

# dus med

## Commodore 64/plus-4

64



Kurt Friis Hansen



**RUN I KIOSKEN**

© C & B Mikrodatainformation ApS 1984  
Blåkildecetret 18, 2630 Tåstrup

Mekanisk, fotografisk eller anden gengivelse af denne bog eller dele af den uden forudgående aftale med forlaget er ikke tilladt ifølge gældende dansk lov om ophavsret.

Forfatter: Kurt Eriis Hansen  
Titel: Dus med Commodore 64/Plus 4  
Tryk: Tåstrup Lyntryk, Gartnervej 1, 2630 Tåstrup

ISBN Nr. 87-980926-4-2







Kapitel 1 Computeren i praksis	11
Kapitel 2 Commodore TASTATUR og SKÆRM	13
2.1 Tastaturet i detaljer	13
2.1.1 Auto-repeat på alle taster	14
2.1.2 Valg af alfabet	15
2.1.3 Udkobling af RUN/RESTORE m.m.	15
2.2 Læsning af tastaturet fra BASIC	16
2.3 Dit helt personlige tastatur	17
2.3.1 Brugerdefinerede tegn	17
2.3.2 Brugerdefineret tastatur	22
2.4 Billedskærmen i Commodore 64 BASIC	24
2.5 Man kan mere end blot skrive til skærmen	25
2.5.1 Hvad er et device	25
2.5.2 Åbning og lukning til devices	26
2.5.3 Læsning af skærmens indhold	27
2.5.4 Avanceret INPUT i BASIC	29
2.5.5 Farver og andre effekter med INPUT	30
Kapitel 3 Commodore BASIC	32
3.1 Om RAM og ROM	32
3.2 Tal og beregninger	33
3.3 Variabler	34
3.4 Arrays - en særlig form for variabler	36
3.4.1 Arrays og hukommelse	37
3.5 Matematiske udtryk	38
3.6 Sammenligninger og forgreninger	39
3.7 Logiske operatorer	40
3.8 Binære operatorer	40
3.9 Betingede hop på en anden måde	41
3.10 Løkker	43
3.11 Sådan laver du hurtigere programmer	44
Kapitel 4 Programmering i maskinkode	49
4.1 Hjælpemidler i programmeringen	50
4.2 Om talsystemer	51
4.2.1 Det binære talsystem	51
4.2.2 Hexadecimale tal	54
4.2.3 Hexadecimale tal - hvorfor?	55
4.3 Maskinkode og BASIC	56
4.3.1 Overførsel af variabler til maskinkode-rutine	57
4.3.2 Matematiske beregninger i maskinkode	58
4.3.3 Nyttige maskinkode rutiner i Commodores ROM	60

4.3.4 Ind- og udlæsning af data i maskinkode	60
4.3.4.1 Åbning af en kanal i maskinkode	61
4.3.4.2 Læsning fra en kanal i maskinkode	61
4.3.4.3 Skrivning til en kanal i maskinkode	62
4.3.4.4 Lukning af en kanal i maskinkode	63
4.3.4.5 LOAD og VERIFY i maskinkode	63
4.3.4.6 LOAD program under BASIC ROM	64
4.3.4.7 SAVE i maskinkode	65
4.4 Computerens hukommelse	66
4.4.1 APPEND af BASIC programmer	68
4.4.2 Placering af maskinkode programmer	69
4.4.3 Sådan lagres BASIC programmer i hukommelsen	70
4.4.4 BASIC-program laver BASIC-program	72
4.4.5 Relokerbar maskinkode loader	74
Kapitel 5 Commodore PLUS/4 og maskinkode	78
5.1 Beskrivelse af de enkelte monitor kommandoer	78
Kapitel 6 Programmer på kassette	83
6.1 Kasettebåndoptageren og data	83
6.2 Kasettebåndoptagerens optageformat	84
6.3 STATUS variabel og kassette	86
Kapitel 7 Commodore og disk	89
7.1 Commodore 1541 - specifikationer	90
7.2 Devicenumre og disk	90
7.2.1 Kommunikationen mellem forskellige enheder	91
7.2.2 Parallel kommunikation	92
7.2.3 Seriel kommunikation	92
7.3 Disketter som lagermedie	93
7.3.1 Disketten inddeles i spor	94
7.3.2 Diskette-typer	95
7.3.3 Formattering af disketter	95
7.4 Vær god ved dine disketter	96
Kapitel 8 Lagring af programmer på diskette	98
8.1 Directory og filer	98
8.2 Læsning af diskettens indhold	99
8.3 Brug af WILDCARDS eller JOKERE	100
8.3.1 Forslag til navngivning af filer	101
8.4 Filtyper	102
8.4.1 Wildcards og filtyper	103
8.5 Programfiler	104
8.5.1 SAVE til diskette	104

8.5.2 VERIFY af lagret program	105
8.5.3 LOAD fra diskette	106
8.6 Læsning af filer	107
8.6.1 Kanaler for data-overførsel	107
8.6.2 Åbning af en disk-kanal	107
8.6.2.1 Åbning af fil med replace	109
8.6.3 Skrivning til disk	109
8.6.3.1 Listning til disk	110
8.6.4 Læsning fra disk	111
8.6.4.1 INPUT# kommandoen og disk	111
8.6.4.2 GET# kommandoen og disk	113
8.6.4.3 GET# kommando - læsning af directory	114
8.6.5 STATUS-variablen og disk	117
8.6.6 Lukning af en disk-kanal	119
8.7 Læsning af fejlkanalen	120
8.7.1 Læsning af fejlkanalen på Commodore 64	120
8.7.2 Læsning af fejlkanalen på Commodore Plus/4	121
8.8 Specialkommandoer for disk	121
8.8.1 Formattering af disketter	122
8.8.2 Initialisering af BAM	123
8.8.3 Opdatering af BAM	123
8.8.4 Sletning af filer	123
8.8.5 Ændring af filnavnet	124
8.8.6 Kopiering af filer	124
8.8.7 Sikkerhedskopiering af disketter	125
8.9 Commodore Plus/4 og C16 disk-kommandoer	135
8.10 Sekventielle filer	136
8.10.1 Sekventiel læsning af programfiler	137
8.10.2 Chaining af programmer	138
8.10.3 APPEND af filer	139
8.10.4 Ikke lukkede filer	140
8.11 Relative filer	141
8.11.1 Database baseret på relative filer	143
8.12 Random filer	152
8.12.1 Blok kommandoer	152
8.12.1.1 Buffer pointer	153
8.12.1.2 Block read kommandoer	153
8.12.1.3 Block write kommandoen	153
8.12.1.4 Block allocate kommando	154
8.12.1.5 Block free kommando	154
8.12.1.6 Block execute kommando	155

8.12.2 Memory kommandoer	156
8.12.3 Slettebeskyttelse af programmer	157
8.12.4 Brug af flere disk-stationer	160
8.13 DOS 5.1 til Commodore 64	161
Kapitel 9 Kommunikation med printere og andet udstyr	164
9.1 Commodore 64 user port	164
9.1.1 RS-232 kommunikation	164
9.1.1.1 Baudrate, parity osv.	166
9.1.1.2 Formatvalg på Commodore 64	167
9.1.1.3 Åbning og lukning af RS-232 kanal	169
9.1.1.4 Datakommunikation via RS-232	170
9.1.2 STATUS og RS-232	173
9.2 Parallel kommunikation via user porten	174
9.3 Centronics interface	177
Kapitel 10 Commodore 64 grafik	186
10.1 Billedskærmens muligheder	186
10.1.1 Farvehukommelse for tekst	186
10.1.2 Multicolor tekst	187
10.1.3 Extended color tekst	188
10.2 Programmering af sprites	188
10.2.1 Fremstilling af sprites	189
10.2.2 Multicolor spritegrafik	192
10.2.3 Normal højopløsningsgrafik	192
10.2.4 Multicolor højopløsningsgrafik	194
10.3 Specielle skærmfunktioner	199
10.4 VIC chip'en og interrupts	201
Appendix A Oversigt over Commodore BASIC	203
A.1 Oversigt over Commodore 64 BASIC	266
A.2 Oversigt over DOS 5.1 kommandoer	267
Appendix B BASIC fejlmeldinger	268
B.1 Der er forskel på fejl	268
B.2 BASIC Fejlmeldinger	269
B.3 Ekstra fejlmeldinger Commodore Plus/4	273
B.4 Commodore 1541 fejlmeldinger	273
Appendix C Commodore 1541 disk-format	277
C.1 1541 BAM format	
C.2 Directory opbygning	277
C.3 En enkelt optegnelse i directory	279
Appendix D Forkortet indtastning af BASIC-ord	

## **Appendix E Commodores alfabet**

**E.1 SKERM-koder**

**E.2 ASCII-koder**

## **Appendix F Commodore skærmadresser**

**F.1 Commodore 64 skærmadresser**

**F.2 Commodore PLUS/4 skærmadresser**

## **Appendix G Commodore 64 tilslutninger**

## Forord

Commodores hjemmecomputere har opnået stor udbredelse verden over. Faktisk er Commodore 64 i dag verdens mest solgte computer overhovedet - uanset prisklasse. Også i Danmark indtager denne computer en betydningsfuld stilling. Det er også værd at fremhæve, at en meget stor del af Commodore 64 brugerne besidder den tilhørende disk-station Commodore 1541. I forening leverer disse to produkter en datakraft, der for få år siden kun eksisterede i større firmaers dataafdelinger.

I dag er hjemmecomputere et normalt indslag i dagligdagen i mange danske hjem. I langt de fleste tilfælde fungerer de som avancerede spillemaskiner, der leverer adspredelse og afveksling i hverdagen. Af mange opfattes dette element af morskab som noget odiøst. Danskere skal være fornuftige, og de må helst ikke have det for sjovt. Det faktum, at mange primært køber en computer for at more sig, er ikke velset i de "finere" kredse.

Legen vækker ofte nysgerrigheden og lysten til selv at prøve kræfter med disse nye maskiner og deres muligheder. I og for sig er dette både sundt og godt, men det mindsker ikke frygten for computeren i magtens korridorer. Tværtimod. Viden og indsigt er ensbetydende med magt, og om ikke andet, så fører det daglige arbejde med computere til øget viden og større indsigt på en lang række felter - ikke blot programmering.

At lære at programmere en computer er det samme som at lære at tale et andet sprog - f.eks. tysk eller engelsk. Blot er sproget langt simplere og mere begrænset. Dette sprog - hvadenten det hedder BASIC, Comal, Fortran, Forth eller Cobol - er i sig selv uinteressant. Det er blot et redskab, der gør det lettere at forme sine ideer og ønsker, så en computer - der er dum som et brædt - kan forstå dem. Først når man har forklaret computeren, hvad den skal gøre, er den i stand til at udføre den ønskede opgave.

Formålet med denne bog er udelukkende at hjælpe læserne til at forstå mulighederne og begrænsningerne i det computersprog, der benyttes i Commodores computere. Bogen henvender sig primært til brugere af Commodore 64 med tilhørende udstyr, dog gennemgås også de kommandoer, der findes i de nye Commodore computere Plus/4 og C16.

Brugere af andre Commodore computere vil også kunne læse bogen med godt udbytte, idet BASIC-sproget, der anvendes, stort set er det samme på alle Commodore maskiner. F.eks. svarer BASIC'en i Commodore 64 til den BASIC, der anvendes i VIC-20 og en del tidligere Commodore modeller. Den BASIC, der findes i Commodore Plus/4, har mange lighedspunkter med Commodores større erhvervsmaskiner.

Det primære mål med denne bog er at hjælpe læseren til at fremstille programmer, der virker. Derfor er begrebet "*korrekt programmeringsteknik*", som sådan, ikke behandlet. Jeg er af den overbevisning, at det eneste, der tæller, er, om et program virker og arbejder efter hensigten - og ikke de metoder og fremgangsmåder, der er anvendt, for at nå dette mål. Eller sagt på en anden måde:

Da vi som små skulle lære at binde vore snøreband, var det vigtigste at få knuden til at holde. Med voksende erfaring kom knuden til at se pænere ud, men virkningen forblev den samme!

Kurt Friis Hansen

### **Tak**

til alle de mennesker, der har bistået med råd og vejledning. Især til Jan Nymand fra Commodore Data A/S, som har ydet en uvurderlig bistand med fremskaffelse af materiale på Commodore Plus/4, men også en tak til Commodores administrerende direktør Kristian Andersen, som velvilligt stillede den nye Commodore computer til forfatterens rådighed, flere måneder før introduktionen her i landet.

Ligeledes en tak til Bjarne Hansen og Carl Eriø fra C&B Mikrodatainformation, der har gjort denne bog mulig, og til alle venner og bekendte, hvis forståelse har været en stor hjælp under arbejdet.



## Kapitel 1

### Computeren i praksis

En computer er ikke i stand til at udføre nogen form for arbejde, databehandling eller processer, medmindre en eller anden har fortalt den, hvad den skal foretage sig.

Når en moderne computer forekommer intelligent, er det udelukkende en menneskelig indsats, der er årsag hertil. Et programmerings-sprog er ikke andet end et mere eller mindre avanceret program, fremstillet af mennesker.

I sig selv er computeren helt blottet for initiativ, intelligens, omtanke osv. Den er bogstaveligt talt dum som et brædt!

Computerens eneste fordel er, at den er i stand til at udføre een gang stillede ordrer igen og igen, med en i menneskelig forstand formidabel sikkerhed, præcision og ikke mindst hastighed. Disse egenskaber skyldes netop, at computeren ikke er i stand til at tænke selv! Den får ingen gode ideer. Ej heller ideer, som efter et stykke tid viser sig knapt så gode!

Modsat manges opfattelse udfører en computer ikke altid de ordrer, den får besked på. Eller rettere, det produkt, vi opfatter som en færdig computer, har en varierende grad af pålidelighed.

Der findes to hovedårsager.

Den vigtigste er det menneskelige element. Det program, som styrer computeren, og overhovedet gør det muligt for os mere normale mennesker at kommunikere med maskinen, er fremstillet af mennesker. Uanset hvor store anstrengelser, programmøren har udfoldet, vil der *altid* være en eller anden kombination af indtastninger og ordrer, der ikke er taget højde for.

Den anden årsag er slid. Mange har den opfattelse, at de enkelte byggestene i en computer - de integrerede kredsløb - har en næsten ubegrænset levetid. Dette er ikke tilfældet. Selvom de er fremstillet af faste stoffer, hersker der en enorm aktivitet i kredsenes indre - en form for "bevægelse". Og alle former for bevægelse medfører slid under en eller anden form.

Vi ved af erfaring, at nogle mennesker kan tåle mere "slid" end andre. Vi bliver ikke lige gamle. Vi er heller ikke i stand til at opretholde samme fysiske aktivitetsniveau hele livet igennem. Med alderen indtræder små skavanker, som knirkende knæ, stivhed i kroppen osv. Dette gælder også for computere.

Den store fare er ikke, at computeren dør eller opfører sig helt sindsygt. Næh, den helt store fare er minimale fejl, som kun optræder med mellemrum eller i visse situationer - i computeren eller tilsluttet udstyr. De er næsten

umulige at spore. Akkurat som visse former for sindslidelser, der kun optræder visse situationer eller i perioder.

Dette er ikke sagt for at forskrække, men blot for at gøre opmærksom på, at man også har brug for en ganske sund portion kritisk sans, når man arbejder med computere! Der er *ingen* grund til at acceptere, at et eller andet udsagn er sandt, blot fordi det er resultatet af en computers arbejde.

Computer-verdenen er slet ikke så perfekt, som mange - ud fra forskellige motiver - forsøger at give indtryk af. Den indstilling vil du også nå, når du har prøvet at fremstille egne programmer. Du vil i praksis finde ud af, at *intet* her i livet er helt fejlfrit. Det er en meget positiv erfaring at få.

Alle former for databehandling er centreret omkring indlæsning og lagring af data under en eller anden form. Det program, en computer skal udføre, er data. De informationer, der indtastes som svar på programmet, er data, ligesom de resultater, der kommer ud af hele processen.

Fodrer vi maskineriet med forkerte data, bliver resultaterne også derefter. Denne simple sandhed, som mange har så svært ved at forstå, danner essensen i computer-branchens mest betydningsfulde grundregel:

### GARBAGE IN - GARBAGE OUT

- direkte oversat til dansk: *affald ind - affald ud!*

Programmørens opgave - uanset om der er tale om hobby eller erhverv - er at undgå "affald". Denne opgave er i sig selv så stor, at der ikke er plads til at kæmpe med mangelfulde eller manglende oplysninger om det udstyr, man anvender.

Denne bogs grundide, er at dokumentere brugen af Commodore 64 computeren, tilslutningsmuligheder og udstyr på en måde, så indlæsning, behandling og lagring af data kan foregå så sikkert og enkelt som muligt i praksis.

## Kapitel 2

### Commodore TASTATUR og SKÆRM

Tastaturet og billedskærmen er den allervigtigste del af computeren. Commodores computere foretrækkes ofte frem for andre, fordi tastaturet er af meget høj kvalitet - faktisk bedre end på mange professionelle computere. Commodores tastatur adskiller sig også fra mængden på et andet punkt. Alle tegn, der kan indtastes fra tastaturet, er tydeligt markeret herpå, så man ikke har brug for at slå op i manualen, bare for at fremstille et pænt skærm-lay-out i sine programmer.

Ud over almindelige tegn og grafik-karakterer, er det også muligt at udløse særlige funktioner fra tastaturet - farvevalg, flytning af cursoren (markøren), sletning af skærmen m.m.

