Programmering i maskinkode på *AMIGA*

A.Forness & N.A.Holten

Hæfte 7

Indhold

DATASKOLEN
Postboks 62
Nordengen 18
2980 Kokkedal

Telefon 49 18 00 77

Postgiro 7 24 23 44

BLITTER III

I dette kapitel skal vi se nærmere på SCROLLning. Ordet "SCROLL" udtales "skroll" og betyder "rulle".

I programeksemplerne MC0701 og MC0702 vil vi demonstrere SCROLLning af en tekst. Lad os først forklare hvordan det virker.

Et tegnsæt som indeholder alfabetet plus en del andre tegn (punktum, komma, parenteser osv.), kaldes ofte for en FONT (dansk: tegnsæt). En FONT er nødvendig for at kunne lave en SCROLL - eller med andre ord: Du skal bruge en færdig opsætning af bogstaver - et alfabet - for at kunne lave en SCROLL.

For at kunne SCROLLe (rulle) for eksempel ordet "AMIGA" over skærmen skal følgende gøres:

Sæt bogstavet "A" (fra din egen eller vores færdige FONT) ved højre skærmkant. Læg mærke til at vi skal sætte det ind fra højre på skærmen. Begynder du fra venstre - som naturligvis også kan lade sig gøre rent teknisk, kan du ikke læse ordet før alle bogstaverne er SCROLLet ind på skærmen. Det er svært at læse tekst, som SCROLLes fra venstre. Fra højre kommer bogstaverne i den rækkefølge som de læses.

SCROLL (flyt) derefter hele skærmen (alle skærmdata) 1 PIXEL (BIT) til venstre. Dette repeteres lige så mange gange som bogstaverne er brede - f.eks 16 gange, hvis "A" er 16 PIXELs bred.

Dermed er der igen kommet en ledig plads ved højre skærmkant igen, og vi kan gå igang med næste bogstav.

Således fortsætter man bogstav for bogstav indtil hele teksten er forsvundet ud i venstre skærmkant. På den måde får man en jævn SCROLLning af teksten hen over skærmen.

Du kan forestille dig det som en slidske, hvor man lægger et bogstav øverst. Når det er gledet et stykke, lægges næste bogstav i. Når et bogstav når enden af slidsken, skal det fjernes således at bogstaverne ikke hober sig op og renden fyldes op.

Det var altså selve teknikken en smule forenklet. Før vi fortsætter med SCROLLning, skal vi se på et andet begreb i dataverdenen, nemlig ASCII. I forbindelse med datamaskiner er det et meget kendt udtryk, og det udtales på dansk som "ASKI". Når du trykker en tast ned på tastaturet, bliver det registreret og forvandlet til en såkaldt ASCII-kode inde i AMIGAens operativsystem.

ASCII er en forkortelse for "AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE". Det udtales: "amerikæn standard Coud får informeisjen intertjeinsh", og kan vel bedst oversættes med: Amerikansk standard for udveksling af information.

Lad os vise nogle eksempler. Et (stort) "A" har værdien (ASCII-koden) 65 (DECIMALT). Mellemrums-tasten har værdien (ASCII-koden) 32. Se også i tabellen bagest i brevet som indeholder alle ASCII- koder for et komplet alfabet.

FONTen, som bruges i programeksemplerne, indeholder ikke alle ASCII-tegn. Istedet for at sætte de tegn, som ikke er med i vores FONT, som blanke tegn (mellemrum), har vi lavet en anden rækkefølge af tegnene i FONTen i forhold til ASCII-koderne.

Her ser du hvordan tegnene i vores FONT er sat op:

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

[	\	]	,	.	
26	27	28	29	30	31

Rækkefølgen og værdierne er forandrede i forhold til standard (eller sædvanlig) ASCII-kode, så vi skal konvertere (forandre) ASCII-koderne til de værdier vi benytter os af i vores FONT. Bogstavet "C" er i 67 ASCII-kode, og det skal ændres til 3, som er "C" i vores FONT.

Vi begynder nu på programeksempel MC0701. Af pladshensyn er det ikke trykt i brevet. Du finder det på kursusdiskette #1 under DIRECTORY BREV07.

