøjer positionen umiddelbart til venstre med 1. Det er det samme som sker DECIMALT fra 9 til 10, fra 19 til 20 eller fra 1499 til 1500.

OPGAVE 0102: Her er nogle DECIMALE tal du kan oversætte til HEXADECIMALT: 30, 31, 32, 33, 63, 64, 65, 66, 127, 128, 129, 130, 170, 172, 254, 255, 256, 257, 2747, 2748, 2749, 4095, 4096, 4097, 4098, 4099, 5000, 10000, 20000, 43980, 43981.

OMREGNING MELLEM BINÆRE OG HEXADECIMALE TAL

Det kan være et både besværligt og kedeligt arbejde at omregne fra DECIMALT til BINÆRT, eller fra DECIMALT til HEXADECIMALT. Vi anbefaler at du bruger en regnemaskine. CASIO og HEWLETT PACKARD har gode kalkulatorer til basis-omregninger. Disse kalkulatorer kan også omregne OKTALE tal (tal med basis 8), men det får du ikke brug for i dette kursus, der er rettet specielt mod AMIGA.

Hvis du anskaffer dig en kalkulator, der kan omregne mellem DECIMALE og BINÆRE/HEXADECIMALE tal, sparer du meget arbejde. Det kan også være nyttigt at kunne omregne et BINÆRT tal til et HEXADECIMALT, eller omvendt. Det er imidlertid så let, at du med en smule øvelse kan gøre det hurtigere end med en kalkulator.

Det gode ved denne metode er, at du ikke behøver lære hvorfor det fungerer som det gør. Det forstår du automatisk og uden anstrengelser når du har gjort det nogle gange.

Lad os tage to BYTEs, opdele dem i NIBBLEs og sætte dem BINÆRT ved siden af hinanden - f.eks. %1001 1110 1011 0101.

Hver NIBBLE består af fire BINÆRE cifre. Nu giver du de fire positioner i hver NIBBLE positionsværdierne 8-4-2-1, fra venstre mod højre. Derpå summerer du de positoner i NIBBLEn som er sat (tændt eller 1) og noterer det ned HEXADECIMALT.

Hvis vi tager fat på den NIBBLE, der er længst mod venstre i vort tal - altså den NIBBLE som er %1001 - så får vi dette HEXADECIMALE resultat:

```
POS-VÆRDI :    8       4       2       1
NIBBLE    :    1       0       0       1
DECIMALT  :    8   +   0   +   0   +   1   =   9
```

Det DECIMALE tal 9 er også 9 HEXADECIMALT ($9). Næste NIBBLE:

```
POS-VÆRDI :    8       4       2       1
NIBBLE    :    1       1       1       0
DECIMALT  :    8   +   4   +   2   +   0   =  14
```

Det DECIMALE tal 14 er det HEXADECIMALE $E. Næste NIBBLE:

```
POS-VÆRDI :    8       4       2       1
NIBBLE    :    1       0       1       1
DECIMALT  :    8   +   0   +   2   +   1   =  11
```

Tallet 11 DECIMALT svarer til $B HEXADECIMALT. Næste NIBBLE:

```
POS-VÆRDI :    8       4       2       1
NIBBLE    :    0       1       0       1
DECIMALT  :    0   +   4   +   0   +   1   =   5
```

Tallet 5 DECIMALT er også 5 HEXADECIMALT ($5).

Vort tal omregnes altså NIBBLE for NIBBLE og vi får at:

%1001 11101101 = $9EB5. (DECIMALT er tallet 40.629).

Når du har læst så langt, kan du sikkert se at det heller ikke er vanskeligt at gå fra HEXADECIMALT til BINÆRT. Det er bare at gøre det hele omvendt:

Lad os for nemheds skyld tage samme tal:

```
$9   =    %1001 (POS-VÆRDI 8 og 1)
$E   =    %1110 (POS-VÆRDI 8, 4 og 2)
$B   =    %1011 (POS-VÆRDI 8, 2 og 1)
$5   =    %0101 (POS-VÆRDI 4 og 1)
```

Derefter sætter man de fire NIBBLEr sammen til det BINÆRE tal: %1001 1110 1011 0101.