Oveni dette råder man over en computer, der rummer den enklest mulige måde at rette indtastninger på. Modsat mange andre computere, kan man frit rette et BASIC-program overalt på skærmen - et tryk på RETURN-knappen, og computeren er klar over, hvad der gælder herefter. Princippet kaldes for en *full-screen editor*, modsat en *line editor*, som ofte finder anvendelse i andre computere. En *line editor* kræver, at man forud for hver rettelse fastlægger den programlinje, man ønsker at rette. Computeren skriver denne på skærmen - normalt nederst. Først herefter kan man rette i programlinjen - og kun i denne!

#### 2.1 Tastaturet i detaljer

Tastaturet aflæses konstant - 60 gange i sekundet - uanset hvad computeren iverigt foretager sig. Trykkes der på en knap på tastaturet, registrerer computeren dette, og gemmer værdien i et særligt område af hukommelsen - kaldet *tastatur-bufferen*.

Tastatur-bufferen kan rumme op til 10 tegn (tastetryk), hvilket i praksis vil sige, at selv om du et kort øjeblik arbejder meget hurtigt på tastaturet, vil computeren normalt altid registrere tegnene, selvom "udskriften" i et kort øjeblik ikke kan følge med! Computeren kan maksimalt klare 3600 tegn i minuttet - eller ca. 5 til 6 gange så meget, som verdens hurtigste person på en skrivemaskine kan klare!

Antallet af tegn i tastatur-bufferen, kan du finde på adressen (hukommelsespladsen) 198, og bufferens indhold lagres på adresserne 631 til

640. Princippet i bufferens virkemåde, kan du let afprøve med dette program:

```
130 A$="RUN"+CHR$(13):REM HUSK RETURN TIL SIDST
140 FOR N=1 TO LEN(A$)
150 : POKE 630+N,ASC(MID$(A$,N,1))
160 NEXT N
170 POKE 198,LEN(A$):REM ANTAL KARAKTERER
180 LIST
```

Normalt stopper et program *altid* efter LIST, men linjerne 140 til 160 indlæser kommandoen RUN efterfulgt af RETURN i tastaturbufferen. Computeren får i linje 170 at vide, at der er indtastet 4 tegn. Når LIST er afsluttet, tror den derfor, at du har indtastet kommandoen, og kører programmet igen!

Du kan udnytte denne teknik i mange situationer - også under fejlsøgning i programmer. Uanset om LIST kaldes fra et program eller ej, ned sætter et tryk på CTRL-knappen den hastighed, som listningen foretages med.

Alle karakterer er tilladte - også forkortede kommandoer (f.eks. ? for PRINT) og *kontrol-karakterer*. Kontrol-karakterer er en betegnelse for alle de tegn-koder, som ikke producerer nogen form for tekst i udskrifter - f.eks. i PRINT-sætninger. Disse karakterer kontrollerer selve udskriften - f.eks. *carriage return* (ASCII nr. 13) og *line feed* (ASCII nr. 10), som på dansk kaldes henholdsvis *vogn-retur* og *linjeskift*.

Normalt befinder alle kontrolkarakterer sig i området under ASCII værdi 32 (mellemrum). Commodore computere adskiller sig fra andre ved at området ASCII 128 til 159 også indeholder kontrol-karakterer (Se Appendix E). I mange programmer har man behov for at afgøre, om der er tale om kontrolkarakterer, der indtastes, eller ej. Anvendes den følgende lille rutine, er det meget let at skelne:

```
720 GET A$
730 IF (ASC(A$) AND 127)<32 THEN ...kontrolkarakter
740 normal karakter
```

#### 2.1.1 Auto-repeat på alle taster

Normalt er det kun ganske få af tasterne, der har automatisk *repeat*, men det kan let ændres med kommandoen:

```
POKE 650,128
```

som indkobler *auto-repeat* på *alle* taster, og:

```
POKE 650,0
```

som kobler tastaturet tilbage i *normal*-stillingen igen. Selvom autorepeat kan være behageligt at arbejde med, mens man udvikler programmer, er det ikke

umiddelbart nogen fordel, når "amatører" skal betjene computeren. Derfor kan det være en fordel at indlede programmer med den sidste kommando, for at sikre, at auto-repeat-funktionen altid er udkoblet, mens programmet kører!

### 2.1.2 Valg af alfabet

Når computeren tændes, er alfabetet altid STORE BOGSTAVER og GRAFIK. I mange programmer har man også behov for at benytte SMA BOGSTAVER. Det er derfor vigtigt, at tastaturet forbliver i den valgte stilling, mens programmet kører - især da hele skærbilledet påvirkes, når alfabetet "skiftes". Det er derfor altid en fordel, at indelede et program med de ASCII-koder (tegn), der udvælger det ønskede alfabet, samt at "låse" tastaturet med kommandoen:

```
PRINT CHR$(8)
```

Inden programmet forlades, vil det være en god ide, at "aflevere" tastaturet i normalstillingen igen. Dette gøres med:

```
PRINT CHR$(9);CHR$(142)
```

### 2.1.3 Udkobling af RUN/RESTORE m.m.

De fleste af os, har lært at påskønne virkningen fra et samtidigt tryk på RUN/STOP og RESTORE knapperne. Ligeså har vi ofte "forbandet" virkningen, når vi ved en fejltagelse er kommet til at stoppe et program - midt i det hele. Der findes dog en ganske enkel metode, som kan udkoble RUN/RESTORE kombinationen; nemlig:

```
POKE 808,225 - indkobling: POKE 808,235
```

Ønskes kun en udkobling af STOP-funktionen, benyttes:

```
POKE 808,239 - indkobling: POKE 808,237
```

Ønsker man at forhindre listning af sine programmer, kan det også gøres. Indtast blot:

```
POKE 775,200 - indkobling: POKE 775,167
```

## 2.2 Læsning af tastaturet fra BASIC

I langt de fleste tilfælde, er GET-kommandoen, mere end tilstrækkelig, når man vil læse tastaturet - eller afgøre, om en knap har været trykket på tastaturet. Den normale fremgangsmåde er:

```
97Ø GET A$:IF A$="" THEN 97Ø
```

Ofte har man kun behov for at vide, om der trykkes på en knap eller ej - værdien er i den forbindelse underordnet. Har man plads-problemer i sit program, eller ønsker man at undgå *garbage collection* (se næste kapitel) så længe som muligt, er den følgende løsning langt bedre, idet der ikke anvendes nogen streng-variabel overhovedet:

```
128Ø WAIT 2Ø3,63
```

WAIT - kommandoen læser (i dette tilfælde) indholdet i adresse 203 og AND'er resultatet med 63. Rutinen venter indtil resultatet heraf er forskelligt fra nul. Funktionen er eksakt den samme, som i denne linje:

```
15ØØ IF (PEEK(2Ø3) AND 63)=Ø THEN 15ØØ
```

men WAIT-sætningen er både kortere og mere "elegant". Adressen 203 indeholder normalt værdien 64, men trykkes der på en knap på tastaturet, ændres værdien til *nummeret* på knappen (max. 63) - ikke at forveksle med ASCII-værdien (lagres på adresse 197)!

Vil man også være sikker på, at programmet først fortsætter, når brugeren igen har løftet fingeren fra knappen, kan følgende programlinje tilføjes:

```
129Ø WAIT 2Ø3,64
```

Nu fortsætter programmet først, når knappen igen er sluppet!!!

WAIT-kommandoen rummer en række ekstra muligheder for tastatur kontrol - muligheder, som GET eller INPUT kommandoerne ikke byder på. Mulighederne fremgår af følgende liste:

WAIT 653,1	Venter indtil en SHIFT knap trykkes.
WAIT 653,2	Venter indtil COMMODORE knappen trykkes.
WAIT 653,4	Venter indtil CTRL knappen trykkes.

Princippet kan også anvendes, hvis man vil være helt sikker på, at der findes tegn i tastatur-bufferen - f.eks. er det ikke nødvendigt at teste for en tom streng (IF A\$="" THEN...) i følgende situation:

```
1250 WAIT 198,255
1260 GET A$
1270 RETURN
```

Linje 1260 udføres kun, hvis der er karakterer i tastaturbufferen (adresse 198 fortæller antallet).

## 2.3 Dit helt personlige tastatur

I mange tilfælde har man brug for et skræddersyet tastatur - f.eks. kunne det være meget behageligt, at man kunne placere f.eks. danske bogstaver, hvor de hører hjemme - både store og små. Eller i det hele taget kreere et tastatur, helt efter sine egne ønsker.

Opgaven kan deles i to punkter:

1. Fremstilling af specielle tegn forskellige steder på tastaturet.
2. Tildeling af bestemte ASCII-værdier til bestemte taster. F.eks. er det en let sag, at fremstille danske karakterer - men vil vi også have de danske karakterer til at optræde på den rette plads i ASCII-alfabetet - f.eks. fordi vi foretager alfabetiske sorteringer i et program - stilles lidt andre og større krav. Standard ASCII-værdierne for de danske tegn er:

Æ - 91	æ - 123
Ø - 92	ø - 124
Å - 93	å - 125

Commodores tastatur, bytter om på små og store bogstaver, og det kan vi ligesågodt tage med i overvejelserne, men dermed har vi jo ikke løst opgaven.

### 2.3.1 Brugerdefinerede tegn

Normalt læses de tegn, computeren skriver på skærmen, fra en ROM-kreds, der indeholder mønsteret på hvert tegn. Hvert tegn består af 8 gange 8 punkter, og mønsteret rummes i 8 på hinanden følgende bytes i ROM'en (Læs mere om bits og bytes i kapitlet Programmering i maskinkode). Indholdet i ROM'en kan vi ikke forandre, men det er muligt, at skabe vort eget tegnsæt i RAM og fortælle computeren, at dette tegnsæt skal benyttes i stedet.

Tegnenes indbyrdes placering i ROM'en - og senere i RAM-hukommelsen - afgøres af POKE-koden og ikke af ASCII-koden. En fremgangsmåde, som ikke gør tingene lettere at forstå, men forenkler sagerne for computeren.

POKE-værdierne kan meget let findes, ved direkte at POKE forskellige tal ind i skærm-hukommelsen. Denne er placeret fra adresse 1024 og frem til og med adresse 2023 - medmindre vi fortæller computeren noget andet. Det følgende lille program, giver dig en udskrift af alle karaktererne i ROM'en - bemærk, at der findes et komplet "billedsæt" for hvert af de to alfabeter, computeren råder over.

```

100 PRINT CHR$(147)
110 PRINT "ØNSKES ALFABET 1 ELLER 2"
120 WAIT 203,63:GET A$
130 IF A$<"1" OR A$>"2" THEN 120
140 A=VAL(A$)-1
150 PRINT CHR$(14+A*128):REM ALFABETVALG
160 T=1104
170 FOR N=0 TO 255
180 : POKE T+N,N
190 NEXT N
200 PRINT CHR$(19);
210 GOTO 110

```

Nu er det blot at tælle sig frem på skærmen (eller at slå op i tabellen i Appendix E) - så har vi de rigtige koder. Bemærk, at alle koder mellem 128 og 255 skriver *negative* tegn.

Vi ved nu, at de *billeder* vi ønsker, at ændre svarer til koderne 27, 28 og 29 - lad os nøjes med at ændre alfabet 1, og kun de "positive" bogstaver. Har du først fundet ud af systemet, er det en let sag, at ændre de øvrige "bogstav-billeder", så de svarer til dine krav.

Først skal vi have fundet billedernes *start-adresse*. Den kan beregnes med følgende formel:

$$\text{startadresse} = \text{bogstavkode} * 8 (+2048 \text{ hvis alfabet 2})$$

Vi får da startadresserne for "billed/bogstavkoderne" 27, 28 og 29 til henholdsvis 216, 224 og 232. Vor næste opgave er at fremstille "billederne".

Når computeren viser et tegn på skærmen, kan vi illustrere et tændt punkt (bogstavfarven) med 1 og et slukket punkt (baggrundsfarven) med 0. F.eks. ser bogstavet M således ud, set med computerens øjne:

```

01100011 = 99
01110111 = 119
01101011 = 107
01100011 = 99
01100011 = 99
01100011 = 99
01100011 = 99
00000000 = 0

```

Hver af de vandrette *bit-mønstre* svarer til 1 byte i computerens hukommelse, og de efterfølgende decimaltal er de værdier, der skal bruges for at skabe de



ønskede mønstre. Bemærk, at første søjle og nederste række altid er 0, når der er tale om almindelige bogstaver og tegn. Det skaber afstand mellem tegnene og linjerne! Ønsker vi, at tegnet skal vises negativt, erstatter vi blot 1 med 0 og 0 med 1. Bemærk også, at alle lodrette linjer i et tegn har dobbelt bredde (11 i stedet for 1). Det sker for at gøre tegnene tydeligere på en TV-skærm!

Lad os nu prøve, at fremstille billederne på vore tre tegn E, Ø og Å.

00111111	63	00111110	62	00011100	28
01100110	102	01100011	99	00000000	0
01100110	102	01100111	103	00111110	62
01111111	127	01101011	107	01100011	99
01100110	102	01110011	115	01100011	99
01100110	102	01100011	99	01111111	127
01100111	103	00111110	62	01100011	99
00000000	0	00000000	0	00000000	0
E		Ø		Å	

Nu har vi billederne, og skal blot have fortalt computeren, hvor den kan finde vort nye alfabet.

Først skal vi vælge et sted, at placere alfabetet. Dette valg skal foretages i to trin. Først skal vi træffe et overordnet valg, nemlig at vælge det 16K-ROM område, hvori alfabetet skal placeres. Dette er det vigtigste valg, idet også skærm-hukommelsen skal befinde sig i samme område! De fire områder, vi kan vælge mellem er:

nr	adresse	virkning
0	0-16383	ROM-karaktersæt tilgængeligt - normalt
1	16384-32767	ROM-karaktersæt ikke tilgængeligt
2	32768-49151	ROM-karaktersæt tilgængeligt
3	49152-65535	ROM-karaktersæt ikke tilgængeligt

Da BASIC-området strækker sig fra 2048 til 40959 er det ikke videre fornuftigt at placere et nyt alfabet her. Næ, lad os vælge område 3! Her kan vi ovenikøbet drage fordel af, at computerens billedkreds (VIC-chip'en) ikke kan "se" ROM-kredsene. Set fra billedkredsens side, er hele området fra 49152 til 65535 RAM. Vi kan således også placere vort alfabet i det RAM-område, der ligger "skjult" under KERNAL-ROM'en - fra adresse 57344 og op! Det eneste vi skal huske, er at vi også skal flytte skærmen, og at denne altid skal ligge i et RAM-område, der er "synligt" fra KERNAL-ROM'ens side - dvs. i området 49152 til 53247. Området fra 53248 til 57344 er forbeholdt I/O-kredse, programmering af lyd, samt programmering af billedkredsen. I denne forbindelse er det FORBUDT område.

Skærmen optager kun 1000 bytes, og kan i praksis placeres på enhver 1K byte grænse, i den valgte blok - ialt 16 forskellige steder.

Placeringen bestemmes således:

nr	adresse	bemærkninger
0	0	Forbudt i område 0
1	1024	Normalt (område 0)
2	2048	
3	3072	
4	4096	ROM-billeder (område 0 og 2) - forbudt i område 3
5	5120	ROM-billeder (område 0 og 2) - forbudt i område 3
6	6144	ROM-billeder (område 0 og 2) - forbudt i område 3
7	7168	ROM-billeder (område 0 og 2) - forbudt i område 3
8	8192	Optaget af ROM'er (område 2 og 3)
9	9216	Optaget af ROM'er (område 2 og 3)
10	10240	Optaget af ROM'er (område 2 og 3)
11	11264	Optaget af ROM'er (område 2 og 3)
12	12288	Optaget af ROM'er (område 2 og 3)
13	13312	Optaget af ROM'er (område 2 og 3)
14	14336	Optaget af ROM'er (område 2 og 3)
15	15360	Optaget af ROM'er (område 2 og 3)

Skærmens eksakte adresse, beregnes som startadressen på det valgte område fra foregående tabel plus en værdi fra ovennævnte tabel. Er startadressen f.eks. 1024 i område 3 fås:

$$49152+1024=50176$$

Lad os blot beslutte, at det er her vi placerer skærmen. Nu skal vi blot have forklaret det til din Commodore 64.

Først fortæller vi den, hvilket grundområde vi benytter (0 til 3). Det sker således:

POKE 56576,(PEEK(56576)AND252)OR(3-grundområdenummer)

i vort tilfælde:

POKE 56576,(PEEK(56576)AND252)OR(3-3)

Dernæst skal vi fortælle computeren, hvilket delområde (0-15) vi har valgt. Det sker således:

POKE53272,(PEEK(53272)AND15)OR(16\*delområde)

i vort tilfælde:

POKE53272,(PEEK(53272)AND15)OR(16\*1)

Når dette er gjort, skal computerens styresystem have at vide, hvor det kan finde skærmen henne - ellers skrives alle tekster til det gamle område. Det

fortæller vi computeren således:

POKE 648,(64\*grundområde+4\*delområde)

i vort tilfælde:

POKE 648,(64\*3+4\*1)

Vi er ikke helt færdige endnu! Vi skal også have fortalt computeren, hvor den kan finde tegnsættet! Vi har valgt grundområdet, så det eneste, der er tilbage er at vælge delområdet for tegnsættet. Der er ialt 8 delområder for tegnsættet, hver på 2 K. Der gælder dog det specielle forhold, at alle delområder, med et lige nummer (0, 2, 4 og 6) svarer til det område, der normalt betegnes som alfabet 1, og de ulige numre (1, 3, 5 og 7) betegnes som alfabet 2. Uanset, hvilket område vi vælger, vil Commodore+SHIFT knapperne medføre, at computeren flytter sin opfattelse af, hvor alfabetet starter, frem og tilbage mellem delområderne 0 og 1, 2 og 3, 4 og 5 samt 6 og 7. I praksis kan vi derfor ligesågodt forestille os, at der kun findes 4 delområder, der hver rummer 2 alfabeter - nemlig alfabet 1 (store bogstaver og grafik) og alfabet 2 (store og små bogstaver). Disse fire alfabetområder ligger således placeret i hvert af de fire grundområder:

nr	adresse	bemærkninger
0	0	
1	4096	ROM-karakterer (grundområde 0 og 2) kan læses i grundområde 3
2	8192	
3	12288	

Når vi fortæller computeren, hvor den skal finde vort nye alfabet, så gør vi det således:

POKE53272,(PEEK(53272)AND240)OR(4\*alfabetområde)

i vort tilfælde:

POKE53272,(PEEK(53272)AND240)OR(4\*3)

som placerer "billederne" af tegnene i de to alfabeter under KERNAL-ROM'en fra adresse 61440 (49152+12288).