Linie 1:	Sluk alle INTERRUPTS.
Linie 3-4:	Standser diskette-motoren.
Linie 6:	Slukker BITPLANE, COPPER og SPRITE-DMA'erne.
Linie 8-17:	Opsætter en 256 * 352 PIXELs skærm med et BITPLANE og med 2 i MODULO. Hvad en MODULO er og hvordan den fungerer forklares senere i dette brev.
Linie 19-25:	Lægger skærmadressen ind i COPPER-listen.

Linie 27-28:	Lægger COPPER-listens adresse ind i COPPER-pointeren.
Linie 29:	Starter BITPLANE og COPPER-DMA'erne igen.
Linie 31:	Her begynder hoved-LOOPen.
Linie 32-36:	Venter til elektronstrålen er nået til skærmlinie nummer 300.
Linie 38:	Hopper til rutinen (SUBRUTINEN eller underprogrammet) "scroll".
Linie 40-41:	Checker om venstre mus-tast er trykket ned. Hvis ikke, hopper programmet tilbage til rutinen "mainloop".
Linie 43:	Slukker COPPER-DMAen.
Linie 45-47:	Læg adressen på den gamle COPPER-liste ind i COPPER-pointeren.
Linie 49:	Starter COPPER- og SPRITE-DMAerne.
Linie 51:	Tænder alle INTERRRUPTs igen.
Linie 54-58:	Her har vi defineret to WORDs, som skal bruges til tællere i forbindelse med SCROLLningen.
Linie 60:	Her begynder rutinen som tager sig af selve SCROLLningen.
Linie 61:	Lægger adressen på "scrollcnt" ind i A1.
Linie 62:	Sammenligner "scrollcnt" med værdien 8.
Linie 63:	Hvis ikke "scrollcnt" er 8, hoppes der til "nochar".
Linie 65:	Sætter "scrollcnt" til 0.
Linie 67:	Lægger adressen på "charcnt" ind i A1.
Linie 68:	Lægger værdien fra den adresse, som A1 peger på ind i D1. Altså, lægger værdien af "charcnt" ind i D1.
Linie 69:	Lægger 1 til i "charcnt". Læg mærke til, at den lægger 1 til direkte i det deklarerede WORD, og ikke i D1!.
Linie 71:	Lægger adressen på "text" ind i A2.
Linie 72:	Sætter D2 til 0.

Linie 73: Lægger værdien som A2 + D1 peger på ind i D2. Denne instruktion henter en ASCII-kode fra din egen tekst ("text"), hvor OFFSET til teksten befinder sig i D1. Altså, hvis D1 indeholder 0, vil første bogstav blive lagt i D2. Hvis D1 indeholder 1, vil den hente andet bogstav, osv.

Linie 75: Checker om D2 er lig 42. ASCII-koden 42 er en "*". Denne "*" skal være sidste tegn i teksten for at indikere at teksten er slut.

Linie 76: Hvis ikke vi er nået til sidste tegn, så hop til "notend".

Linie 78-79: Hvis det var sidste tegn i teksten, så sættes "charcnt" til 0, og der lægges et mellemrum ind i D2 (ASCII-koden for mellemrum er 32).

Linie 82: Lægger adressen til "convtab" ind i A1. Denne tabel indeholder bogstavernes (tegnenes) position i FONTen.

Linie 83: Lægger værdien som A1 + D2 peger på, ind i D2. Den gamle værdi i D2 (som indeholdt ASCII-koden for bogstavet) bruges her som en offset i tabellen.

Altså: Hvis D2 indeholder 32 (et mellemrum) hentes den 33te (husk at der regnes fra 0) BYTE i den tabel, som indeholder vores FONT's position. I dette tilfælde $1F eller 31 DECIMALT. Vi har nu lavet ASCII-koden om til vores egne FONT-værdier, og kun brugt en eneste instruktion!

Linie 84: Indholdet i D2 roteres en BIT til venstre. Dette er det samme som en multiplikation med 2. Det skal gøres fordi et tegn i FONTen er 16 PIXELs (2 BYTES) bredt.