Let og ligetil, ikke sandt?

Når man programmerer i maskinkode på AMIGA - og for den sags skyld også på andre datamaskiner - så benytter man sig af en assembler. Det er et program, som lader dig programmere på en enkel måde, idet den oversætter det du skriver til ettaller og nuller. Dybest set er det instruktioner, som fortæller de forskellige processorer, hvilke ledninger der skal tændes (være lig med 1) for at opnå en bestemt virkning. Dette vil blive forklaret meget bedre og grundigere i et senere kapitel.

Rent teoretisk kan man godt skrive maskinkode direkte til maskinen, men det vil være så godt som umuligt at holde styr på alle de enere og nuller, der skal til. Man kommer hurtigt op over 500.000 cifre, og at finde et manglende 1-tal eller nul - eller et som er fejlplaceret - er så godt som umuligt.

I en assembler, som man bruger til AMIGA, skriver man følgende for f. eks at addere $D til et registret a0 (meget mere om dette senere):

```
add.l  #$0D,a0
```

Skrevet med enere og nuller vil det være:

11010001111111000000000000000000000000001101

Det er let at se, hvad der er lettest....!

OPGAVE 0103:
Omregn følgende tal til BINÆRT eller HEXADECIMALT:
%0000001001101000, %1010101111001101, $ABCD, $FFFF, $D5C6.

HVAD ER EN ASSEMBLER ?

For at kunne programmere i maskinkode på AMIGA skal du bruge en assembler. Det er et program, som oversætter dine instruktioner til maskinkode: et-taller og nuller. Du skriver korte kommandoer (f.eks. MOVE, CMP eller ADD) og assembleren oversætter dette. Selve oversættelses-processen kaldes at assemble.

I disse breve vil vi bruge en assembler som hedder K-SEKA. Hvis du ikke allerede har denne assembler, anbefaler vi dig stærkt at anskaffe den. Hvis du ønsker det kan du købe den hos os for 760,- kr. Vejledende udsalgspris (oktober 1989) er 848,- kr.

Lad os begynde med at forklare princippet for et assemblingsprogram som K-SEKA. Når du skriver dine instruktioner, så lagrer K-SEKA disse i AMIGAens hukommelse. Når du synes dit program er klar til at blive testet, så beder du assembleren (K-SEKA) om at tage dine instruktioner og oversætte dem til enere og nuller.

De instruktioner du skriver kaldes for din kildekode (vi vil bruge det engelske udtryk "SOURCE KODE", udtales sårs koud). Selve oversættelsen kaldes for at ASSEMBLE. De et-taller og nuller som K-SEKA producerer (og lagrer et andet sted i AMIGAs hukommelse) kaldes objektkode (engelsk: OBJECT CODE, udtales object koud).

Hvis du allerede har programmet har du sikkert også brugsanvisningen. For de af vore studerende, der lige har købt programmet, vil vi her gennemgå hvordan K-SEKA startes - og hvordan man skriver, assembler og prøvekører et lille programeksempel.

Først starter du Amiga på normal måde med Workbench. Derefter dobbelt-klikker du på icon'en mærket CLI.

Når AMIGADOS viser: 1>, så skriver du seka og trykker på RETURN. Når seka er indlæst, spørger maskinen, hvor mange kilobytes (Kb - 1 KB = 1024 BYTES) du vil afsætte til arbejdshukommelse. 150 Kb er som regel rigeligt, så skriv: 150 og tryk på RETURN. Nu er du (og K-SEKA) klar til at programmere i maskinkode.