Nu mangler vi blot at kopiere begge alfabeter over i det nye område, inden vi fortæller computeren, at den skal anvende et andet "billedmateriale". Denne kopiering, foretages således:

```
100 REM AFBRYD INTERRUPTS OG TASTATUR
110 POKE 56334,PEEK(56334)AND 254
120 REM INDKOBLING AF KARAKTERROM
130 POKE 1,PEEK(1)AND251
140 REM SELVE KOPIERINGEN
150 FOR N=0 TO 4095
160 : POKE 61440+N,PEEK(53248+N)
170 NEXT N
180 REM UDKOBLING AF KARAKTERROM
190 POKE 1,PEEK(1) OR 4
200 REM TILLAD INTERRUPTS OG TASTATUR
210 POKE 56334,PEEK(56334)OR 1
```

Nu kan vi flytte skærmen, indlæse vore egne karakterer osv. Normalt sker det efter linje 210, så ingen bemærker, hvad vi har gjort, men for at du kan se, hvad der sker, gør vi det omvendt denne gang. Indtast dette program, sammen med det foregående, og nyd resultatet!

```
10 REM NY SCREEN-ADRESSE
15 POKE 648,3*64+4*1
20 REM START PAA ALFABETET
25 POKE53272,(PEEK(53272)AND240)OR(4*3)
30 REM START PAA SKAERM
35 POKE 53272,(PEEK(53272)AND15)OR(16*1)
40 REM START PAA GRUNDOMRAADE
45 POKE 56576,(PEEK(56576)AND252)OR(3-3)
50 REM SKRIV ALLE TEGN
55 FOR N=0 TO 255
60 : POKE 256*PEEK(648)+N,N
65 NEXT N
```

Vort personlige alfabet indlæses med disse programlinjer:

```
300 AF=61440:AE=216
310 FOR N=AF+AE TO AF+AE+31
320 : READ X
330 : POKE N,X
340 NEXT N
350 PRINT "PRØV AT FINDE TEGNENE PÅ TASTATURET"
360 END
370 DATA 63,102,102,127,102,102,103,0
380 DATA 62,99,103,107,115,99,62,0
390 DATA 28,0,62,99,99,127,99,0
```

### 2.3.2 Brugerdefineret tastatur

Før vi kan definere et andet tastatur - ikke blot bogstav-"billeder" - er det nødvendigt, at vide, hvordan computeren arbejder med tastaturet.

Hver gang, der trykkes på en knap, indlæses en værdi mellem 0 og 63 i adresse 203 (se WAIT-kommandoen herover). Denne værdi, anvendes til opslag

i een af fire tabeller. De fire tabeller er

NORMAL ASCII-værdi uden andre knapper  
 SHIFT ASCII-værdi sammen med SHIFT-knappen  
 CONTROL ASCII-værdi sammen med CTRL-knappen  
 COMMODORE ASCII-værdi sammen med C=-knappen

Afhængigt af knap-kombinationen, laves et opslag i en af tabellerne, og den til "bogstavknapperne" tilhørende ASCII-værdi, findes frem. Er knap-kombinationen ikke defineret, findes værdien 255 på den pågældende plads.

Hver af de fire tabeller, er bygget over samme skema. Næmlig:

	0	1	2	3	4	5	6	7
+ 0	DEL	RETURN	CUR	<del>PA</del> <sup>F2</sup>	<del>PS</del> <sup>F1</sup>	<del>PF</del> <sup>F3</sup>	<del>PT</del> <sup>F5</sup>	CUD
+ 8	3	W	A	4	Z	S	E	VS
+16	5	R	D	6	C	F	T	X
+24	7	Y	G	8	B	H	U	V
+32	9	I	J	Ø	M	K	O	N
+40	+	P	L	-	.	:	@	,
+48 (92)	*	;	HOME	HS	=	(94)	/	
+56	1	(95) CTRL	2	SPACE	C=	Q	STOP	

*64 = No help, please!*

CUR = cursor til <sup>højre</sup>venstre, CUD = cursor ned, VS og HS er henholdsvis venstre og højre SHIFT-knap, tallene i parenteser, er ASCII-værdier for karaktererne pund, pil op og pil til venstre!

De fire tabeller, starter på adresserne:

start	type
60289	NORMAL
60354	SHIFT
60419	COMMODORE
60536	CONTROL

Vil vi ændre betydningen af en knap - f.eks. således at tegnene ";" og "." giver Æ og Å, samt ; og : sammen med SHIFT, skal vi blot ændre positionerne 45 og 50 i NORMAL og i SHIFT tabellen, så de indeholder henholdsvis ASCII-værdierne 91 og 93 for NORMAL tabellen samt 59 og 58 i SHIFT tabellen. Der er blot et lille problem. Tabellen ligger i ROM, og vil vi foretage ændringer, må vi kopiere både BASIC- og KERNAL-ROM over i den underliggende RAM-hukommelse, og derefter koble disse ROM-kredse ud.

OBS: Det forgående program, der placerer karakter-"billederne" under KERNAL-ROM'en kan nu ikke anvendes. Det ville have helt katastrofale følger. Ønsker du at kombinere den teknik, der vises i det følgende, er det bedst at placere skærm og karaktersæt under adressen 40960 - i et område, der er beskyttet mod overskrivning fra BASIC (Se afsnittet Hvorhen med maskinkoden senere i bogen).

Når vi skal kopiere de to ROM'er til det underliggende RAM-område, kan vi

udnytte den lille finesse, at *alle* de værdier vi POKE'r til ROM-adresserne, lander i den underliggende RAM. Når vi PEEK'er adressen, får vi naturligvis kun værdierne i ROM'erne - indtil videre. Kopieringen foretages således:

```
100 FOR N=40960 TO 49151
110 : POKE N,PEEK(N):REM BASIC-ROM
120 : POKE N+16384,PEEK(N+16384):REM KERNAL-ROM
130 NEXT N
```

Nu mangler vi blot, at koble både KERNAL og BASIC ROM'erne ud. Det sker således:

```
140 POKE 1,PEEK(1)AND 253
```

Nogen umiddelbar forskel kan vi ikke mærke, men prøv tastaturet, efter at følgende programlinjer er udført:

```
140 POKE 60289+45,91
150 POKE 60289+50,93
160 POKE 60354+45,58
170 POKE 60354+50,59
```

Kombinerer du denne teknik med den foregående, er det *intet* problem, at fremstille et tastatur, der lever op til alle dine ønsker - herunder også danske tegn, placeret de rigtige steder på tastaturet, med de rigtige ASCII-værdier! Dit eneste problem er nu teksten på knapperne - prøv at forsyne tastaturet med nogle små selvkøbende mærkater, som du har forsynet med *dine* tegn - det virker faktisk!

Bemærk, at computeren går tilbage i *normal*-stillingen, hver gang, der trykkes på RUN/RESTORE! Det kan forhindres med en af de tidligere beskrevne POKE'r, men fremgangsmåden bør kun anvendes i programmer, der virker. Ellers skal du normalt starte helt forfra, hvis computeren begynder at skabe sig!!!!

#### 2.4 Billedskærmen i Commodore 64 BASIC

Blandt de største problemer i Commodore 64 BASIC, er styring af udskrifternes placering. Der findes ingen kommando, som på f.eks. Plus/4 (Se LOCATE kommandoen i Appendix A).

Normalt løses problemet ved at anvende det nødvendige antal *cursor down* og *cursor right* karakterer, men en mere elegant metode findes - nemlig:

```
POKE 214,linje:POKE 211,kolonne:SYS 58732
```

Linje-nummeret kan være i området 0 til 24, og kolonne-nummeret i området 0 til 39. SYS-kommandoen placerer cursoren (markøren), det ønskede sted på skærmen.

I enkelte tilfælde kan det også være ønskeligt, at vise cursoren på skærmen - f.eks. i forbindelse med en GET-kommando. Nedenstående rutine, placerer cursoren, det ønskede sted på skærmen (variablerne LI og KO), tænder for cursoren og afventer en indtastning. Efter indtastningen slukkes cursoren igen, og værdien skrives på skærmen:

```
100 POKE 214,LI:POKE 211,KO:SYS 58732
110 POKE 204,0:REM CURSOR ON
120 WAIT 203,63:REM AFVENT INPUT
130 GET A$
140 IF PEEK(207) THEN 140
150 POKE 204,1:REM CURSOR OFF
160 PRINT A$;
```

Linje 140 venter, indtil cursoren er i den "usynlige" fase, hvor vi uden problemer kan afbryde cursor'en. Hvis cursoren var "tændt" i dette øjeblik ville den efterlade den kendte "hvide" firkant på skærmen, og det står næppe højest på ønskelisten.

## 2.5 Man kan mere end blot skrive til skærmen

I Commodore 64 virker skærmen som et *device* på linje med printer, kassettebåndoptager og disk-station. I praksis betyder det, at man både kan skrive til og læse fra skærmen - akkurat som man kan skrive til eller læse fra en fil på en diskette.

### 2.5.1 Hvad er et device

Computeren kender ikke noget til skærme, disk-stationer m.m. Den kender kun *devices* (enheder), som man enten kan sende data til, modtage data fra eller begge dele. Dette er ikke 100% sandt, men hvis vi accepterer, at tingene forholder sig sådan i store træk, vil det følgende være langt lettere at forstå.

Styringen af kommunikationen med de enkelte *devices* sker i operativsystemet, som Commodore har givet navnet *KERNAL*. I Commodore 64 rummes *KERNAL*-rutinerne i ROM - fra adresse hex E45F og opefter. Derfor kaldes denne ROM også *KERNAL-ROM*.

Alle *devices* i Commodores operativsystem er tildelt et særligt nummer, som klart specificerer overfor *KERNAL*-rutinerne, hvordan de skal forholde sig til hvert enkelt *device*. F.eks. har skærmen *device* nummer 3, og dette nummer

er det *eneste*, der skelner udskrifter til skærm fra udskrifter til f.eks. RS-232 (device nummer 2). Listen over gyldige device numre ser således ud:

nr	retning	enhedsbetegnelser
-----		
0	kun ind	tastatur
1	ind og ud	kassettebåndoptager
2	ind og ud	RS-232 (seriel port)
3	ind og ud	skærm
4-5	kun ud	printere (IEC seriel port)
8-11	ind og ud	disk-stationer(IEC seriel port)

Alle device-numre fra 4 og op til 15 går i realiteten til den serielle IEC-port, selvom kun de nævnte numre har en fastlagt betydning.

## 2.5.2 Abning og lukning til devices

Hver gang vi ønsker at sende data til eller modtage data fra et device, skal vi åbne en kommunikations-kanal. Er der tale om udskrift til skærm (PRINT) eller indlæsning fra tastatur (GET eller INPUT) løser BASIC automatisk problemet for os. I alle andre situationer, må vi selv sørge for at åbne en kanal til det pågældende device - f.eks. skærmen (læs mere om de øvrige devices i de særlige kapitler for kassette, disk og RS-232 m.m.).

Abning af en kommunikationskanal sker med kommandoen:

OPEN *logisk fil, device, sa*

evt. fulgt af et *fil-navn* (navn på et program eller datasamling). Det logiske filnummer, er et nummer, som anvendes i alle efterfølgende kommandoer til det pågældende device. Denne teknik sparer os for at specificere "hele remsen", hver gang vi vil tale med et device. Det logiske filnummer kan valgfrit fastlægges til en værdi mellem 1 og 127 - værdier i området 128 til 255 er primært forbeholdt printerbrug, idet de fremtvinger en automatisk *line-feed* efter hver *carriage-return* (RETURN).

Betegnelsen *sa* står for begrebet *secondary address* - sekundær adresse - og denne specificerer en række forskellige funktioner, afhængigt af det benyttede device. I vort tilfælde - skærmen - kan *sa* antage 2 værdier, nemlig 0 og 1, som specificerer henholdsvis *læsning* fra og *skrivning* til skærmen. OPEN-kommandoen kan forekomme i tre versioner i forbindelse med skærmen:

```
OPEN lf,3,0 - læsning fra skærm
OPEN lf,3   - læsning fra skærm (identisk)
OPEN lf,3,1 - skrivning til skærm
```

Skal der skrives til et åbnet device, sker det med en særlig udgave af PRINT - kommandoen, nemlig *PRINT#*. Bemærk, at kommandoen *ikke* må forkortes til *?#*, ligesom der heller ikke må forekomme mellemrum mellem "PRINT" og tegnet



"#!" Kommandoen benyttes således:

PRINT# *If*, variabler, tekst etc.

hvor *If* er det logiske filnummer, der blev anvendt i OPEN-kommandoen. Analogt hermed, findes også to læse-kommandoer, nemlig GET# og INPUT#, der anvendes således:

GET# *If*,X\$,N,Y\$(23).....

INPUT# *If*,X\$,N,Y\$(23).....

Også her står *If* for det logiske filnummer, der blev anvendt i OPEN-kommandoen.

Når vi har afsluttet kommunikation med et *device*, skal vi "afbryde forbindelsen" - lukke kanalen - igen. Det sker med CLOSE-kommandoen, der har en opbygning, der er helt anlog til de foregående kommandoer - dvs.:

CLOSE *If*

hvor *If* igen står for det logiske filnummer, der blev anvendt i OPEN-sætningen.

Har du besvær med at forstå sammenhængen så forestil dig, at du vil ringe til en bekendt. Først løftes røret, og nummeret drejes (OPEN). Dernæst foregår samtalen. Din tale svarer til PRINT#, og lytning svarer til INPUT# og GET#. Når røret igen lægges på, svarer det til CLOSE. Computeren adskiller sig dog ved, at den kan håndtere op til 10 "telefonlinjer" på een gang.

### 2.5.3 Læsning af skærmens indhold

En af de mest kraftfulde anvendelser af disse kommunikationsmuligheder er læsning af det *faktiske* indhold i et nærmere defineret udsnit af skærmen.

Praktisk taget alle programmer forudsætter en vis form for kommunikation med brugeren. Alle programmer, som tillader dette, kaldes *interaktive programmer* - uanset omfanget af brugerstyringen. Den del af et program, der udgør *grænsefladen* eller *interface't* til brugeren er samtidig den mest krævende del af al programmering. Målet er at sikre programmet mod "tossede" svar.

Normalt anvendes udelukkende GET og INPUT kommandoer, til indlæsning af data undervejs i et program. GET-kommandoen er den mest velegnede, men stiller samtidig de største krav til kontrol af indtastningen. INPUT kommandoen er enkel at anvende, men helt umulig at styre - prøv f.eks. at se, hvad der sker, hvis du ved en fejltagelse kommer til at trykke på HOME-knappen - eller endnu værre CLR-knappen - midt i en INPUT-sætning. Uøvede brugere bliver normalt forvirrede eller forskrækkede over resultatet, og helt fikst virker den slags "forudsigelige uforudsigelige" effekter jo ikke.

En bedre, og mere enkel løsning, er en kombination af GET-kommandoen og læsning fra skærm. Metoden, som gennemgås i detaljer herunder, giver 100 procent sikkerhed i indtastningen, uden at vi behøver, at lave større kontrolrutiner!