Linie 86: Lægger adressen på "font" ind i A1.

Linie 87: Adderer værdien i D2 til A1. Nu peger A1 på adressen på det første bogstav, som skal lægges ind på skærmen.

Linie 89: Lægger skærmadressen (screen") ind i A2.

Linie 90: Adderer 6944 til A2. Som nævnt er skærmen 352 PIXELs bred (inklusive 16 PIXELs OVERSCAN på hver side) og har 2 i MODULO. Altså bliver en skærmlinie 46 BYTEs. (352/8 + 2). MODULO virker på BLITTERen på samme måde som på BITPLANEs. Fordi den synlige skærmbredde er sat til 352 PIXELs (det samme som 44 BYTEs), bliver der to BYTEs til overs pr. skærmlinie. Når vi definerer

2 i MODULO, vil BITPLANE-DMAen hoppe over 2 BYTE i slutningen af hver optegnet linie. Dette resulterer i at vi får 2 BYTEs med usynlige skærmdata på hver linie. Det er i dette område, vi lægger bogstaverne, som skal SCROLLes. Når vi gør det på denne måde, ser det ud som om bogstaverne kommer ind udenfor skærmen.

Altså: Når vi lægger 6944 til, havner vi på linie 150, og BYTE-position 44 (150*46+44 = 6944).

Linie 92: Lægger værdien 19 ind i D0. D0 bruges som en tæller for at tælle højden på det bogstav (tegn) som skal lægges ind på skærmen (skærmhukommelsen). I den FONT vi bruger her er bogstaverne 20 PIXELs høje.

Linie 94: Her begynder LOOPen som skal lægge tegnet fra FONTen ind i skærmhukommelsen.

Linie 95: Lægger værdien som A1 peger på, ind i adressen som A2 peger på.

Linie 96: Adderer 64 til adresseregister A1. Nu vil A1 pege på næste linie i FONTen (32 tegn * 16 PIXELs = 512. 512/8 = 64).

Linie 97: Adderer 46 til A2. A2 peger nu på næste linie i vores skærmhukommelse.

Linie 98: Trækker 1 fra D0 og checker om D0 er -1. Hvis ikke, hoppes der tilbage til rutinen, som har LABELen "putcharloop".

Linie 101-102: Venter til BLITTERen er ledig.

Linie 104: Lægger adressen på skærmen ("screen") ind i A1.

Linie 105: Adderer 7820 til A1. A1 vil nu pege på nederste linie, som skal SCROLLes (46*(150+20) = 7820). Det er jo almindeligt at pege på øverste venstre hjørne på BLITen, men i dette tilfælde skal vi pege på nederste (sidste) linie, som skal BLITtes. Dette fordi vi kører BLITTERen i DESCENDING-MODUS (altså at BLITTERen køres "baglæns" således at den tæller nedefter og ikke op i hukommelsen). Hvis vi havde ladet BLITTERen køre i sædvanlig MODUS ville vores tekst have rullet den forkerte vej.)

Linie 107: Lægger værdien, som ligger i A1 ind i BLITTERens A-pointer.

Linie 108: Lægger samme værdi ind i D-pointeren.

Linie 109: Sætter A-MODULO til 0.

Linie 110: Sætter også D-MODULO til 0.

Linie 111: Sætter A-MASK til $FFFFFFFF.

Linie 112: Sætter den logiske operation D=A, og angiver 2 BITs skift på A-kanalen. Det er denne værdi, som indeholder hvor mange PIXELs, der skal SCROLLes ad gangen. I dette tilfælde SCROLLer vi PIXELs to og to. Hvis du ønsker, at der skal SCROLLes langsommere, kan du sætte denne til 1, og ændre tallet 8 i programlinie 62 til 16.

Linie 113: BIT 1 i BLTCON1-registret sættes til "1" for at angive DESCENDING-MODUS.

Linie 114: Værdien $5017 lægges ind i BLTSIZE (bredde = 23 WORDs og højde = 20 linier) Værdien regnes ud således: 20*64+23=$5017). Når denne instruktion er udført, starter BLITTERen automatisk.