Tryk på ESC-tasten. Du kommer nu ind i K-SEKAs tekst-editor (en separat del af K-SEKA). Det er her du skal skrive dine programmer. Vi viser nu et eksempel du kan skrive ind. Om du bruger store eller små bogstaver spiller ingen rolle:
(eksemplet står på næste side)

```
move.w          #$4000,$DFF09A
move.w          #$03A0,$DFF096
loop:

move.w          $DFF006,$DFF180     Programeksempel MC0101

btst            #6,$BFE001
bne.s           loop
move.w          #$83A0,$DFF096
move.w          #$C000,$DFF09A
rts
```

Tryk igen på ESC-tasten for at komme ud af tekst-editoren.

I øjeblikket skal du ikke bekymre dig om, hvorfor programmet gør det, det gør. Skriv det ind nøjagtigt som det står og følg så instruktionerne her under.

Efter vi har skrevet et program i K-SEKAs editor skal det assembles før det kan køres (startes). Dette gøres med kommandoen "a". Skriv "a" (uden anførselstegn) og tryk på RETURN. K-SEKA skriver nu ordet OPTIONS>

Dermed menes at du skal angive, hvordan dit program skal assembles. Du kan vælge at få det skrevet ud på printer eller på skærm. Du taster "v" + RETURN for skærmudskrift eller "p" + RETURN for udskrift på printeren. Skriver du "vh", vises assemblingen på skærmen side for side. Sideskift foregår med et tryk på SPACE (mellemrumstasten).

Hvis du ikke vil have en udskrift af programmet trykker du på RETURN. Hvis du vælger "p" (for printer) skriver maskinen: NAME> Du kan så skrive en titel - et navn - på dit program. Dette navn vil blive skrevet øverst på printerarket (engelsk: HARDCOPY).

Hvis dit program har en fejl når det assembles, vil K-SEKA afbryde og fortælle dig i hvilken linje fejlen er. Når du har rettet linjen, assembler du en gang til. Når programmet er fejlfrit, vil K-SEKA skrive "No errors" (dansk: ingen fejl).

Nu kan programmet lagres på diskette. Med "w" - kommandoen (w=write) lagrer du i SOURCE CODE (kildekode); mens "wo" - kommandoen (wo=write object) lagrer i OBJECT CODE. K-SEKA vil også bede dig om et et navn til din OBJECT CODE eller SOURCE CODE.

Efter programmet er lagret kan det køres. Det gøres med kommandoen "j".

OBS! VENT MED AT KØRE PROGRAMMET TIL DISKETTEDREVET ER STOPPET - Hvis du starter programmet før diskettestationens lampe er slukket, kan du få fejl på disketten, og det kan blive umuligt at læse fra den senere.

Når du vil standse programeksemplet, trykker du på venstre mus-knap.

Du indlæser et program i maskinens arbejdshukommelse fra disketten med kommandoen "r" (r=read). Hvis du har et andet program liggende i arbejdshukommelsen skal du slette dette først, da det nye ellers vil blive lagt til det gamle. Du sletter et program i SOURCE CODE ved at skrive "ks" og trykke på RETURN. Når K-SEKA spørger: "Are you sure?" taster du "y" (yes) hvis det er det du vil, eller "n" (no), hvis du alligevel ikke ønsker at fjerne den kildekode du har i maskinen.

Når du vil ud af K-SEKA, skriver du "!" og trykker på RETURN. Du vil så blive spurgt om du er sikker (du kunne jo f.eks have glemt at lagre et program). Svar "y" hvis du vil ud af programmet og "n" hvis du har ombestemt dig.

Du har nu lært de mest grundlæggende kommandoer i K-SEKA. Læs videre i programmets manual (brugsanvisning) og eksperimenter selv. Efterhånden, som vi i kurset får brug for flere kommandoer for at kunne programmere effektivt, vil vi tage fat på disse.

CHIP-HUKOMMELSE OG FAST-(HURTIG-) HUKOMMELSE

Her er en lille tabel, som viser hvor meget hukommelse AMIGA-maskinerne har. Forkortelsen "Kb" betyder som nævnt kilobytes og modsvarer 1024 BYTES. Læg mærke til denne forskel på vort dagliglivs kilo, som betyder 1000 og dataverdenens kilo som altså betyder 1024 (2 i 10ende potens).