Lad os antage, at et program kræver indtastning af 4 og altid 4 cifre, et bestemt sted på skærmen. Brugeren skal samtidig have lov til at foretage rettelser, lige så vildt han/hun vil - f.eks. ved at flytte cursoren frem og tilbage og overskrive uønskede værdier. Vanskeligt? Nej, slet ikke. Løsningen kan f.eks. være følgende program:

```
400 WIDTH=3:REM ANTAL TEGN (0, 1, 2 OG 3)
410 LINJE=9:KOL=15:REM PLACERING AF FØRSTE TEGN
420 CURSOR=0:REM AKTUEL PLACERING
430 GOSUB 900:PRINT"****";:REM VIS FORMATET
```

Først placerer vi cursoren i "arbejdsfeltet", og vi viser formatet til brugeren. Dit program bør naturligvis også indeholde en eller anden form for tekst, som forklarer, hvilken indtastning, der forventes.

```
440 GOSUB 900:REM PLACER CUSOREN
450 GOSUB 800:REM LÆS TASTATURET
460 IF (ASC(IND$)AND 127)>32 THEN 530
```

Linje 460 tester, om der er tale om kontrolkarakterer. Er det tilfældet, kan der være tale om RETURN (indtastning afsluttet) eller de to cursor-karakterer, vi tillader.

```
470 REM CURSORKONTROL M.M.
480 IF IND$=CHR$(13) THEN 570
490 IF IND$=CHR$(29) THEN IF CURSOR<WIDTH THEN CURSOR=CURSOR+1
500 IF IND$=CHR$(157) THEN IF CURSOR>0 THEN CURSOR=CURSOR-1
510 GOTO 440
```

Oftest er det en fordel, at teste indtastningen "på stedet", hvis det overhovedet kan lade sig gøre. Linje 530 medfører, at kun "gyldige" karakterer dukker op på skærmen! Overvej evt. at skrive en fejlmelding på skærmen, der informerer brugeren, når der er foretaget en forkert indtastning. Selve indtastningens art, bør fremgå af den "indledende" tekst til "arbejdsfeltet":

```
520 REM TEST AF TILLADTE KARAKTERER
530 IF IND$<"0" OR IND$>"9" THEN 440
540 PRINT IND$;
550 GOTO 440
```

Først skal vi indlæse karaktererne i den endelige tekst-variabel. Det sker direkte fra skærmen, så vi er 100% sikre på, at få det endelige resultat, uanset hvor mange overskrivninger, der er foretaget.

```

560 REM AFSLUTNING EFTER TRYK PAA RETURN
570 CURSOR=WIDTH+1:TEXT$=""
580 GUSUB 900:REM PLACER CURSOR I STARTEN
590 OPEN 1,3
600 FOR N=0 TO WIDTH
610 : GET#1,IND$
620 : TEXT$=TEXT$+IND$

```

Når vi tillader "frie" brugerindtastninger, skal vi huske at undersøge, om alle "feltmærkerne" (\*) er blevet udfyldt! Hvis ikke, placeres cursoren på positionen, og der springes tilbage til indtastningsrutinen. Bemærk linje 630, som sikrer, at FOR-NEXT-løkken altid forlades på en civiliseret måde. Det er aldrig særlig fornuftigt at springe direkte ud af en løkke, der ikke er blevet afsluttet (Se emnet løkker i næste kapitel).

```

630 : IF IND$="*" THEN CURSOR=N:N=WIDTH
640 NEXT N
650 IF CURSOR=(WIDTH+1) THEN 440
660 REM TEXT$ INDEHOLDER GODKENDT INDTASTNING!
670 RETURN:REM TILBAGE TIL HOVEDPROGRAM

```

Vor velkendte indtastningsrutine, som også tænder og slukker for cursoren.

```

790 REM AFVENT INDTASTNING
800 POKE 204,0:REM CURSOR ON
810 WAIT 203,63
820 GET IND$
830 IF PEEK(207) THEN 830
840 POKE 204,1:REM CURSOR OFF
850 RETURN

```

Cursoren og efterfølgende tegns placering fastlægges:

```

890 REM POSITIONER CURSOR
900 POKE 214,LINJE
910 POKE 211,KOL+CURSOR
920 SYS 58732
930 RETURN

```

## 2.5.4 Avanceret INPUT i BASIC

I praksis kan de specielle egenskaber ved Commodore 64'eren's skærm og tastatur udnyttes og kombineres på utallige måder.

Oftentimes kan det være en fordel, at lade programmet levere det mest sandsynlige svar, på forskellige former for input, således at brugeren kun skal indtaste en ønsket værdi, hvis der ikke er tale om en "normal-situation". I alle normale situationer, vil et tryk på RETURN-knappen være nok. Lad os se på et praktisk eksempel.

Programmet herunder skriver meddelelsen "HVORDAN HAR DU DET I DAG?" på skærmen, efterfulgt af normal-svaret "FINT!". Trykkes på RETURN - uden yderligere indtastninger - indlæses teksten "FINT!" i variablen A\$. Ønskes et andet svar, skrives dette blot oveni teksten!

```
100 A$="FINT!":REM NORMAL-SVARET
110 GOSUB 500:REM SKRIV TIL TASTATUR-BUFFER
120 INPUT"HVORDAN HAR DU DET I DAG";A$
130 RETURN
```

Normal-beskeden indlæses i tastaturbufferen. Den maksimale længde er 10 karakterer, men ønsker vi at flytte cursoren tilbage til starten af teksten - hvis et andet INPUT skulle ønskes - er der kun plads til det halve.

```
500 X=LEN(A$)
510 IF X=0 THEN RETURN
520 IF X>5 THEN X=5
530 FOR N=1 TO X
540 : POKE 630+N,ASC(MID$(A$,N,1))
550 NEXT N
```

Cursoren flyttes tilbage til første tegn i "normal-teksten" i INPUT-sætningen. ASCII 157 er lig med *cursor til venstre*.

```
560 FOR N=1 TO X
570 : POKE 630+X+N,157
580 NEXT N
```

Computeren informeres om tegnenes antal i tastaturbufferen.

```
590 POKE 198,2*X
600 RETURN
```

## 2.5.5 Farver og andre effekter med INPUT

Commodores specielle måde at styre skærmen på er ikke luttet ulemper. Anvendelsen af kontrolkarakterer i stedet for BASIC-kommandoer til bestemmelse af farver m.m. gør det muligt, at konstruere særdeles avancerede INPUT-meddelelser.

Alle kontrolkarakterer - uanset type - kan anvendes i tekstdelen i INPUT-kommandoen (INPUT"*tekstdel*";*variabel*). Det er således muligt at starte tekstdelen med RØD-koden og slutte med HVID-koden, således at spørgsmålet skrives i rødt, og svaret indtastes med hvide tegn. Det er muligt, at slette/overskrive tegn på en foregående linje, flytte en del af teksten hen efter spørgsmålstegnet, så den virker som *normal-indtastning* akkurat på samme måde, som i foregående eksempel.

Mulighederne er så store, at det er værd at eksperimentere lidt i praksis!

Prøv dig frem med *cursor*-karaktererne, farve-karaktererne osv. - ja hvorfor ikke prøve at udskrive en tekst, der understreges i linjen nedenunder teksten! Det kan *sagtens* lade sig gøre i *tekstdelen* af INPUT-kommandoen!

God fornøjelse!

## Kapitel 3

### Commodore BASIC

Commodore's BASIC findes i mange udgaver. Den version, der anvendes på Commodore 64, kaldes version 2.0, og er en af de tidligste udgaver. Commodore Plus/4 og C16 er udstyrede med en mere omfattende BASIC (version 3.5), der har store ligheder med den BASIC 4.0, som findes i Commodores professionelle maskiner.

Dette kapitel vil behandle de grundlæggende detaljer i Commodores BASIC-udgaver - hvordan tingene hænger sammen, hvilke variabler og talstørrelser, der kan anvendes osv. På disse punkter, optræder der stort set ingen forskelle mellem f.eks. Plus/4 og Commodore 64.

I appendix A findes en komplet oversigt over alle kommandoer og funktioner i Commodore BASIC 2.0 og BASIC 3.5. Kommandoer, som kun findes i BASIC 3.5, er mærket (+4), selvom betragtningerne også gælder for Commodore C16. For at forenkle tingene benyttes betegnelserne (+4) og Plus/4 i stedet for den mere langtrukne vending *Commodore Plus/4 og C16*. Eneste forskel mellem de to maskiner - set fra programmørens side - er hukommelsens størrelse. Plus/4 er udstyret med 64K RAM og C16 har 16K RAM hukommelse.

#### 3.1 Om RAM og ROM

Forkortelsen *ROM* står for *Read Only Memory* og anvendes som betegnelse for hukommelses-kredsløb med permanent lagrede informationer. Informationerne i denne form for hukommelse kan kun læses, og det er ikke muligt at lagre nye informationer i ROM-kredse.

I modsætning hertil kan man både lagre og læse informationer i RAM (Random Access Memory). Kredsene har dog den ulempe, at de taber alle informationer, når der slukkes for strømmen til computeren.

De programmer og data, vi selv fremstiller, lagres i RAM, og det er disse informationer vi overspiller til kassettebånd eller disketter, hvis vi ønsker at bevare dem til brug ved en senere lejlighed.

ROM-kredsenes indhold forsvinder derimod ikke, når vi slukker for strømmen. Disse kredse anvendes derfor til lagring af de informationer, som styrer hele computeren - eller rettere computerens microprocessor. Uden disse informationer, som kaldes maskinkode, ville det være meget svært at

programmere en computer.

Computerens BASIC er i grunden ikke andet end et avanceret maskinkodeprogram, der oversætter de kommandoer, du indtaster, til et sprog, computerens processor kan forstå. Dette sprog - eller maskinkode - er meget primitivt, men udføres til gengæld meget hurtigt i processoren (læs mere herom i kapitlet **Programmering i maskinkode**).

### 3.2 Tal og beregninger

Langt de fleste computere regner i det binære tal-system (se afsnittet **Om talsystemer**). Dvs. at de decimal-tal, vi anvender i vore BASIC-programmer, skal omdannes til binære tal, som computeren anvender internt i alle beregninger.

Så længe der er tale om hele tal, volder konverteringen til og fra det binære tal-system ingen problemer. Er der derimod tale om decimaltal opstår der konverteringsfejl, idet mange decimaltal giver *uendelige brøker* i det binære talsystem - akkurat som tallet  $\frac{1}{3}$  er en uendelig brøk i titals-systemet!

Af praktiske årsager er vi nødt til at begrænse antallet af cifre, som computeren anvender til interne beregninger - ellers kunne vi hurtigt løbe ind i problemer - både med hukommelsesplads og med regnehastighed.

I Commodores computere begrænses alle binære tal til 32 cifre, svarende til 4 bytes - eller 9 decimalcifre:

maksimalt: 11111111111111111111111111111111

Udover disse cifre lagres en eksponent, eksponentens fortegn og selve tallets fortegn. Dette kan rummes i et byte. Totalt benyttes altså 5 bytes til at repræsentere decimaltal i området  $1.70141183 \times 10^{38}$  til  $2.93873588 \times 10^{-38}$ , som er det mindste tal - bortset fra 0, som computeren kan arbejde med. Disse værdier gælder naturligvis både for positive og negative tal!

Når du foretager beregninger - uanset type - skal du være opmærksom på at computeren ofte regner en ganske lille smule forkert! Det skyldes som sagt, den konvertering, der sker mellem computerens tal-system og vort talsystem, samt de afrundinger computeren kan være nødt til at foretage i "uendelige brøker" - vi skriver jo heller aldrig brøken  $\frac{1}{3}$  heeeelt ud som decimaltal - af praktiske årsager!

Det er især vigtigt, at have disse forhold i tankerne, når du foretager sammenligninger - se afsnittet **Sammenligninger senere** i dette kapitel.

### 3.3 Variabler

Når vi skal foretage beregninger har vi brug for, at fortælle computeren, hvilke tal den skal bruge. Vi kan selvfølgelig bruge computeren som en lommeregner, og indtaste formler og tal hver gang, men vi kan også fremstille et program.

Når vi fremstiller et program, skal computeren lagre værdierne i programmet og de værdier, vi indtaster, så den kan finde dem igen. Til det formål, skal disse værdier udstyres med et navn.

Commodores computere tillader, at vi giver vore informationer - også kaldet variabler - et navn af vilkårlig længde. Eneste betingelse, der skal være opfyldt, er:

- Første tegn i navnet skal være et bogstav fra A til Z.
- Alle følgende tegn skal enten være bogstaver fra A til Z, eller cifre fra 0 til 9.
- Variabelnavne må ikke indeholde navne, der anvendes til kommandoer, funktioner m.m. F.eks. er navnene SUBTOTAL (TO), KONTO (ON og TO), START (ST) osv. ikke tilladte!

Selvom et variabelnavn kan have en vilkårlig længde, så anvender computeren kun de to første tegn i variabelnavnet! Computern skelner således ikke mellem variabelnavnene SLUT, SL og SL1, men variabelnavne, som S, S1 og SL kan skelnes uden problemer.

Computeren er også i stand til at skelne mellem flere forskellige variabeltyper (se også afsnittet Arrays - en særlig form for variabler senere i kapitlet):

- Numeriske variabler - ingen specielle kendetegn. F.eks. A, MOMS osv.
- Heltals (integer) variabler, som kan rumme hele tal i området +32767 til -32768. Variablerne kendetegnes med et "%" -tegn efter navnet - f.eks. A%, MOMS% osv.
- Tekst (streng) variabler, som kan rumme op til 255 tegn af enhver type. Variablerne kendetegnes med et "\$" -tegn efter navnet - f.eks. A\$, MOMS\$ osv.

Bemærk, at de grænser, der gælder for variabelernes størrelser, også gælder for mellemresultater i en udregning. Har du f.eks. to strengvariabler X\$ og Y\$, hver på 128 tegn, vil følgende udregning give en fejlmelding:

```
A$=LEFT$(X$+Y$,150)
```



selvom resultatet er indenfor det gyldige område!!!

Heltalsvariabler kan i sig selv ikke rumme værdier udenfor området 32767 til -32768, men mellemresultater må gerne være større eller mindre. Blot skal *slutresultatet* holdes indenfor det tilladte. Således er:

```
XX=32000:YX=31000:AX=XX*YX/64000
```

fuldt ud tilladt! Det skyldes, at Commodores computere ikke regner med hele tal internt, selvom det ville være hurtigere. Alle heltalsvariabler omdannes til *floating point* eller decimaltal med eksponenter, *inden* en udregning foretages, og *resultatet* omdannes så igen til en heltalsvariabel! Kun hvis resultatet af en beregning er udenfor det tilladte område for heltalsvariabler, vil man få en fejlmelding.

Første gang en variabel benyttes, tildeles den, medmindre andet specificeres, værdien 0 (numeriske eller integer variabler) eller længden 0 (streng-variabler) - dvs. at strengen ikke indeholder nogen tegn. F.eks. vil programlinjen:

```
A=23:A$="ABC":PRINT A,B,LEN(A$),LEN(B$)
```

give udskriften:

```
23      Ø      3      Ø
```

på skærmen.

Når en variabel har været benyttet een gang, findes der ingen enkle metoder til at slette den fra variabelhukommelsen igen! Eneste mulighed er, at slette *hele* variabelhukommelsen med kommandoen CLR, og så genoprette de variabler, man skal bruge i resten af programmet - en ikke særlig praktisk løsning!

Udover CLR-kommandoen, sletter også RUN og NEW indholdet i variabelhukommelsen. NEW-kommandoen har dog den ulempe, at den også sletter *programmet*!!! Så vær meget opmærksom, når du indtaster noget, der begynder med NE, i computeren!!!

Er uheldet sket, er der dog stadig mulighed for at redde programmet. Du skal blot indtaste følgende linjer, direkte fra tastaturet og *UDEN* linjenumre:

```
POKE PEEK(43)+256*PEEK(44)+1,PEEK(44)
Noter resultatet af: PRINT PEEK(43)+256*PEEK(44)
Noter resultatet af: PRINT PEEK(55)+256*PEEK(56)-3
FORX= værdi1 TO værdi2 : IF PEEK(X) OR PEEK(X+1) OR PEEK(X+2) THEN NEXT
POKE 46, (X+3)/256: POKE 45, (X+3)-256*PEEK(46)
CLR
```

Nu kan programmet LIST'es igen! Selv om du nu ved, hvordan man redder et program efter NEW, er der ingen grund til letsindighed, vel?

### 3.4 Arrays - en særlig form for variabler

Arrays er en særlig form for variabler, der består af et antal elementer, som er organiseret i en bestemt struktur, under eet fælles navn. Også arrays findes i tre forskellige former, der kan skelnes indbyrdes af computeren, ligesom computeren er i stand til at skelne almindelige variabler fra array-variabler.

Navngivningen følger de samme regler, som almindelige variabler; det samme gælder for de værdier, der kan lagres i hvert element i et array (se foregående afsnit).

Før computeren kan anvende et array, en matrix eller en dimensioneret variabel - kørt barn har mange navne - skal den *dimensioneres*. Dvs. at computeren skal advares om, hvor megen plads, der skal reserveres til array'et. Det sker ved en DIM-kommando, f.eks. således:

DIM A(2,3)

DIM-kommandoen, reserverer plads til 12 elementer i det array, der bærer navnet A. Elementerne er:

A(0,0) - A(0,1) - A(0,2) - A(0,3)  
 A(1,0) - A(1,1) - A(1,2) - A(1,3)  
 A(2,0) - A(2,1) - A(2,2) - A(2,3)

Array'et siges at indeholde 2 *dimensioner* (2 og 3), som tilsammen indeholder 12 elementer fra A(0,0) til A(2,3). Et array kan maksimalt indeholde 255 dimensioner og 32767 elementer. F.eks. vil array'et X(2,2,2) have fire dimensioner bestående af ialt 3\*3\*3\*3 eller 81 elementer fra X(0,0,0,0) til X(2,2,2,2).

I praksis tillader Commodore's BASIC, at man anvender array's, som maksimalt indeholder 11 elementer (0-10) i hver dimension - uden at man skal dimensionere array'et forud. Selvom det umiddelbart forekommer overflødigt, kan det varmt anbefales *altid* at dimensionere array's forud for deres brug (v.hj.af DIM). Så slipper man for uventede fejl i sine programmer. Fejl som kan være meget svære at finde, når der er gået et par måneder!

## 3.4.1 Arrays og hukommelse

Når et array dimensioneres, afsættes der plads i hukommelsen til de senere data, der skal indlæses. Om array'et dimensioneres via en DIM-sætning, eller når programmet møder en array-variabel i programmet for første gang, ændrer intet i de pladskrav, der stilles. Blot vil en dimensionering via en DIM-kommando normalt gå hurtigere, medmindre BASIC'en møder det største array element først. I det tilfælde er der ingen forskel. Altså er f.eks.

```
DIM A(10,10,10)
```

og

```
A(10,10,10)=0
```

lige hurtige og lige pladskrævende, set fra computerens side.