Linie 116: Lægger adressen på "scrollcnt" ind i A1.

Linie 117: Adderer 1 til "scrollcnt".

Linie 119: Hopper tilbage til programlinie 38, og fortsætter programudførelsen derfra.

Linie 122-135: Her er vores COPPER-liste deklareret.

Linie 138: Her har vi defineret skærmhukommelsen.

Linie 141: Her ligger BUFFERen til FONTen.

Linie 144 237: Her ligger tabellen som bruges til at konvertere (forandre) fra ASCII-kode til tegn-position i vores FONT.

Linie 240: Her ligger teksten som skal SCROLLes. Du kan selvfølgelig skrive din egen tekst ind. Læg mærke til at vi har brugt specielle tegn for Æ, Ø og Å. Det har vi gjort fordi det kan være vanskeligt at få fat i disse bogstaver på alle tastaturer.

Altså, når du skal bruge et "Æ" skriver du "@", vil du have et "Ø" bruger du "#" og "Å" er et "$"-tegn. Glem ikke at skrive et "*"-tegn til sidst.

For at kunne køre dette program skal du først ASSEMBLE det.

Derefter lægger du filen "FONT" ind på LABELen "FONT" således:
(se næste side)

```
SEKA>a
OPTIONS>
No errors
SEKA>ri
FILNAME>font
BEGIN>font
END>  (tryk bare på RETURN)
SEKA>j
```

BLITTER IV

Vores næste programeksempel - MC0702 - er helt magen til det første (MC0701) bortset fra, at vi har lagt en såkaldt COLORCYCLING til - eller rettere sagt: COPPERCYCLING.

Denne effekt laver et farvemønster inde i teksten (i selve bogstaverne eller FONTen) som SCROLLes. Dette gøres ved at få COPPERen til at vente på en speciel skærmlinie, for så at sætte tekstfarven til for eksempel rød.

Derefter skal COPPERen vente på næste skærmlinie for at sætte en ny farve ind i registret, som benyttes til at farvelægge selve teksten.

Dette gentages 20 gange (teksten eller FONTen er jo 20 PIXELs høj). På denne måde kan vi få nydelige farvemønstre på vores tekst.

Farveskiftet sker ved at opdatere COPPER-listen hele tiden og så kombinere dette med en forskydning af farvetabellen, som lægges ind.

Vi gennemgår nu CYCLE-rutinen for programeksempel MC0702. Resten af programmet er magen til MC0701 så det gennemgår vi ikke.

Linie 40: Denne instruktion er føjet til hovedrutinen for at opdatere CYCLINGen.

Linie 124: Her har vi deklareret et WORD, som vi bruger som tæller i forbindelse med CYCLE-rutinen.

Linie 126: Her begynder CYCLE-rutinen.

Linie 127: Lægger adressen på "cyclecnt" ind i A1.

Linie 128: Lægger værdien som ligger i "cyclecnt" ind i D1.

Linie 130: Adderer værdien 2 til "cyclecnt".

Linie 132: Sammenligner D1 med værdien 96.

Linie 133: Hvis D1 er forskellig fra 96, så hoppes der til rutinen "notround".

Linie 135: Lægger værdien 0 (CLR) ind i "cyclecnt".

Linie 136: Lægger værdien 0 ind i D1.

Linie 139: Lægger adressen på "cycletable" ind i A1. Rutinen mærket "cycletable" er den tabel, som indeholder de farvedata, der skal rulle i teksten.

Linie 140: Lægger adressen på "cyclecop" ind i A3. Dette er begyndelsen på de COPPER-instruktioner, som håndterer CYCLINGen. Læs COPPER-listen (linie 153-208) for at forstå hvordan de følgende programlinier fungerer:

Linie 142: Lægger værdien 19 ind i D0, som her bruges som tæller.

Linie 144: Her starter LOOPen som lægger farvedataene ind i COPPER-listen.

Linie 145: Lægger værdien som A1+D1 peger på, ind i adressen som A3+6 peger på. Studer farvetabellen ("cycletable") og COPPER-listen nøje og læg mærke til, hvor værdierne hentes og lægges.