MASKINE	CHIP	FAST (HURTIG)
AMIGA 500	512 Kb	0 (Kan udvides til 8,5 Mb)
AMIGA 1000	512 Kb	0 (Kan udvides til 8 Mb)
AMIGA 2000	1024 Kb	0 (kan udvides til 8 Mb)

Mb = megabyte, 1 Mb = 1.048.576 BYTES (2 i 20ende potens).

Måske må vi slå en ting fast med det samme. Når man taler om adresser i AMIGA, skyldes det, at AMIGAens hukommelse kan betragtes som en lang række af tomme kasser (BYTES). Man kalder så den første BYTE for 0, den næste for 1 og således nummereres de fortløbende til "rækken af kasser" slutter. På en måde får den sjette BYTE adressen $5 (BYTEs 0, 1, 2, 3, 4 og 5). AMIGAens adresser angives altid HEXADECIMALT og helst med 6 cifre: $000005.

CHIP-hukommelsen er de første 512 Kb af AMIGAens hukommelse. Den kaldes sådan fordi den bruges af special-CHIP'sene. Denne del af hukommelsen har alle processorerne (altså også special-CHIPs) adgang til - selvom de af og til må vente på hinanden. De specielle processorer AGNUS (bl.a. diskette og hukommelseskopiering), PAULA (bl.a. Keyboard, joystick, lyd) og DENISE (bl.a.bitplane, skærmopdatering) kan altså kun bruge den hukommelse, som leveres med maskinen fra begyndelsen.

Udvider du med mere hukommelsesplads, kaldes det ofte FAST-hukommelse. Betegnelsen FAST (hurtig) skyldes, at den ekstra hukommelse ligger på så høje adresser, at det kun er selve hovedprocessoren (68000) der kan bruge den. Derved slipper 68000 på at vente på andre processorer og adgangen til hukommelsen bliver dermed FAST (hurtig).

Vær opmærksom på, at det ikke kun er selve databehandlingen der foregår uhindret, men også læsningen af næste instruktion i det program AMIGAen udfører - forudsat programmet er lagt i FAST-hukommelsen. Dette medfører at et program i FAST-hukommelsen ofte udføres hurtigere, end hvis det befandt sig i CHIP-hukommelsen.

Herefter vil vi betegne FAST-hukommelse og CHIP-hukommelse som henholdsvis FAST-RAM og CHIP-RAM.

Bogstaverne RAM kommer fra det engelske udtryk RANDOM ACCESS MEMORY og betyder frit oversat "hukommelse hvortil der kan skrives og hvorfra der kan læses". Dette i modsætning til den del af hukommelsen der kaldes ROM (READ ONLY MEMORY) - hvorfra der kun kan læses. Dine programmer bliver altså liggende i RAM, mens f.eks. KICKSTART i AMIGA 500 og AMIGA 2000 ligger i ROM. Som du sikkert ved er KICKSTART navnet på AMIGAs operativsystem.

Når det gælder AMIGA 1000 og dens KICKSTART, er sagen noget anderledes. Her læses KICKSTART (256 Kb) ind i RAM fra diskette, hvorefter den "skrivebeskyttes" så den kun kan læses. På den måde ændres en del af hukommelsen til ROM. Når du slukker for AMIGA 1000, forsvinder KICKSTART.

KICKSTART - hukommelsen er at betragte som en FAST-hukommelse. Grunden er den, at den del af hukommelsen, der benyttes til KICKSTART, ligger på så høje adresser at de specielle grafikprocessorer ikke har adgang til dem.

HUKOMMELSESKONFIGURATION (Adresseområder):

MASKINE	CHIP-RAM	FAST-RAM	
AMIGA 500	$000000-$07FFFF	$200000-$9FFFFF	*)
AMIGA 1000	$000000-$07FFFF	$200000-$9FFFFF	
AMIGA 2000	$000000-$07FFFF	$200000-$9FFFFF	

*) Ekstrahukommelsen (A501) på undersiden af AMIGA 500 ligger på adresserne $C00000-$C7FFFF, og er derfor FAST-RAM. Dette fordi den havner udenfor det første 512 Kb område (CHIP-RAM), som strækker sig fra $000000-$07FFFF.