De krav til hukommelsen, som arrays stiller, kan være meget store, selvom det umiddelbart ikke virker sådan. I tabellen herunder, finder du en opgørelse over de pladskrav, der stilles.

Array-type	pladskrav
<hr/>	
Integer	array-navn 5 bytes pr.dimension 2 bytes pr.element 2 bytes formel: $5+2 \times \text{dimensioner} + 2 \times \text{elementer}$ eksempel: DIM AB%(2,3,4,5) $5+2 \times 4 + 2 \times (3 \times 4 \times 5 \times 6) = 733 \text{ bytes}$
<hr/>	
Numerisk	array-navn 5 bytes pr.dimension 2 bytes pr.element 5 bytes formel: $5+2 \times \text{dimensioner} + 5 \times \text{elementer}$ eksempel: DIM AB(2,3,4,5) $5+2 \times 4 + 5 \times (3 \times 4 \times 5 \times 6) = 1813 \text{ bytes}$
<hr/>	
Streng/tekst	array-navn 5 bytes pr.dimension 2 bytes pr.element 3 bytes+tegn formel: $5+2 \times \text{dimensioner} + 3 \times \text{elementer}$ eksempel: DIM AB\$(2,3,4,5) $5+2 \times 4 + 3 \times (3 \times 4 \times 5 \times 6) = 1093 \text{ bytes}$ +summen af alle teksterne
<hr/>	

I eksemplet herover, afsættes der 1093 bytes til et streng-array, før vi overhovedet har læst et eneste tegn ind i det. Teoretisk set kan arrayet AB\$(2,3,4,5) rumme 255 tegn i hvert element, men i praksis vil der opstå et lille problem - computerens hukommelse er ikke stor nok! Det ville nemlig kræve  $1093+255 \times 360 \text{ bytes} = 92893 \text{ bytes}$  (360 elementer), og det er jo lidt i overkanten af den plads, vi har til rådighed. Til sammenligning er her

pladskraverne for almindelige variabler:

#### Variabel-type pladskrav

---

Integer	navn: 2 bytes, data 5 bytes
Numerisk	navn: 2 bytes, data 5 bytes !!!
Streng/tekst	navn: 2 bytes, data 5 bytes + 1 byte pr. tegn

---

Bemærk, at modsat mange andre BASIC-dialekter, spares der ingen plads, når man anvender integer-variabler, og da computeren også er længere tid om at udføre beregninger med integer-variabler, grundet den konvertering til og fra *floating point*, der foretages (se side 35), er der *ingen* fornuftig grund til at anvende *integer variabler* i dine programmer!

Med *integer arrays* stiller sagen sig dog lidt anderledes, da man virkelig kan spare en hel del plads - i eksemplet forud blev der sparet 1080 bytes. *Integer arrays* er altså værd at have i tankerne, hvis *pladshensyn* vejer tungere end *hastighed* - og det er jo tilfældet nu og da!

### 3.5 Matematiske udtryk

Alle matematiske udtryk udføres fra venstre mod højre efter de normale regneregler. Paranteser kan benyttes, hvis man ønsker at ændre denne rækkefølge. F.eks. som i dette eksempel:

udtryk	$1+10+27*2$	$1+(10+27)*2$	$(1+10)+27*2$
1.trin	$11+27*2$	$1+37*2$	$11+27*2$
2.trin	$11+54$	$1+74$	$11+54$
3.trin	65	75	65

Den rækkefølge processerne udføres i - f.eks. multiplikation før addition - bestemmes af den vægt, hvert enkelt led i udtrykket tillægges. Denne vægt eller betydning er fastlagt i en tabel, som kaldes et *operator-hieraki*. Denne tabel fortæller computeren, hvilke udregninger, der skal udføres forud for andre, hvor der er valgmuligheder. På denne måde sikres det, at de regneregler, som vi alle kender eller burde kende, overholdes. Operator-hierakiet ser således ud:

- 1 - Funktioner - f.eks. SQR(X), SIN(X) etc.
- 2 - Negation (fortegnet minus)
- 3 - Multiplikation og division (\*, /)
- 4 - Addition og subtraktion (+, -)
- 5 - Relationer (>, <, =, <=, >=)
- 6 - Logisk NOT - f.eks. NOT K%
- 7 - Logisk AND - f.eks. B AND 127
- 8 - Logisk OR - f.eks. B OR 3

Operatorer med samme vægt - f.eks. plus og minus - udføres fra venstre mod højre, medmindre parenteser bestemmer noget andet.

### 3.6 Sammenligninger og forgreninger

Sammenligninger eller relationer er et af de mest kraftfulde redskaber programmøren har til rådighed. De muliggør *betinget programmering* - dvs. at bestemte dele af programmet udføres, når en forud fastlagt betingelse er opfyldt. F.eks.:

```
IF A=1 THEN PRINT "DET VAR GODT"
```

Bemærk, at lighedstegnet i dette tilfælde har en anden betydning, end i sætningen  $A=22*B$ . I første tilfælde er der tale om en *relations-* eller *sammenligningsoperator*, mens der i andet tilfælde er tale om en *tildelingsoperator*!

Når man skal foretage en sammenligning baseret på lighed (=) eller ulighed (<>), vil det ofte være en fordel, at teste i et *interval* i stedet for en test af en eksakt værdi, grundet de afrundingsfejl, der kan opstå i BASIC!!! F.eks. er:

```
IF ABS(A)<0.0001 THEN.. eller IF ABS(A)>0.0001 THEN..
```

at foretrække frem for:

```
IF A=0 THEN.... eller IF A<>0 THEN....
```

I første tilfælde udføres ordrerne efter THEN, hvis A er større end -0,0001 og mindre end 0,0001, mens den alternative udgave *kun* udføres, når A er lig med 0. Er værdien A baseret på mere eller mindre omfattende udregninger, kan der opstå afvigelse - måske ude på niende decimal - som gør, at værdien *aldrig* bliver nul i *praksis*, selvom værdien teoretisk set skulle være nul.

Programfejl af denne art kan være *ekstremt* svære at afsløre! Derfor kan man lige så godt vænne sig til *altid* at teste betingelser i et interval af passende størrelse. Det er faktisk ikke så få "uløselige" problemer, man undgår på

denne måde!

### 3.7 Logiske operatorer

Logiske operatorer anvendes til at sammenknytte flere *betingelser*, der skal være opfyldt, før en bestemt proces udføres. F.eks. som her:

IF A<2 AND B>3 THEN .....

eller

IF NOT(A=2) THEN....

Parantesen i det sidste eksempel er ikke strengt nødvendig (se *operator-hierarkiet* herover), men i mere omfattende udtryk kan det være en fordel at dele et udtryk op i mindre og mere overskuelige enheder.

Det er ikke unormalt at have problemer med forstå operator-hierarkiet, når der er tale om *logiske operatorer*. En god *HUSKE-REGEL* (OBS!!!) er at "oversætte" NOT til fortegnet *minus*, AND til *gange* og OR til *plus*. Så er det hele langt lettere at overskue. F.eks. i situationen:

IF A=B AND C=D OR NOT Q=D AND H=J OR NOT K=L THEN...

kan det ofte være en hjælp, at "oversætte" udtrykket til:

IF a \* c + (-d) \* h + (-k) THEN ....

når man skal finde ud af den *rækkefølge*, udtrykkene udføres i! At man så ofte havde gjort klogt i at anvende "rigeligt" med paranteser lige fra første færd, er en helt anden sag. F.eks. er dette udtryk eksakt det samme:

IF (A=B AND C=D) OR (NOT(Q=D) AND H=J) OR NOT(K=L) THEN...

og nu er der ikke nær den samme usikkerhed om rækkefølgen!

### 3.8 Binære operatorer

Når de logiske operatorer NOT, AND og OR anvendes i et udtryk, er der i

realiteten tale om binære operationer, internt i computeren (læs mere om det binære talsystem i kapitlet *Programmering i maskinkode*).

Først må vi vide, at udtrykket:

IF betingelse THEN ...

egentlig arbejder på grundlag af en simpel test på nul. Er betingelsen opfyldt, tildeles den værdien -1, og er betingelsen ikke opfyldt tildeles betingelsen værdien 0. Udtrykket efter THEN udføres, når betingelsen er *forskellig* fra nul. Dette medfører at sætningerne:

IF A<>Ø THEN.. og IF A THEN...

er *identiske* i deres virkning!

Alle binære udregninger internt i computeren foretages med 16 bit heltalsværdier. Disse værdier befinder sig i området -32768 til +32767. Noteret på binær form, fås den følgende sammenhæng:

```
0111 1111 1111 1111 = 32767
0000 0000 0000 0001 = 1
0000 0000 0000 0000 = 0
1111 1111 1111 1111 = -1
1111 1111 1111 1110 = -2
1000 0000 0000 0000 = -32768
```

Bemærk, at værdien -1 svarer til, at alle bits er sat i det binære format. Dvs. at en betingelse enten tildeles den binære værdi:

0000 0000 0000 0000 eller 1111 1111 1111 1111

Foretages f.eks. en sammenknytning af to betingelser med den logiske operator AND, svarer det i princippet til at foretage den binære AND-operation. De logiske operatoren OR og NOT har en analog virkning (sandhedstabeller findes i beskrivelserne af AND, henholdsvis OR og NOT i Appendix A).

Denne egenskab ved operatorerne AND, OR og NOT har især betydning, når vi arbejder med grafik på Commodore 64. En detaljeret gennemgang af teknikkerne findes i kapitlet om grafik senere i bogen.

### 3.9 Betingede hop på en anden måde

Normalt programmeres betingede hop ved hjælp af IF...THEN sætninger. Metoden er dog ikke lige effektiv i alle situationer. F.eks. kan der opstå

behov for at udføre forskellige subrutiner afhængigt af et tryk på en enkelt knap.

Er der kun tale om to evt. tre valgmuligheder, kan IF...THEN sætninger stadig være en god løsning - især hvis ASCII-koderne er "spredt". Er der tale om en vis sammenhæng - f.eks. tryk på funktionsknapperne F1, F3, F5 og F7, findes et mere effektivt alternativ; f.eks. kunne et program byde på følgende valgmuligheder:

```

990 PRINT CHR$(147);"VALGMULIGHEDER:":PRINT
1000 PRINT "F1 = SØG"
1010 PRINT "F3 = RET"
1020 PRINT "F5 = SLET"
1030 PRINT "F7 = STOP"
1040 PRINT:PRINT"INDTAST VALG"
1050 WAIT 203,63
1060 GET A$
1070 IF A$<CHR$(133) OR A$> CHR$(136) THEN 1050
1080 A=ASC(A$)-132
1090 ON A GOSUB 1200, 1400, 1600, 1800
1100 GOTO 990

```

Eksemplet er "lige ud ad landevejen", men hvad gør vi, hvis indtastningerne ikke har nogen egentlig sammenhæng - f.eks. bogstaverne A, D, H, K og W? IF...THEN metoden kan naturligvis altid benyttes, men en enklere og langt mere fleksibel metode findes:

```

700 SVAR$="ADHKW"
710 WAIT 203,63
720 GET I$:I=0
730 FOR N=1 TO LEN(SVAR$)
740 : IF I$=MID$(SVAR$,N,1) THEN I=N
750 NEXT N
760 IF I=0 THEN 710
770 ON I GOTO A-linje, D-linje, H-linje, K-linje, W-linje

```

Bemærk, at negative værdier i forbindelse med ON...GOTO/GOSUB giver fejlmeldingen ILLEGAL QUANTITY ERROR!

En helt speciel anvendelse af ON...GOTO/GOSUB udnytter det faktum, at værdien 0 (nul) medfører, at der ikke udføres nogen forgrening.

De fleste større programmer indeholder mange del- og subrutiner, hvis betydning og startlinjer kan være svære at finde - især, hvis man råder over et TOOLKIT eller BASIC-udvidelse med mulighed for renummerering af programmet. Efter udførelse af en RENUMBER kommando, tager det normalt et stykke tid, inden man igen er klar over rutinernes placering. Samme problem gør sig gældende med ældre programmer, man ønsker at ændre.

Følgende lille trick letter genfindning af bestemte rutiner betydeligt:



```

100 REM      CURSOR  TAST IND  TEST IND TAST
110 ON Ø GOSUB 2020, 2130, 2275
120 REM      SØG  RET  SLET
130 ON Ø GOTO 1000, 1250, 1470

```

Ingen af linjerne udføres, men placeres de i starten af programmet, er det aldrig noget problem, at genfinde starten af forskellige programdele (GOTO) eller subrutiner (GOSUB). Fordelen fremfor almindelige REM-sætninger er, at linjenumrene ændres i overensstemmelse med rutinernes startlinjer, når RENUMBER udføres (dog ikke i SIMONS BASIC, der ikke kan udføre en pålidelig renummerering!)

### 3.10 Løkker

Løkker eller loops anvendes overalt, hvor dele af et program skal udføres et forud fastlagt antal gange. F.eks. kan programmet:

```

100 A=1
110 PRINT A
120 A=A+1
130 IF A<100 THEN 110

```

med fordel erstattes af:

```

100 FOR A=1 TO 99
110 PRINT A
120 NEXT A

```

der udfører eksakt samme opgave. Løkkestrukturens generelle opbygning:

```

100 FOR variabel=startværdi TO slutværdi STEP trin
... program
270 NEXT (variabel)

```

forudsættes bekendt, og anvendelsen af løkker i programmer plejer sjældent at forårsage nævneværdige problemer - bortset fra eet punkt, og det er, når man vil forlade løkkerne *før* tiden!

Den gyldne regel om, at man aldrig må springe ud fra en løkke, er en regel, der er baseret på almindelig sund programmerings-fornuft. Grunden er den enkle, at de fleste computere lagrer løkke-værdierne i et særligt arbejdsområde, og kun har begrænset plads til rådighed i dette område. Springer man ud af løkken, efterlades disse værdier i arbejdsområdet, som langsomt fyldes med den slags "affald". Selvom Commodore 64 er temmelig tolerant på dette område, er der stadig en vis fornuft i sagen - især da det er en forholdsvis enkel sag at forlade en løkke på *civiliseret* vis:

```

100 FOR N=1 TO X
110 IF betingelse THEN N=X:GOTO 200
... program
200 NEXT N

```

Udviser Commodore 64 *forståelse*, hvis man ønsker at forlade en FOR-NEXT-løkke før tiden, bli'r maskineriet til gengæld helt hysterisk, hvis man springer ind i en løkke. Normalt fås fejlmeldingen NEXT WITHOUT FOR, men under uheldige omstændigheder kan programforløbet - afhængigt af "forhistorien" - tage en helt uventet vending. Overraskelser kan også opstå, hvis man hopper ud af en løkke, inden den er afsluttet!

Uanset al talen om fornuftig eller ikke fornuftig programmering, er der een ting, der taler imod at overtræde den gyldne "løkke-regel". Det kan være uhyggeligt svært at overskue, hvad der sker i praksis, og fejlfinding i et program kan lige pludselig ændre karakter fra at være en trivial affære til en større omgang hjernegymnastik! Men selvfølgelig - hvis du savner spænding her i livet, er der alle muligheder for at opnå det gennem "ulovlig" omgang med løkker!

### 3.11 Sådan laver du hurtigere programmer

Når man fremstiller et program, har man to helt klare ønsker. Det første er, at programmet skal være let at overskue og forstå, så man også ved en senere lejlighed, kan foretage rettelser og forbedringer, uden de store anstrengelser. Eksempel 1 viser, hvordan man kunne udforme en subrutine, så man også senere kan forstå, hvad der sker:

Eksempel 1:

```

1000 REM MOMS AF DE INDTASTEDE VÆRDIER
1010 REM TL=ANTAL INDTASTNINGER
1020 :
1030 SUM=0
1040 FOR N=0 TO TL
1050 : MOMS=0.22*INDLÆSNING(TL)
1060 : SUM=SUM+MOMS
1070 NEXT N
1080 RETURN

```

Eksempel 2:

```

1000 SU=0: FORN=0TOTL: SU=SU+0.22*IN(TL):NEXT:RETURN

```

Der er næppe nogen tvivl om, hvilket af de to eksempler, der vil være lettest at forstå om en måned eller tre. Problemet er blot, at det sidste eksempel

udføres langt hurtigere af computeren, og det er det andet ønske, man har til et program: *hastighed!*

De tricks og tips, der skildres i dette afsnit, kan forbedre dit programs regnehastighed ganske betydeligt, men anvender du disse teknikker, vil programmerne næsten være umulige at gennemskue, når der er gået et stykke tid. Træerne vokser nu engang ikke ind i himlen, og derfor kan følgende fremgangsmåde anbefales:

- Start med at fremstille et program, der er så overskueligt opbygget, som overhovedet muligt (eksempel 1).
- Når programmet virker efter hensigten, gemmes dette, som originalen.
- Nu fremstilles en kopi, der er rettet, så den indeholder alle de tidsbesparende tricks, der overhovedet kan anvendes. Denne udnævnes til arbejdskopien.
- Når der skal foretages ændringer, skal disse foretages i *originalen* eller en kopi af denne. *Speed-up* teknikerne anvendes på en *kopi* af dette nye program.

Dette er den  *eneste* måde at opnå alle fordelene på! Som sagt, vil en gennemført anvendelse af de følgende *speed-up* teknikker i næsten alle tilfælde føre til, at dit program bliver nærmest helt uoverskueligt og ekstremt vanskeligt at rette! Gevinsten er op imod en halvering af udførelsestiden - afhængigt af programmets art!