Linie 146: Adderer 2 til D1. Næste gang den udfører programlinie 145, vil den hente næste farvedata i tabellen. Vi lægger 2 til fordi hvert farvedata er et WORD langt.

Linie 147: Adderer 8 til A3. A3 vil nu pege på næste plads i COPPER-listen, hvor næste farvedata skal lægges.

Linie 148: Trækker 1 fra D0. Checker så om D0 er -1. Hvis det ikke er tilfældet, så hoppes der tilbage til "cycleloop".

Læg mærke til at denne LOOP (linie 145-148) udføres 20 gange.

Linie 150: Hopper tilbage til hovedrutinen.

Linie 203: Her ligger COPPER-instruktionerne som sætter farver på SCROLL-teksten (i selve bogstaverne)

Disse COPPER-instruktioner kan forklares således: Vent til elektronstrålen er nået til linie $C2 (194). Læg farve "?" (hvilken farve ved vi ikke fordi farven jo lægges ind når programmet køres) ind i COLOR 01 ($DFF182), som er vores tekstfarve. Vent til elektronstrålen er nået til linie $C3 (195)... og så videre og så videre. Dette gentages 20 gange i COPPER-listen.

Linie 313-322: Her har vi deklareret vore farvedata.

FONTS

I dette kapitel vil vi se lidt på FONTs. En FONT er en betegnelse for et tegnsæt - også et du har lavet selv. Som bekendt bruges FONTs blandt andet i SCROLLning.

I programmerne MC0701 og MC0702 bruger vi en FONT, som er 16 PIXELs bred og 20 PIXELs høj, men det er helt i orden at lave en FONT, som både er mindre eller større. I vore programeksempler har vi kun brugt en BITPLANE i FONTen. Dette gør, at FONTen bliver ensfarvet, og at vi kun skal bruge et farveregister.

I programeksemplet MC0702 har vi kun en BITPLANE - men vi "pyntede" lidt på FONTen ved at skifte farver med COPPERen. Hvis du laver en FONT med flere farver, skal du selvfølgelig huske på at SCROLLe samtlige BITPLANES SYNKRONT (dvs med samme hastighed).

Lad os begynde med at beskrive hvordan du laver din egen FONT. Når du skal tegne din egen FONT kan du bruge f.eks Deluxe Paint, som er et meget udbredt tegneprogram til AMIGA). Før du begynder at tegne bør du begynde med at finde ud af, hvilke specialtegn du vil have med (punktum, komma, bindestreg, spørgsmåltegn osv..

Derefter må du ikke glemme at sætte en plads af til mellemrums-tegnet (som let kan glemmes). Det er jo bare en blank "firkant".

Når du er færdig med at tegne, skal du sætte bogstaverne i en passende orden. I vores FONT har vi 32 tegn (hele alfabetet plus komma, punktum og mellemrum). Vi valgte at sætte alle tegn på en horisontal række. Altså, "A" sættes øverst oppe i venstre hjørne på skærmen. "B" sættes 16 PIXELs til højre for "A", osv.

Billedstørrelsen vi brugte blev da: 32 (tegn) * 16 (PIXELs) = 512 PIXELs bred, og 20 PIXELs høj. Vi kan nu regne ud at FONTen optager (512/8) * 20 = 1280 BYTEs i hukommelsen.

Fordi vi satte alle tegnene på **en** række, behøver vi kun at kende x-positionen i FONTen, når den bliver konverteret fra ASCII-kode til "FONT-position". Hvis du vil tegne større bogstaver i din FONT og måske have flere tegn med, bliver det vanskeligt at få alle bogstaverne på en række. Du må da sætte flere rækker op under hinanden. Dette betyder at du skal bruge både x- og y-positionen når ASCII-koden bliver konverteret til "FONT-position" i programmet.

Et almindeligt problem når du tegner en FONT er, at nogle bogstaver er bredere end andre (f.eks er "W" bredere end "I").

Dette betyder at der kommer forskellig afstand mellem bogstaverne når de SCROLLes på skærmen (det ser ikke pænt ud).