Læg igen mærke til, at hukommelsesadresser angives med 6 cifre, også selvom det medfører, at man må sætte nogle nuller foran adressen.

KORT EDB-ORDBOG

ASSEMBLER	Program, som oversætter dine specielle maskinkodeinstruktioner til et-taller og nuller.
BINÆRT	Talsystem med basen 2. I det daglige bruges basen 10.
BIT	Den enkelte bestandel i en BYTE.
BYTE	En dataenhed som indeholder 8 BITs.
CHIP-RAM	Hukommelse som bruges af både MC-68000 og AMIGAs co-processorer AGNUS, PAULA og DENISE.
CO-PROCESSOR	Hjælpe-processorer indbygget i AMIGAen for at udføre specielle funktioner og dermed øge hastigheden på grafik og programmer.
CUSTOM-CHIP	De specielle AMIGA co-processorer: AGNUS, PAULA og DENISE.
DOS	Betyder "Disk Operativ System". Det program, der styrer diskettestationen.
FAST-RAM	Hukommelse som kun kan bruges af hoved-processoren MC-68000.
HARDWARE	De "hårde" dele af AMIGA, eller andre computere: f.eks skærm, keyboard og processorer.
HEXADECIMALT	Talsystem med basen 16. Se BINÆRT.
INTERRUPT	"Afbryd". Hovedprocessoren får besked på at afbryde det den er i gang med og udføre noget andet.
K-SEKA	Den assembler dette kursus benytter.
NIBBLE	En halv BYTE, altså 4 BITs.
RAM	Den del af hukommelsen du kan skrive til (lagre i) og læse fra.
ROM	Den del af hukommelsen du kun kan læse i. Informationerne er lagret en gang for alle af fabrikanten.
SOFTWARE	Betegnelse for de progammer du læser ind i AMIGAen. F.eks. WORKBENCH.

KOMMENTARER TIL BREV I

Du har nu læst BREV I og kan det forhåbentlig på fingrene. Det er et relativt let brev du er gået igennem. Men vi advarer stærkt mod at undervurdere informationen i dette brev. Hvis du ikke lærer dette grundlag ordentligt, vil du få problemer når det senere bliver vanskeligere.

Opgaverne her i BREV I kan du, som nævnt tidligere, sende ind til os og få rettet. Det er dog ikke nødvendigt, fordi løsningerne altid vil findes i næste brev. Så i BREV II finder du opgaveløsningerne til opgaverne i dette brev.

Alle breve sendes til dig pr efterkrav medmindre du forud-betaler næste brev på vedlagte girokort senest den 10. i måneden. Herved sparer du efterkravsgebyret og portoen og på den måde bliver kurset væsentligt billigere for dig.

Desværre kan man ikke købe f.eks. brev nr 8 alene. Man skal først have købt alle de foregående breve. Når kurset er slut og du har gennemgået alle brevene, kan du, hvis du ønsker det, få tilsendt en prøve som tester hvor meget du har lært. Denne prøve returnerer du til os, og vi vil så rette den. Derefter modtager du et diplom som bekræfter, at du har gennemgået kurset og har bestået prøven.

Du er meget velkommen til at komme med kommentarer til vort brevkursus. Finder du et afsnit for dårligt forklaret, eller synes du, at der mangler noget i brevene vil vi blive meget glade over at høre fra dig. Som tak vil vi tage dit navn med i en liste i dette kursus over personer, der har været os til hjælp.

Til sidst er der kun at sige: GIV IKKE OP! Man behøver ikke være superintelligent for at gennemføre kurset. Det er nok med sund fornuft og lidt arbejde.

Fortsat god fornøjelse.

Med venlig hilsen
DATASKOLEN

Carsten Nordenhof