#### SADAN ØGER DU PROGRAMMETS HASTIGHED:

- Erstat alle integer variabler og integer arrays med numeriske variabler og arrays. Er der problemer med pladsen, så begynd med de variabler, der benyttes indeni løkker. Eneste undtagelse, er de variabler, der skal anvendes i forbindelse med PEEK og POKE, samt de *binære* operatorer AND, OR og NOT! I disse tilfælde, er det en fordel, at variablerne allerede er lagret som integer værdier. I alle andre tilfælde fører integer variabler til øget tidsforbrug!
- Erstat alle ASCII-tal, der benyttes i løkker, med variabler. Konverteringen fra vore tal, til computerens interne form, tager tid - hver gang. F.eks.:

```
123 A=23:FOR N=1 TO 7:B(N)=A:NEXT
```

og ikke:

```
123 FOR N=1 TO 7:B(N)=23:NEXT
```

I udgave 1 foretages konverteringen fra ASCII til internt format kun een gang, men hele 7 gange i udgave 2.

- Erstat alle hyppigt anvendte talværdier, som f.eks. 0,1,2 osv. med variabler - f.eks. NU,EN,II,TR,FI,FE,SE,SY,OT....osv.

- Anvend variabelnavne med kun et bogstav, hvor det overhovedet er muligt! Er der ikke bogstaver nok i alfabetet, så foretag en vurdering af, hvilke variabler, der benyttes hyppigst i dit program. F.eks. benyttes variablen AN 200 gange i det følgende eksempel:

```
122 AN=22:FOR N=1 TO 100:BX(N)=AN:CG(N)=AN:NEXT N
```

Her er der virkelig noget at hente ved at ændre navnene fra BX, CG og AN til f.eks. B, C og A!

- Anvend almindelige aritmetiske udtryk i stedet for simple udgaver af funktioner. Således udføres A\*A hurtigere end A^2.
- Fjern alle mellemrum og REM-sætninger i programmet!
- Lav programlinjerne så lange, som muligt. Ved at anvende forkortet indtastning af alle kommandoer og funktioner (f.eks. ? i stedet for PRINT osv), kan der blive plads til flere tegn på en linje. Bemærk, at du kun kan indtaste 80 tegn i en linje - uanset metoden - men hver linje kan fuldt lovligt have en længde på indtil 255 tegn! En linje, der rummer mere end 80 tegn, når den LIST'es, kan kun rettes, ved at foretage en helt ny indtastning af programlinjen!
- Anvend så små linjenumre, som muligt! F.eks. vil GOTO 230 tage længere tid at udføre end GOTO 23!
- Erstat alle forekomster af værdien nul (0) med et punktum. F.eks.:

```
11 X=.
```

i stedet for

```
11 X=0
```

Selvom det ser sygt ud, så virker det!

- Undgå strengvariabler, strengfunktioner etc. overalt, hvor det er muligt. Hvis A\$ f.eks. rummer tegnene "SLØVT", vil:

```
PRINT"SLØVT" være hurtigere end PRINTA$
```

ligesom

```
PRINT"A" vil være hurtigere end PRINTCHR$(65)
```

I det sidste tilfælde er tidsbesparelsen op imod 50 procent!!!

- Undgå unødigt brug af strengvariabler, hvor det er muligt. F.eks. vil programlinjen:

```
11 X$="":FOR N=1 TO 40:X$=X$+"A":NEXT
```

skabe en masse "affald" (på engelsk: *garbage*) i det område af hukommelsen, som strengene lagres i. Når området er fyldt - selvom der kun anvendes ganske få strenge!!! - udløses automatisk en *garbage*

*collection*, som fjerner "affaldet" og igen skaber orden i denne del af hukommelsen. Som grundregel kan man gå ud fra, at hver gang, der læses noget fra en strengvariabel over i en anden eller strenge adderes, så skabes der *garbage*. Det skyldes, at computeren ikke genanvender den plads, der blev anvendt af en strengvariabel, før den fik tildelt en ny værdi. I stedet fyldes de nye værdier bare ind i den tiloversblevne hukommelse. Først, når der ikke er mere plads til rådighed, vil computeren rydde op i det rod, den selv har skabt. Da computeren nu er nødt til at undersøge hver enkelt strengvariabel, for at finde de "ubrugte" dele af strenghukommelsen, er det en tidskrævende proces! I uheldige tilfælde kan en sådan *garbage collection* tage helt op til 3 til 5 minutter. Imens sker der overhovedet ingenting på skærmen - du får ikke engang en melding, der fortæller, at computeren er ved at rydde op! Derfor tror mange, at computeren er død, afbryder programmet - med de skrappe midler - og spilder timer på at finde en fejl, som altså slet ikke eksisterer.

- Anvend altid NEXT uden efterfølgende variabel, i stedet for f.eks. NEXT N osv.
- Benyttes IF...THEN...sætninger, opbyg da programmet så den betingelse, der er hyppigst opfyldt, fører til, at næste linje udføres. Det tager nemlig længere tid, at udføre THEN-delen af sætningen, end at springe til næste linje!
- Anvend *lagdelte* IF...THEN...sætninger i stedet for lange udtryk med *logiske operatoren*. Sætningen:

IF (A=2) AND (B=3) AND (C=4) THEN...

kan omdannes til:

IF C=4 THEN IF A=2 THEN IF B=3 THEN....

Dette eksempel forudsætter, at betingelsen B=3 opfyldes hyppigst, at A=3 opfyldes mindre hyppigt, og at C=4 kun sjældent opfyldes (mellemmommene er kun medtaget for at gøre tingene mere overskuelige).

- Placer alle subrutiner *først* i programmet. Når computeren møder kommandoen GOSUB *linjenummer*, starter den jagten på det ønskede linjenummer fra begyndelsen af programmet. Placer de hyppigst anvendte subrutiner allerførst. F.eks. vil program-strukturen:

```
1      ....
      hovedprogram
113 første subrutine
```

være langsommere end:

```
1 GOTO 67
2 første subrutine
67 hovedprogram
```

Prøv metoderne i praksis på et af dine egne programmer, og sammenlign tiderne for udførelsen. I visse (særligt heldige) tilfælde kan programmets

## Commodore BASIC

hastighed mere end fordobles!

## Kapitel 4

### Programmering i maskinkode

Maskinkode er det sprog processoren forstår. Det har intet med BASIC, COMAL, FORTH, PASCAL eller noget andet *højniveausprog* at gøre. I princippet er der ikke tale om andet end en kombination af et-taller og nuller, som for hver enkelt kombinations vedkommende udløser en ganske specifik proces i mikroprocessoren.

Maskinkode udføres ekstremt hurtigt, sammenlignet med den indbyggede BASIC i din Commodore 64 computer. En instruktion tager typisk mellem 2 og 7 milliontedel sekund at udføre - til gengæld er instruktionerne uhyre primitive, sammenlignet med BASIC.

Lad det være sagt med det samme:

**Har du svært ved at forstå BASIC, er det tidsspilde at gå i krig med maskinkode!**

Ikke fordi maskinkode er vanskeligt, men fordi processorens sprog er så primitivt, som det er. Processoren udfører *alle* instruktioner ukritisk! Der er *ingen* fejlmeldinger eller hjælp at hente. Derfor stilles der meget høje krav til programmørens omhyggelighed.

I BASIC er der ikke noget i vejen for at indtaste et program, bare for at se, om det nu virker efter hensigten. Dvs. at overlade det til computeren at afsløre evt. bommerter! Den udvej findes *ikke*, når der arbejdes med maskinkode - i hvert fald ikke, hvis man ikke råder over en række hjælpemidler, som normalt ikke er tilgængelige på hobbymarkedet.

Principielt findes kun to muligheder, når man programmerer i maskinkode. Enten virker programmet, og så er fejlene normalt meget små, eller også virker det ikke, og så opfører computeren sig helt hysterisk. I de fleste tilfælde nægter den helt at samarbejde, og i de mere sjældne tilfælde kan man få helt fantastiske lys- og lydeffekter.

Trods alle ulemperne bør fordelene ikke overses. Maskinkodeprogrammer er ekstremt hurtige, når de endelig virker. Den enorme hastighed er også medvirkende til, at selv de mest bagvendte programkonstruktioner udføres relativt hurtigt. Dette punkt gavner især begynderen, som udelukkende kan koncentrere sig om at få programmerne til at virke efter hensigten. Et stort plus, når man tænker på, at det tager endog meget lang tid, at mestre de mere avancerede finesser og programmeringstrick - og så er 6510 (6502) processoren endda en af de mere primitive processorer på markedet.

### 4.1 Hjælpemidler i programmeringen

Hvis man ønsker at stifte bekendtskab med 6502/6510 maskinkode, er det meget vigtigt, at man råder over de mest nødvendige hjælpemidler og redskaber. Det vigtigste er at købe en bog, der i dybden forklarer, hvordan man programmerer i maskinkode. Der findes mange bøger om emnet, men kun få er virkeligt anvendelige. Blandt de bedste (efter forfatterens mening) er:

*6502 Assembly Language Programming* af Lance A. Leventhal fra forlaget Osborne/McGraw-Hill.

Evt. suppleret med den følgende bogtitel, som indeholder en lang række nyttige rutiner for den mere seriøse maskinkodeprogrammør, der ikke ønsker at opfinde hjulet een gang til:

*6502 Assembly Language Subroutines* af Lance A. Leventhal og Winthrop Saville - samme forlag.

Bøgerne er af meget høj standard, hvilket naturligvis medfører, at de ikke er helt billige. Førsteklasses faglitteratur koster nu engang.

Udover litteratursiden er der også andre ting at have i tankerne. Her er to forslag til udstyr:

#### TIL DEN NYSGERRIGE

Det kan naturligvis lade sig gøre, at fremstille helt perfekte maskinkodeprogrammer, alene ved hjælp af blyant og papir - jeg kender blot ingen, der har gjort det! Ønsker du ikke, at "spilde" mere tid end højst nødvendigt, bør udstyret omfatte:

- Monitorprogram
- Kassetdebåndoptager (helst disk-station)
- Printer til udskrifter

#### TIL DEN "SERIØSE"

Den seriøse bruger ønsker naturligvis maksimalt udbytte af sin indsats. Ofte er der behov for at fremstille større programmer - mere end nogle få hundrede bytes - og programmeringen skal være så bekvem, som muligt, for at undgå unødigt spildtid. F.eks. kan en rimeligt dokumenteret assemblerkode til et 4K maskinkodeprogram sagtens fylde 100K eller mere. Derfor bør udstyret omfatte:

- En god monitor - af hensyn til programmørens velbefindende.
- En disk-station - af hensyn til produktionshastigheden. Gerne to.
- En god printer, som også er i stand til at producere Commodores



specialtegn. Printerens skal være så hurtig, som økonomisk muligt. En listning af omfattende programmer kan let fylde 50 til 100 sider, så en printer, der arbejder med 25 til 50 karakterer pr. sekund, er ikke noget at råbe hurra for.

- En god assembler og en god maskinkode-monitor.

I næste kapitel kan du se, hvad en maskinkode-monitor er - i dette tilfælde er der tale om den monitor, der er indbygget i Commodore Plus/4. Vælg en af tilsvarende kvalitet til din Commodore 64.

En assembler er et program, som tillader programmøren at indtaste sine programmer i form af *Mnemonics*. Mnemonics er specielle betegnelser for de enkelte maskinkode-operationer, som gør det let for os mennesker, at huske betydningen. F.eks. dækker de følgende tre udtryk samme maskinkode operation:

01001000	maskinkode
48	hex-kode
PHA	assembler-kode (mnemonic)

Der er næppe tvivl om den form, der er lettest at huske, vel?

Når du vælger assembler, skal du være klar over, at der findes mange forskellige typer, og ligesom i alle andre situationer her i livet, er en god assembler ikke helt billig, men husk, at en høj pris ikke er en garanti for kvalitet. Suk!

Medmindre du allerede har erfaring med assemblere fra andre computere, bør du tage en snak med Commodore's brugerklub eller en erfaren maskinkodeprogrammer, inden det endelige valg træffes. Der findes ganske enkelt så meget "skrammel" på markedet, at det vil være helt umuligt at give generelle retningslinjer. Sørgeligt, men sandt.

## 4.2 Om talsystemer

Før man går i gang med at programmere en computer i maskinkode, kan det være en fordel, at sætte sig ind i de grundlæggende begreber, som bits og bytes, det binære talsystem og det hexadecimalt talsystem. Det gør livet lettere i det lange løb.

### 4.2.1 Det binære talsystem

Mikroprocessorens sprog består af en kombination af et'er og nul'er. Afhængigt af kombinationen udføres forskellige mikro-processer, der i kombination med andre mikro-processer f.eks. kan føre til, at bogstavet "A"

dukker op på TV-skærmen.

Grundenheden i denne kode kaldes et *BIT*. Et bit, er en hukommelsesplads, som kan indeholde een og kun een information - f.eks. 0 eller 1. Det kan sammenlignes med en pære - f.eks. i din skrivebordslampe - som enten kan være tændt eller slukket. Et relæ, som enten kan bryde eller slutte en kontakt. En transistor, som enten kan spærre eller lede en strøm. Osv.

Disse enten/eller informationer samles normalt til større enheder af praktiske årsager. Mikroprocessoren i din computer, kan læse eller lagre otte bits på een gang. Denne mængde kaldes et *BYTE*. Otte bits rummer ialt  $2^8$  eller 256 kombinationsmuligheder. I daglig tale oversætter vi det til at en mikroprocessor kan benytte talstørrelser (internt) fra 0 til 255 - en mere korrekt sammenligning ville måske være at tale om otte lyskontakter, som enkeltvis kan være slukkede eller tændte. Hvis vi anvender "0" for en slukket kontakt og "1" for en tændt kontakt, kunne vi opstille en tabel, der så således ud (vi nummererer kontakterne fra 0 til 7):

kombinationsnr.	kontakter	virkning
0	00000000	alle slukkede
1	00000001	kontakt 0 tændt
2	00000010	kontakt 1 tændt
3	00000011	kontakt 0 og 1 tændt
4	00000100	kontakt 2 tændt
5	00000101	kontakt 0 og 2 tændt
6	00000110	kontakt 1 og 2 tændt
7	00000111	kontakt 0, 1 og 2 tændt
8	00001000	kontakt 3 tændt
16	00010000	kontakt 4 tændt
32	00100000	kontakt 5 tændt
64	01000000	kontakt 6 tændt
128	10000000	kontakt 7 tændt
255	11111111	alle kontakter tændt

Dette, at en kontakt kan have to stillinger, tilstande eller værdier, kaldes binære tilstande eller værdier, og den måde, vi har bygget vort kontaktsystem op på, svarer egentlig til det binære talsystem.

I det binære talsystem har vi kun to cifre, nemlig 0 og 1, og alle udregninger er baseret på dette faktum! Vi kan foretage alle normale matematiske udregninger, akkurat på samme måde som i titals-systemet. F.eks. kan vi lægge to tal sammen:

0	1	0	1
+ 0	+ 0	+ 1	+ 1
----	----	----	----
0	1	1	10
====	====	====	=====

Det sidste eksempel, kan måske kræve lidt forklaring. Når vi lægger 1 og 1 sammen, får vi i titals-systemet 2, men det ciffer eksisterer ikke i to-tals-systemet, eller det binære talsystem. Når vi får en værdi, der

overstiger det højeste ciffer i et tal-system (akkurat som i tital-systemet), sætter vi 1 i mente og tæller videre fra 0). F.eks. 9+1 i titals-systemet:

Ni plus en. Det kan vi ikke! Vi sætter en i mente, og tæller et ciffer frem fra 9 - det er nul. Vor tællerække ser egentlig således ud: 012345678901234567890123... osv.

I to-tals-systemet får vi følgende fremgangsmåde:

En plus en. Det kan vi ikke! Vi sætter en i mente, og tæller et ciffer frem fra en, det er også nul! Vor tællerække ser blot således ud: 0101010101010101010... osv.

Den teknik vi har udnyttet, er baseret på en særlig egenskab ved vort moderne talsystem - nemlig, at et ciffers placering i et tal også siger noget om værdien. F.eks. repræsenterer 9-tallet i de følgende tre tal tre forskellige værdier:

129 192 912

I det første eksempel, repræsenterer nitallet værdien 9, i det andet eksempel værdien 90 og i det sidste eksempel værdien 900! Positionen eller placeringen af et ciffer i et tal har altså også betydning.

Når vi fastlægger betydningen, tildeler vi hver position i et tal et nummer - fra 0 og opefter, som vist her:

tal	.....1111
-----	
position	.....3210

Værdien af et tal fastlægges da ud fra hvert ciffers position i tallet. Er der tale om et tal i titals-systemet, vil følgende udtryk give tallets værdi:

$$\text{værdi} = \dots 1 \times 10^3 + 1 \times 10^2 + 1 \times 10^1 + 1 \times 10^0$$

Kan du se sammenhængen? Grunden, til at vi benytter tallet 10 i eksemplet herover, er at grundtallet i titals-systemet netop er 10.