En måde at undgå dette på, er at tegne alle bogstaverne lige brede. En anden og bedre (dvs pænere) metode er, at "trykke" dem sammen når de SCROLLes på skærmen. Dette kaldes PROPORTIONAL SPACING (oversat: proportionalskrift). Lad os vise det med et eksempel.

- Tegnene i FONTen er 16 PIXELs brede.

- Antag at tegnet "W" optager 14 PIXELs i bredden.

- Antag at tegnet "I" optager 8 PIXELs i bredden.

Først SCROLLes bogstaverne "WWWWW" over skærmen. Dette gøres rent teoretisk på følgende måde:

Sæt bogstavet "W" ind i højre skærmkant. SCROLL skærmen i alt 16 PIXELs mod venstre. Sæt så næste "W" ind, og SCROLL 16 PIXELs.

Således fortsætter du så længe du selv vil.

Når vi SCROLLede 16 PIXELs mellem hver "W" blev der 2 PIXELs mellemrum mellem bogstaverne (som er ganske passende).

Lad os nu SCROLLe bogstaverne "IIIII" over skærmen.

For at få "I"-erne til at se pæne ud når de læses, skal vi kun (når vores FONT bruges) SCROLLe 10 PIXELs mellem hver bogstav (8 PIXELs bredde plus 2 PIXELs mellemrum). Altså, sæt et "I" på skærmen, SCROLL 10 PIXELs, sæt næste "I" på skærmen osv.

Hvis vi nu vil SCROLLe bogstaverne "IW" over skærmen gør vi det på følgende måde når vi vil tage hensyn til proportionaliteten:

Sæt tegnet "I", SCROLL 10 PIXELs, sæt tegnet "W" og SCROLL 16 PIXELs.

Du ser altså at maskinen skal vide, hvor mange PIXELs hvert enkelt tegn (bogstav) skal SCROLLes. Disse værdier kan du lægge i tabellen, som bruges til at konvertere ASCII-koder, således at du får både positionen og bredden på bogstavet som konverteres. Prøv at lave din egen FONT og et program, som benytter sig af PROPORTIONAL SPACING.

Vi påstår ikke det er let, men det er fuldt ud muligt, hvis du prøver at holde lidt orden og en god oversigt i dit program.

Benyt mange delrutiner (SUB-rutiner). Det gør programmet mere struktureret og derved mere overskueligt.

MASKINKODE VI

Vi starter dette kapitel om maskinkode på AMIGA med at forklare, hvad en SIGNERET værdi er og hvordan den opfører sig, når du skal bruge sådanne tal i programmering.

Du har allerede hørt om USIGNEREDE og SIGNEREDE værdier, men hvad er det egentlig? Lad os sige at det drejer sig om en BYTE (8 BITs som sædvanlig). En BYTE som er USIGNERET (såkaldt USIGNERET BYTE) kan kun indeholde værdier fra 0 til 255. Den kan hverken betragtes som positiv eller negativ - den angiver kun et antal indenfor disse givne grænser.

En SIGNERET BYTE derimod, kan indeholde både positive og negative værdier. Den kan indeholde værdier fra -128 til 127, altså MINUS 128 til PLUS 127. (ialt 256 tegn).

Lad os først vise nogle eksempler på værdier (vi forklarer nærmere derefter):

BINÆRT	HEXADECIMALT	USIGNERET	SIGNERET
00000000	$00	0	
00010111	$17	23	
11111111	$FF	255	-1
11111110	$FE	254	-2
10000000	$80	128	-128
10000001	$81	129	-127
10000010	$82	130	-126
01111111	$7F	127	
01111110	$7E	126	

Du har sikkert forstået hvordan det virker allerede? Altså: BITen helt til venstre i BIT-gruppen som i dette tilfælde er BIT 7, bestemmer om værdien skal være POSITIV eller NEGATIV.

Hvis den er "0", vil værdien være POSITIV eller nul (0-127). Hvis den derimod er "1" (SAT eller ON), vil værdien være NEGATIV (-1 til -128). Det er måske ikke så let at forstå de syv andre BITs.