Den formel vi lige har benyttet, gælder for alle talsystemer, og i den mere generelle form ser formelen således ud:

$$\text{værdi} = \dots C_3 \times G^3 + C_2 \times G^2 + C_1 \times G^1 + C_0 \times G^0$$

hvor G er *grundtallet* i talsystemet, og  $C_0$  er cifferet i position 0,  $C_1$  er cifferet i position 1 osv. Denne sammenhæng kan vi udnytte, når vi skal omregne et tal fra et tal-system til et andet:

F.eks. vil tallet:

111 i femtals-systemet give værdien:

$$1x5^2+1x5^1+1x5^0=31 \text{ i titals-systemet}$$

111 i tretals-systemet giver værdien:

$$1x3^2+1x3^1+1x3^0=13 \text{ i titals-systemet}$$

og 111 i to-tals-systemet giver værdien:

$$1x2^2+1x2^1+1x2^0=7 \text{ i titals-systemet}$$

Benytter du denne fremgangsmåde, er det meget enkelt, at omregne tal fra et tal-system til et andet. Prøv f.eks. at omregne en 11110011, 10101, 11100011 og 10101010 i to-tals-systemet til et tal i titals-systemet (Resultaterne kan du kontrollere med program-eksemplet, som står under kommandoen DEF FN i Appendix A bag i bogen.

#### 4.2.2 Hexadecimale tal

De principper, vi lige har gennemgået, gælder naturligvis også for det hexadecimale talsystem - seksten-tals-systemet. Dette tal-system har grundtallet seksten, og det vil sige, at vi skal benytte 16 cifre. Det giver et lille problem, da vi jo kun råder over cifrene 0 til 9!

Problemet løser vi ved at forfremme de første seks bogstaver i alfabetet til cifre i seksten-tals-systemet. Det hele kommer så til at se således ud:

ciffer	værdi i titals-systemet
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Lad os prøve at omregne de hexadecimale tal 111 og 1CA til titals-systemet:

$$1 \times 16^2 + 1 \times 16^1 + 1 \times 16^0 = 273 \text{ i titals-systemet}$$

$$1 \times 16^2 + 12 \times 16^1 + 10 \times 16^0 = 458 \text{ i titals-systemet}$$

Prøv selv et par eksempler. Resultaterne kan kontrolleres med program-eksemplet i AND operatoren i Appendix A bag i bogen.

Mens begrundelsen for at kunne forstå det binære tal-system er klart baseret i processorens virkemåde og opbygning, er der ingen umiddelbar indlysende grund til at give sig til at lære det hexadecimale talsystem - skulle man tro!

#### 4.2.3 Hexadecimale tal - hvorfor?

Begrundelsen skal alene søges i menneskers dovenskab! Man kan spare uhyggelige mængder skrivearbejde ved at anvende hexadecimale tal - hex-tal mellem venner - i stedet for binære tal. Så snart vi skal udtrykke store tal i det binære tal-system, bliver udtrykket næsten uoverskueligt stort:

binært	hex	decimal
10101010	AA	170
11111111	FF	255
1000000000000000	8000	32768
1111111100000000	FF00	65280
1111000010100101	FOA5	61605
1111111111111111	FFFF	65535

Omregning fra binære tal til decimaltal er ikke ligefrem den mest enkle, men omregningen fra binære til hexadecimale tal, er meget enkel! Derfor benyttes hexadecimale tal! Omregningen er baseret på det faktum, at hver gruppe af fire binære cifre, svarer til eet hexadecimalt ciffer:

### binært hex decimal

0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Behovet for at lære disse to talsystemer er begrænset for de fleste computer-brugere - især hvis du er ejer af en Commodore 64. Køber du derimod en Commodore Plus/4, kan du få stort udbytte af at lære det hexadecimalt talsystem. Brugen af den indbyggede *MONITOR* lettes betydeligt, hvis du vænner dig til at anvende hex-tal.

### 4.3 Maskinkode og BASIC

Inden man fremstiller et maskinkode program, er det nødvendigt at træffe en række vigtige beslutninger - herunder hvor koden skal placeres, og hvordan vi kalder maskinkode programmet fra BASIC.

BASIC åbner to muligheder for at kalde et maskinkode program - hver med sine fordele og ulemper:

#### SYS - kommandoen

Kommandoen overfører kontrollen til et maskinkode program, der ligger på den specificerede adresse - formatet er:

#### SYS *adresse*

Adressen kan være en vilkårlig værdi mellem 0 og 65535 (hex FFFF). SYS-kommandoen kan *ikke* overføre værdier til maskinkode rutinen, og der kan heller ikke overføres værdier tilbage til BASIC. Problemet kan let omgås ved at lagre værdier, der skal overføres til BASIC på nærmere fastlagte adresser i hukommelsen - for derefter at læse disse værdier med PEEK-funktionen.

#### USR - funktionen

Funktionen kan overføre en numerisk værdi til maskinkode rutinen, og

det er også muligt at overføre en værdi til BASIC, når man returnerer fra maskinkoden. Anvendelsen er knap så enkel, som SYS-kommandoen; især volder det mange brugere store problemer, at styre parameteroverførslen. Maskinkode kaldet udformes således:

```
100 AD=maskinkodens adresse
110 POKE 784,76
120 POKE 785,AD-256*PEEK(AD)
130 POKE 786,INT(AD/256)
140 X=USR(Y)
```

Linje 110 er udelukkende medtaget, som en sikkerhedsforanstaltning! Dette POKE sætter maskinkode kommandoen JMP, som benyttes i alle funktionskald - når man eksperimenterer med maskinkode, er det uhyre let, at komme til at forandre dette byte, og så sker der ting og sager. Bemærk også linje 120. Mange programmer, der offentliggøres i forskellige computerblade anvender sætningen:

```
120 POKE 785,AD AND 255
```

i stedet. Denne version er ikke universel anvendelig, idet værdier over 32767 (hex 7FFF) giver fejlmeldingen ILLEGAL QUANTITY ERROR! Anvend den viste metode i stedet, og du undgår den slags problemer.

Variablen Y overføres til computerens interne *floating-point* akkumulator startende fra adresse 97 (hex 0061). Der er tale om en del af computerens lagerplads for interne beregninger. Når rutinen returnerer tilbage til BASIC igen, overføres værdien, der er lagret i dette område, til variablen X.

**OBS: POKE-kommandoerne skal altid stå  
umiddelbart før USR-kommandoen**

#### 4.3.1 Overførsel af variabler til maskinkode-rutine

Da indholdet i *floating-point* akkumulatoren er lagret i Commodores interne 5 byte *binær-decimale* format, kan den overførte værdi ikke anvendes umiddelbart i almindelige integer beregninger, som er det hyppigst forekommende arbejdsformat i de flestes maskinkode rutiner. Problemet kan dog løses forholdsvis enkelt, idet KERNAL ROM'en rummer en række konverteringsrutiner, som vi også kan anvende i vore egne maskinkode programmer. De vigtigste rutiner er:

```
JSR $B1AA
```

som konverterer indholdet i *floating-point* akkumulatoren til en hex-værdi mellem 32767 (7FFF) og -32768 (FFFF) - husk nul er lig med hex 0000. Ligger værdien udenfor området, fås fejlmeldingen ILLEGAL QUANTITY ERROR. Ønsker man en konvertering til *unsigned integer* i området 0 til FFFF (65535) anvendes i stedet rutinen:

```
JSR $B7F7
```

I begge tilfælde findes værdien lagret på adresserne:

\$64 - MSB (mest betydende byte)  
\$65 - LSB (mindst betydende byte)

Bemærk, at værdierne er lagret *modsat* det normale format for 6502/6510 processorer (LSB/MSB). I den sidstnævnte rutine overføres MSB til processorens A-register og LSB til processorens Y-register. Værdierne findes også lagret i normalt adresseformat på adresserne \$14/\$15 (LSB/MSB).

Tilbage læsning af integer-værdier fra maskinkode rutinen er lige så enkel. Først skal MSB placeres i processorens A-register og LSB i processorens Y-register, hvorefter den følgende rutine kaldes:

JSR \$B391

Resultatet placeres i *floating-point akkumulatoren*, og denne værdi returneres af *USR* - funktionen.

### 4.3.2 Matematiske beregninger i maskinkode

Det største problem, "hobbyfolket" har, er udførelse af matematiske beregninger i maskinkode. Så længe, der er tale om addition og subtraktion, burde der ikke være problemer, men kommer vi ind på multiplikation eller endnu værre - division - får piben en helt anden lyd.

Det er ingen spøg at udvikle multiplikations-rutiner og lignende; lykkeligvis er det heller ikke nødvendigt. Din Commodore computer råder over alle de rutiner, vi skal bruge - så hvorfor opfinde den dybe tallerken een gang til?

I det foregående afsnit fandt vi ud af, hvordan vi konverterer hex-tal til og fra *floating-point* formatet, som computeren anvender i *alle* sine normale BASIC-beregninger. Nu skal vi blot lære, at udnytte nogle små-rutiner. Alle vore beregninger vil foregå i *floating-point akkumulatoren* - i det følgende forkortet *FPA* - men vi får også brug for et ekstra arbejdsområde i vore beregninger - dette område kalder vi *ARG*.

I princippet er det uinteressant for praktikereren at vide, hvad der egentlig forgår i alle detaljer - bare han ved, hvad han skal "hælde ind" i de forskellige registre, hvilke rutiner, han skal kalde, og hvor han finder resultatet, er resten uinteressant. Akkurat som med en lommeregner. Man ved, hvad man skal gøre for at nå de ønskede resultater, men hvad der egentlig sker inde i regnemaskinen, interesserer ikke ret mange mennesker. Derfor vil læseren blive forskånet for alenlange - og i grunden uinteressante - beskrivelser af processerne. Vi koncentrerer os, om de "knapper, der skal trykkes på". Rutinerne er:

Datakonvertering:

JSR \$B1AA - FPA til integer (-32768/32767)  
JSR \$B391 - A/Y til FPA (5 bytes fra \$61)  
JSR \$B7F7 - FPA til integer (0-65535)  
JSR \$BC9B - FPA konverteres til integer



### Flytninger m.m.

A/Y-registrene (MSB/LSB) peger mod startadressen på de 5 bytes, som skal overføres til FPA, henholdsvis FPA og ARG. Rutinen kan benyttes til at indlæse de 5 værdi-bytes fra en numerisk variabel, eller til at overføre 5 bytes fra enten akkumulator 3 (start hex 0057) eller akkumulator 4 (start hex 005C).

```
JSR $B850 - FPA = konstant i A/Y-registre minus FPA (A/Y til ARG)
JSR $B867 - FPA = konstant i A/Y-registre plus FPA (A/Y til ARG)
JSR $BA28 - FPA = konstant i A/Y-registre gange FPA (A/Y til ARG)
JSR $BA8C - ARG = konstant i A/Y-registre
JSR $BB0F - FPA = konstant i A/Y-registre delt med FPA (A/Y til ARG)
JSR $BBA2 - FPA = konstant i A/Y registre
JSR $BBC4 - Akkumulator 4 = FPA
JSR $BBCA - Akkumulator 3 = FPA
JSR $BBFC - FPA = ARG
JSR $BC0C - ARG = FPA (FPA afrundes)
JSR $BF78 - FPA = konstant i A/Y-registre i potens FPA (A/Y til ARG)
```

### Beregninger:

```
JSR $B849 - FPA = FPA + 0.5
JSR $B853 - FPA = ARG - FPA
JSR $B86A - FPA = ARG + FPA
JSR $BA2B - FPA = ARG * FPA
JSR $BAE2 - FPA = FPA * 10
JSR $BAFE - FPA = FPA / 10
JSR $BB12 - FPA = ARG / FPA
JSR $BCLB - afrunding af FPA
JSR $BC9B - FPA konverteres til integer
JSR $BF7B - FPA = ARG ^ FPA (ARGFPA)
```

Med disse funktioner skulle det være muligt at foretage praktisk taget enhver beregning i egne maskinkode rutiner. Husk, at potens funktionen også kan anvendes til uddragning af rødder - f.eks.:

```
X1/2 kvadratroden af X
X1/3 kubikroden af X
```

Her er et lille eksempel, som sætter cursoren på skærmen - først BASIC-programmet:

```
100 LIN=linjenummer 0 til 24
110 KOL=kolonne 0 til 31
120 ADR=820
130 POKE 786,ADR/256
140 POKE 785,ADR AND 255
150 Y=USR(LIN*256+KOL)
```

Y-værdien, der returneres, har ingen praktisk betydning. Maskinkode programmet kan placeres frit i hukommelsen, men da rutinen kun fylder 8 bytes, er der faktisk plads til at placere den lige før kassettebufferen, fra adresse 820. Maskinkode rutinen ser således ud:

```

170: 0334          *= 820
;
190: 0334 20 F7 B7 JSR $B7F7 ;FPA læses til A- og Y-register
200: 0337 AA      TAX
210: 0338 18      CLC
220: 0339 4C F0 FF JMP $FFF0 ;Placer cursoren og retur!
    
```

## 4.3.3 Nyttige maskinkode rutiner i Commodores ROM

Som allerede nævnt i det foregående afsnit, er der ingen grund til at opfinde den dybe tallerken en gang til. Så her er flere maskinkode rutiner, som kan være til stor nytte i mange programmer.

adresse	ændrede registre	anvendelse - registre, der indeholder adresser har format (MSB/LSB)
R500	X og Y	Returnerer startadressen på CIAL i Y/X(\$DC00)
R505	X og Y	Returnerer skærmmformatet X linjer/Y kolonner
R50A	A	Carry=0 sætter cursor til X linjer/Y kolonner Carry=1 returnerer cursorens placering, som linje X og kolonne Y
R518	alle	Reset billedskærm
R544	alle	Slet skærm
R566	alle	Cursor HOME
R59A	alle	Initialisering af video-controller
R5B4	alle	Læs et tegn fra tastatur-buffer. Returnerer ASCII værdi i A og antal tegn+1 i X.
R632	A	Læs et tegn fra skærmen - returneres i A
R716	ingen	Send en ASCII karakter (i A) til skærmen
FCCA	A	Sluk for kassettemotor
FCE2	alle	Computer RESET
FB1C	A	Sæt STATUS til værdi i A
FFB7	A	Læs STATUS - returneres i A

Flere maskinkode-rutiner gennemgås i de følgende special-afsnit.

## 4.3.4 Ind- og udlæsning af data i maskinkode

Arbejder man jævnlgt med maskinkode, lærer man hurtigt, at påskønne Commodore's system med device-numre. Systemet er enkelt at anvende, og meget let at kombinere med BASIC, bare man følger nogle ganske få grundregler.

#### 4.3.4.1 Åbning af en kanal i maskinkode

Uanset hvilken kanal, man ønsker at kommunikere med, gælder følgende standardprocedure:

```
LDA #antal karakterer i filnavnet
LDY #MSB startadresse navn
LDX #LSB startadresse navn
JSR $FFBD
```

Navnet kan udelades til printere og kassette. I de tilfælde sættes A til 0 - X og Y registrenes indhold er underordnet i så fald.

```
LDA #logisk filnummer
LDX #devicenummer
LDY #kommando/kanalnummer - sekundær adresse
JSR $FFBA
```

I de tilfælde, hvor ingen sekundær adresse bruges, bør Y registeret sættes til værdien FF (255).

```
JSR $FFC0
```

Rutinen er identisk med den følgende BASIC kommando, der til enhver tid kan erstatte maskinkode versionen, om så skulle ønskes.

```
OPEN #,dev(,sa("FILNAVN"))
```

#### 4.3.4.2 Læsning fra en kanal i maskinkode

Først skal computeren have at vide, hvilket device vi vil tale med:

```
LDX #logisk filnummer
JSR $FFC6
```

Hvis denne rutine ikke har været kaldt forud, vil indlæsning automatisk ske fra tastaturet. For hver karakter, der skal indlæses, kaldes rutinen:

```
JSR $FFCF
```

Karakteren returneres i register A. Register Y forandres ikke af nogen af de to subrutiner og kan således anvendes som index-register ved indlæsning af flere karakterer. F.eks. som i dette eksempel, der indlæser 256 bytes fra den logiske fil 10:

```
                LDX #$0A
                LDY #$00
                JSR $FFC6
LOOP            JSR $FFCF
                STA BUFFER,Y
                JSR $FFB7    - læs STATUS
                BNE EXIT
                INY
                BNE LOOP
EXIT            RTS
```

### 4.3.4.3 Skrivning til en kanal i maskinkode

Ligesom i foregående rutine skal computeren have at vide, hvilket device vi vil tale med:

```
                LDX #logisk filnummer
                JSR $FFC9
```

Rutinen forandrer ikke indholdet i Y-registret. For hver karakter, der skal skrives, kaldes rutinen:

```
                JSR $FFD2
```

Denne rutine lader både X og Y registrene uberørte. Vær opmærksom på, at karakteren i A-registret sendes til alle aktive kanaler på een gang! Selvom det kan virke meget fristende - f.eks. at skrive til både printer og disk-station - kan der opstå problemer, når man læser STATUS, der ikke kan fortælle, hvilket device, der er "synderen". Ønskes dette ikke, må evt. åbne kanaler lukkes forinden med kommandoen:

```
                JSR $FFCC
```

som først de-aktiverer alle kanaler, og sætter *standard input* til tastatur og *standard output* til skærm. Hvis ingen kanaler er åbne, sendes karaktererne automatisk til skærmen. Eksemplet herunder demonstrerer, hvordan man f.eks. kan sende en linje på 80 karakterer til en printer (lf=3):

```
                LDX #$03
                JSR $FFC9
                LDX #$00
LOOP            JSR $FFB7 - check STATUS
                BNE EXIT
                LDA LINJESTART,X
                INX
                CPX #$50 - linjelængde
                BNE LOOP
EXIT            RTS
```

#### 4.3.4.4 Lukning af en kanal i maskinkode

Rutinen er helt identisk med BASIC CLOSE-kommandoen, og anvendes således:

```
LDX #logisk filnummer
JSR $FFC3
```

Ønsker man at lukke flere kanaler samtidigt, kan følgende subrutine kaldes:

```
JSR $FFE7
```

Den følgende rutine *de-aktiverer* alle åbne kanaler, men lukker dem *ikke*!

```
JSR $FFCC
```

De to sidste subrutinekald sætter *standard input* til tastatur og *standard output* til skærm.