Lad os studere dette lidt nærmere.

Hvis værdien er POSITIV (BIT 7 er "0"), vil BIT 0-6 vise den aktuelle værdi direkte.

Hvis værdien er NEGATIV (BIT 7 er "1"), skal der laves et lille trick for at vi skal kunne se, hvilken værdi som repræsenteres.

Lad os tage værdien 11101001 (=233). Vi ser straks at den er NEGATIV, men ikke **hvilken** NEGATIV værdi den har (måske nogen erfarne programmører kan se det). Dette afsløres på følgende måde:

11101001 = 233

Først inverteres alle BITs ("1" bliver "0", og omvendt), vores BYTE ser derefter således ud:

00010110 = 22

Derefter lægger vi 1 til. BYTEn ser da således ud:

00010111 = 23

Vi har nu fået værdien 23. Altså, 11101001 er lig -23. Enkelt, ikke sandt? Man inverterer bare BITene og lægger 1 til. Lad os prøve at gå den anden vej:

00010111 = 23

Inverter...

11101000 = 232

Læg 1 til...

11101001 = 233

Som vi ser, blev værdien lig med den vi startede med.

Metoden med at invertere og lægge 1 til, kaldes for 2-kompliment. Alle datamaskiner bruger 2-kompliment.

Der findes også en anden metode for SIGNEREDE tal, som ikke bruges på datamaskiner, nemlig en såkaldt 1-kompliment. Lad os hurtigt se på denne metode også:

I 1-kompliment inverteres værdien, men der lægges ikke noget til.

Vi får da følgende:

00000001 = 1

Inverteret bliver det...

11111110 = -1

Et eksempel til måske?

00000000 = 0

inverteret bliver det...

11111111 = -0

Hvad skete der egentlig? Da vi inverterede 00000001, blev resultatet 11111110 som så må være -1. Så langt så godt.

Men hvad skete der når vi inverterede BITene 00000000? Jo, vi fik resultatet 11111111, som så skal være -0! Nu forstår du sikkert hvorfor datamaskiner ikke bruger 1-kompliment.

Vi holder os til 2-kompliment.

Vi har nu set at en SIGNERET BYTE kan indeholde værdier fra -128 til 127 (MINUS 128 til PLUS 127), men hvad kan et SIGNERET WORD indeholde?
Vi har lavet en lille tabel der illustrerer dette. Husk at det altid er BITen længst til VENSTRE I BIT-gruppen, som angiver om værdien skal betragtes som POSITIV eller NEGATIV (i et WORD bliver det altså BIT 15, som bestemmer fortegnet).

STØRRELSE	VÆRDIER
BYTE (8 BITs)	-128 til 127
WORD (16 BITs)	-32768 til 32767
LONGWORD (32 BITs)	-8388608 til 8388607

Vi har lavet et lille programeksempel som du kan eksperimentere med selv:

```
CLR.L D0
MOVE.B      #-50,D0
ADD.B       #70,D0
SUB.B#5,D0
.....

RTS
```

Eksperimenter også med at bruge instruktioner med længderne WORD og LONGWORD (".W" og ".L"), for at se hvad som sker med D0.

Du undrer dig sikkert over, hvordan maskinen ved om den skal behandle tal som SIGNEREDE eller USIGNEREDE. Instruktionerne ADD og SUB vil altid behandle tal/data som SIGNEREDE værdier. Reglen er at alle instruktioner, som udfører matematiske beregninger behandler værdierne som SIGNEREDE. To undtagelser er instruktionerne MULU og DIVU. Disse instruktioner betyder henholdsvis: MULTIPLY UNSIGNED (Multiplicer USIGNERET) og DIVIDE UNSIGNED (Divider USIGNERET). De SIGNEREDE versioner hedder MULS og DIVS. Vi vil komme tilbage med en mere indgående forklaring af MULU, MULS, DIVU og DIVS i et senere brev.

Altså, når det gælder multiplikation og division kan man vælge om man vil arbejde SIGNERET eller USIGNERET, mens addition og subtraktion altid arbejder SIGNERET.

Dette var enden på maskinkodeafsnittet i dette brev. Øv dig på dette indtil næste brev. Før eller siden får du helt sikkert brug får det i din programmering. Held og Lykke!

LØSNINGER TIL OPGAVER I BREV VI

Opgave 0601: Ordet MODULO er et vanskeligt ord at oversætte til dansk. En nogenlunde brugbar oversættelse er: TILLÆG, OVERSPRING eller FORSKYDNING.

Opgave 0602: En BLIT som har BLTSIZE $1A69 bliver 105 PIXELs (linier) høj og 656 PIXELs (BITs) bred.

Opgave 0603: Værdien som A indeholder bliver INVERTERET (logisk NOT, "0" bliver "1" og omvendt).

Opgave 0604: Det mindste antal PIXEL horisontalt som kan defineres i BLTSIZE er 16 (Husk! Et WORD, 16 BITs eller 16 PIXELs).

OPGAVER TIL BREV VII

Opgave 0701: Prøv at ændre SCROLL-hastigheden, farverne i farvetabellen, osv. Altså, Eksperimenter med SCROLL-programmet!

Opgave 0702: Hvad bliver disse BINÆRE tal (BYTE længde) DECIMALT, henholdsvis SIGNERET OG USIGNERET:

%01101001, %10111110, %11110001, %00101101, %10001011 og %11010011

Opgave 0703: Hvad er den højeste og den mindste værdi, en SIGNERET BIT-gruppe på 20 BITs kan indeholde ?

Opgave 0704: Prøv at lave et program, som laver en positiv værdi negativ (SIGNERET værdi) og omvendt (uden at bruge MULS, eller NEG, som også er en instruktion).

DATAORDBOG

SIGNERET	En BIT-gruppe som er SIGNERET indeholder både positive og negative værdier.
USIGNERET	En BIT-gruppe som er USIGNERET indeholder kun positive værdier - kan egentlig betragtes som et antal (værdien 0 er også medregnet her).
OFFSET	Angiver en afstand mellem to punkter i dit program (egentlig afstanden mellem to adresser i hukommelsen).
HAM	HOLD AND MODIFY (udtales: håld and moudifai). Betyder: Hold (stop) og modificer (f.eks. et farveregister).
MODULO	Tillæg eller overspring. Benyttes for at få korrekt afstand til næste data, som f.eks BLITTERen skal indlæse eller udskrive.
STACK	Kan oversættes med "stabel". Hukommelses område som bruges af datamaskiner til at lagre data temporært (midlertidigt).
INVERTERE	Vende om, oftest ændre 1 til 0 eller omvendt.
MASK	Betyder "maske". Bruges til at afdække en eller flere BITs i en BIT-gruppe (BYTE, WORD, LONGWORD) for at opnå en ønsket effekt.
SUBRUTINE	Underrutine. En lille (eller stor) programdel. Et del-program, som udfører en speciel opgave.

ASCII-KODE TABEL

TEGN	KODE	TEGN	KODE	TEGN	KODE
<SPACE>	32	J	74	t	116
!	33	K	75	u	117
"	34	L	76	v	118
#	35	M	77	w	119
$	36	N	78	x	120
%	37	O	79	y	121
&	38	P	80	z	122
'	39	Q	81	{	123
(	40	R	82	|	124
)	41	S	83	}	125
*	42	T	84	~	126
+	43	U	85		
,	44	V	86		
-	45	W	87		
.	46	X	88		
/	47	Y	89		
0	48	Z	90		
1	49	[	91		
2	50	\	92		
3	51	]	93		
4	52	^	94		
5	53	_	95		
6	54	`	96		
7	55	a	97		
8	56	b	98		
9	57	c	99		
:	58	d	100		
;	59	e	101		
<	60	f	102		
=	61	g	103		
>	62	h	104		
?	63	i	105		
@	64	j	106		
A	65	k	107		
B	66	l	108		
C	67	m	109		
D	68	n	110		
E	69	o	111		
F	70	p	112		
G	71	q	113		
H	72	r	114		
I	73	s	115		