#### 4.3.4.5 LOAD og VERIFY i maskinkode

LOAD kommandoen er kun tilladt i forbindelse med kassettebåndoptager og disk-station.

Et LOAD fra disk-stationen kan ske overalt i hukommelsen - også til BASIC og KERNAL ROM området. Karaktererne skrives automatisk til den underliggende RAM. I princippet kan det også lade sig gøre ved LOAD fra kassette, men i det tilfælde fås en LOAD ERROR fejlmelding. Det skyldes, at et program læses to gange fra kassettebåndoptageren. Første gang læses informationerne ind i RAM, og anden gang sammenlignes informationerne med den ekstra kopi på båndet - et VERIFY af programmets integritet. En sammenligning med indholdet i ROM'en vil naturligvis give en afvigelse - for at sige det mildt! Problemet kan omgås - se afsnittet LOAD program til under BASIC ROM

Fremgangsmåden er helt identisk med BASIC kommandoen LOAD:

```
LDA #logisk filnummer
LDX #device nummer
LDY #sekundær adresse
```

Den sekundære adresse kan være enten 0, 1 - hvis den sekundære adresse er 1, LOAD'es programmet absolut - dvs. til den adresse, der er indeholdt i *headeren*.

```
JSR $FFBA
LDA #antal karakterer i filnavnet
LDY #MSB startadresse navn
LDX #LSB startadresse navn
JSR $FFBD
```

Filparametrene er nu specificerede - om ønsket kan navnet undlades ved LOAD fra kassette. I det tilfælde skal A være 0, mens værdierne i X og Y

registrene ikke har betydning. Nu skal det afgøres, om der skal udføres et VERIFY (A=1) eller et LOAD (A=0).

```
LDA #LOAD/VERIFY flag - 0/1
```

Hvis sekundær adressen forud er fastsat til 0, skal vi nu specificere den adresse, hvortil LOAD skal ske - dette specificeres i X (LSB) og Y (MSB) registrene.

```
LDX #startadresse - LSB  
LDY #startadresse - MSB  
JSR $FFD5
```

Er der tale om indlæsning af maskinkode, kan vi slippe for fejlmeldinger, hvis vi "holder op" her. Er det et BASIC program, der indlæses, skal vi nu sætte end-of-program pointeren. LOAD-rutinen returnerer adressen på det første byte efter det sidst indlæste byte.

```
STX $2D  
STY $2E
```

### 4.3.4.6 LOAD program under BASIC ROM

Da det ikke er muligt, at LOAD'e et program fra kassette til området under BASIC ROM'en uden fejlmeldinger, må vi anvende en helt speciel fremgangsmåde, som vist i det følgende program. Er der tale om VERIFY af et disk-program, der "skjules" helt eller delvist af BASIC ROM, kan rutinen også benyttes.

```
LOADROM LDA $01  
PHA  
ORA #$02  
AND #$FE  
STA $01
```

BASIC ROM'en er nu udkoblet, og vi kan valgfrit bestemme os for LOAD eller VERIFY (husk sekundær adresse 1, hvis programmet ikke starter fra BASIC start).

JSR LOAD eller VERIFY rutinen fra foregående afsnit

Pointeren til end-of-program må ikke sættes!

```
PLA  
STA $01  
RTS
```

BASIC er nu igen indkoblet, og computeren virker normalt igen. Er det LOAD'ede program helt "dækket" af ROM, kan rutinen i afsnittet Lægning af maskinkode programmer benyttes til at kalde programmet.

#### 4.3.4.7 SAVE i maskinkode

I praksis er det muligt at SAVE programmer fra ethvert område op til adresse hex CFFF - både til kassette og til disk. Er programmet placeret, så det helt eller delvist dækkes af BASIC ROM'en, kan den følgende rutine anvendes til at koble BASIC-ROM'en ud, *inden* SAVE-koden kaldes.

```
SAVEROM  LDA $01
          PHA
          ORA #$02
          AND #$FE
          STA $01
          JSR SAVE (rutinen herunder)
          PLA
          STA $01
          RTS
```

Det er ikke tilladt at SAVE til andre devices end kassettebåndoptager og disk-station. Sekundær adressen skal være 0 til disk, men kan vælges til enten 0 eller 2 for kassettsens vedkommende. Vælges en sekundær adresse på 2, sendes automatisk en EOF - *end-of-file* marker efter programmet. Sekundær adressen kan også vælges til henholdsvis 1 og 3 - sidstnævnte med EOF - for SAVE af maskinkode til diskette. Det logiske filnummer er uden praktisk betydning! Rutinen er helt identisk med BASIC kommandoen SAVE.

```
LDA #logisk filnummer
LDX #device nummer
LDY #sekundær adresse
JSR $FFBA
```

Denne rutine er ikke nødvendig ved SAVE uden programnavn til kassette - device nr. 1.

```
LDA #antal karakterer i filnavnet
LDY #MSB startadresse navn
LDX #LSB startadresse navn
JSR $FFBD
```

I det følgende fastlægges programmets startadresse. Denne skal placeres i page 0 (adresse 0 til 255) i det sædvanlige 6502/6510 adresseformat (LSB/MSB). Adressen på det register, som LSB-delen af programmets startadresse opbevares i, videregives i A registret:

```
LDA #LSB af programmets startadresse
STA page 0 hukommelse-1
LDA #MSB af programmets startadresse
STA page 0 hukommelse
LDA #adressen på LSB-delen (hukommelse-1)
LDX #LSB af sidste byte +1
LDY #MSB af sidste byte +1
JSR $FFD8
```

Er der tale om et BASIC program, der skal SAVE's, vil sidste del af SAVE rutinen kunne forenkles en del:

```
LDA #$2B - BASIC-start lagres i $2B/$2C
LDX $2D - end-of-BASIC lagres i $2D/$2E
LDY $2E
JSR $FFD8
```

### 4.4 Computerens hukommelse

Computerens hukommelse er opdelt i en række special-områder med hver sit speciale. Grænserne mellem de fleste af disse områder, kan påvirkes af det anvendte program, eller gennem målrettet indgriben fra brugers side. Hovedområderne er:

#### SYSTEMOMRÅDE 0000-03FF

En udnyttelse af hukommelsespladser i dette område bør overvejes meget nøje, da selv den mindste forandring af system-variable m.m. kan få computeren til at "gå i bro". Der findes dog et par enkelte "soft-spots", som brugeren kan udnytte uden problemer. Disse er:

0002	1 byte
00FB-00FE	4 bytes
02A7-02FF	89 bytes (sprite 11 kan placeres her)
0313	1 byte
0334-033B	8 bytes
033C-03FB	192 bytes (kan udnyttes, når kassette ikke bruges)
03FC-03FF	4 bytes

Selvom enkelte af de øvrige adresser kan udnyttes afhængigt af computerens aktuelle virkemåde, er det yderst fornuftigt, at erklære alle ikke-nævnte adresser for OFF-LIMIT.

#### SKÆRMOMRÅDE 0400-07FF

Området fra 0400-07E7 (1000 bytes) anvendes som skærmhukommelse. Er skærmen placeret et andet sted i computerens hukommelse, kan området anvendes til maskinkodeprogrammer, eller BASIC - se BASICOMRÅDE. Området fra 07E8 til 07F7 (16 bytes) kan frit benyttes. Adresserne 07F8-07FF fungerer som SPRITE-pointere, for aktiverede SPRITES. Sprite-pointere og skærm følges altid ad!

#### BASICOMRÅDE 0800-9FFF

Kan være begrænset til maksimalt 7FFF, i forening med et indstiksmodul. Området kan opdeles i følgende områder, hvis grænser er fleksible. Adresserne for disse grænser opbevares i systemområdet. Området kan inddeles i disse hovedområder:



## 002B/002C - Start of BASIC

Starten på BASIC-området. Værdien kan flyttes op eller ned i hukommelsen - efter behov. F.eks. for at skabe plads til maskinkode-programmer under BASIC-området. En evt. ændring af startadressen skal foretages inden, der læses BASIC programmer ind i computeren. Ændringen kan foretages direkte fra tastaturet:

POKE 45, *LSB ny adresse*  
POKE 46, *MSB ny adresse*  
NEW

NEW kommandoen skal altid benyttes, inden der læses programmer ind i hukommelsen.

## 002D/002E - Start of variables

Starten på variabelområdet - sidste BASIC programbyte +1! De tre foregående bytes skal altid være nul! Variabler, integer variabler lagres her.

## 002F/0030 - Start of arrays

Starten på array-hukommelsen for integer arrays, almindelige arrays.

## 0031/0032 - End of Arrays

Slut på array-hukommelsen. Sidste byte +1. Området vokser opefter i hukommelsen.

## 0033/0034 - Start of Strings

Starten på strengområdet. Området svinger i størrelsen afhængigt af, hvor hyppigt strengvariablerne benyttes. En *garbage collection* fremtvinger en "oprydning" i strengområdet, når pointeren nærmer sig den foregående. Befinder der sig "affald" i strengområdet, flyttes startadressen op til den højest mulige adresse, uden at overskrive variablerne.

## 0037/0038 - Top of BASIC

Top-of-BASIC adresse. Sætter topgrænsen for det område, som kan udnyttes af BASIC. Pointeren kan uden problemer flyttes nedefter i hukommelsen, f.eks. for at reservere plads til maskinkode rutiner. Alle informationer, der lagres fra denne adresse og opefter i hukommelsen, påvirkes ikke af BASIC-programmet med tilhørende variabler. Pointeren flyttes med kommandoerne:

POKE 55, *LSB laveste maskinkode adresse*  
POKE 56, *MSB laveste maskinkode adresse*  
CLR

Husk CLR kommandoen efter justering af top-of BASIC.

### BASIC kommandoer, der påvirker inddelingen

Kommando	2B/2C	2D/2E	2F/30	31/32	33/34	37/38
CLR	nej	nej	2D/2E	2D/2E	37/38	nej
RUN	nej	nej	2D/2E	2D/2E	37/38	nej
NEW	nej	2B/2C	2B/2C	2B/2C	37/38	nej

### BASIC ROM A000-BFFF

Det er muligt, at anvende det underliggende RAM-område til maskinkode-rutiner, som kun kræver adgang til KERNAL ROM. Alfabeter og højopløsningsskærm kan uden problemer placeres her.

### FRIT OMRÅDE C000-CFFF

Området er ideelt til opbevaring af maskinkode-rutiner. Commodore DOS 5.1 til 1541 disk-stationen optager området fra C000 (52224) og opefter.

### IN/OUTPUT OMRÅDE D000-DFFF

Området er optaget i "tre lag". Øverste lag er alle kontrol-kredsene, næstøverste lag er karakter-ROM og nederste lag er RAM-område. RAM-området, kan kun udnyttes, når både BASIC og KERNAL ROM er udkoblet - se BACKUP programmet senere i bogen.

### KERNAL ROM

Indeholder computerens styresystem. ROM'en må kun udkobles, når alle interrupts er afbrudt! Udkobling af KERNAL ROM de-aktiverer BASIC ROM'en. Det er muligt, at anvende det underliggende RAM-område til maskinkode-rutiner, som hverken kræver adgang til BASIC eller KERNAL ROM. Alfabeter og højopløsningsskærm kan uden problemer placeres her.

#### 4.4.1 APPEND af BASIC programmer

APPEND betyder at samle eller tillægge. Indenfor data anvendes betegnelsen om selve den proces, at tilføje data til slutningen af eksisterende data - f.eks. ved at indlæse et nyt program i *fortsættelse* af et bestående program. Efter APPEND udgør begge programmer *et* hele - et nyt program.

Commodore 64 råder ikke over en sådan kommando, men den er uhyre let at simulere - endda fra BASIC. De eneste krav, teknikken stiller, er, at begge programmer skal kunne rummes i BASIC hukommelsen på samme tid, og at det *højeste* linjenummer i det først indlæste program er mindre end det *mindste* linjenummer i det program, der APPEND'es! Processen kan gentages uden problemer, bare disse to betingelser er opfyldt, og processen virker lige godt med kassette og disk-station!

Sådan gør du:

- Indlæs første program i hukommelsen.
- Indtast følgende rutiner - uden linjenumre - direkte fra tastaturet:

```
PRINT PEEK(43),PEEK(44) - husk disse to værdier
T=PEEK(45)+256*PEEK(46)-2
POKE 46,T/256
POKE 45,T-256*PEEK(46)
LOAD "NYT PROGRAM",device
POKE 43,gammel PEEK(43)
POKE 44,gammel PEEK(44)
CLR
```

Nu er de to programmer samlet til eet nyt program.

#### 4.4.2 Placering af maskinkode programmer

Maskinkode programmer kan placeres overalt i hukommelsen, f.eks. i kassettebufferen (fra adresse 828) eller lige under det område, der normalt anvendes til *sprite 11*, fra adresse 679, hvor rutinen ikke vil kollideres med noget! Det eneste register, der påvirkes af rutinen, er A-registeret. Rutinen er assemblen på adresse 0000, for at man lettere kan finde afstanden til de to bytes, der skal erstattes med *programadressen* på det maskinkode program, som rutinen skal kalde. Bemærk, at programmet under ingen omstændigheder må indlæses fra adresse 0000!!!

I den viste udgave kan maskinkoderutiner være placeret overalt i hukommelsen - også i adresseområdet \$D000 til \$DFFF!!! Disse rutiner må ikke indeholde kald til BASIC eller KERNAL ROM, da begge disse ROM'er er udkoblede! Anvendes værdien \$FE i stedet for \$FC (mærket med \*), kan programmet lagres under BASIC ROM'en, og alle rutiner i KERNAL ROM'en (i området \$E45F til \$FFFF) kan udnyttes. Ønskes udkobling af både KERNAL og BASIC, men ikke adgang til området D000 til DFFF erstattes FC med værdien FD. I dette tilfælde er det stadig muligt, at benytte alle *input/output-kredse*.

170:	0000	;	*= 0000
190:	0000 08		PHP
200:	0001 78		SEI
210:	0002 A5 01		LDA \$01
220:	0004 48		PHA
(225:	+2 09 ??		ORA #byte)
230:	0005 29 FC*		AND #\$FC*
240:	0007 85 01		STA \$01
250:	0009 20 ?? ??		JSR <i>programadressen</i>
260:	000C 68		PLA
270:	000D 85 01		STA \$01
280:	000F 28		PLP
290:	0010 60		RTS

Linje 225 kan undlades, hvis både BASIC og KERNAL ROM skal udkobles. Hvis

KERNAL udkobles (BASIC udkobles også, men område D000 til DFFF er ikke tilgængeligt) skal *byte* erstattes med værdien 1, og hvis alene BASIC skal udkobles, skal *byte* have værdien 2.

### 4.4.3 Sådan lagres BASIC programmer i hukommelsen

Når en tekst-linje indtastes, sker det til en speciel buffer, placeret fra adresse 0200 til 0258, som kan rumme ialt 89 bytes incl. en afslutningskarakter. I praksis indlæses aldrig mere end 80 karakterer - det maksimale antal karakterer, der kan eksistere i en *logisk* skærmlinje. En *logisk* skærmlinje kan maksimalt bestå af to *fysiske* skærmlinjer. En *fysisk* skærmlinje består af 40 karakterer.

Den samme buffer benyttes i forbindelse med INPUT og INPUT# kommandoen, og dette er den egentlige årsag til, at man maksimalt kan læse 88 karakterer fra disk-station eller kassettebåndoptager med INPUT# kommandoen.

Sker indtastningen af tekstlinjen *udenfor* programmet taler vi om *direkte indtastning* eller *direct mode* i modsætning til *program mode*. Computeren skelner skarpt mellem disse to tilstande - af hensyn til bl.a. programmørens sjælefred! Det kræver nok en lille forklaring.

Når computeren befinder sig i *direct mode*, er den i stand til at afgøre, om der er tale om indtastning af en programlinje eller en *direkte kommando* - f.eks. RUN. Denne sortering foretages ene og alene ud fra det første tegn, den finder i *input bufferen*, efter at brugeren har signaleret, at indtastningen er afsluttet (RETURN). Er det første tegn et ciffer, konkluderer computeren, at der er tale om en programlinje, og er det første tegn et bogstav, er der tale om en *direkte kommando*.

Når computeren har fundet ud af, at det er en programlinje vi har indtastet, går den i gang med at omdanne linjen til et internt format. Når dette er sket, søger computeren i *programhukommelsen* for at finde en programlinje med samme eller højere linjenummer. Kan det ikke lade sig gøre, ved computeren, at linjen skal placeres sidst i *programhukommelsen*. Finder computeren en linje med samme linjenummer, slettes denne, hvorefter computeren indsætter den nye linje. Finder computeren en linje med et højere linjenummer, men ikke en med samme linjenummer, skabes der plads til den nye programlinje, hvorefter denne skrives til *hukommelsen*. Forestil dig nu, hvad der kunne ske, hvis computeren ikke var i stand til at skelne mellem *program* og *direct mode*. I alle de tilfælde, hvor brugeren indtastede et ciffer, som svar på en GET eller INPUT kommando, ville den tro, at der var tale om en ny programlinje, og der skal ikke megen fantasi til at forestille sig, hvilke problemer det kunne medføre. Denne skelnen mellem *direct* og *program mode* er den direkte årsag til, at GET og INPUT ikke kun kan accepteres i programmer!

Som allerede antydnet, omdannes alle programlinjer til et internt format, inden de lagres. Dette format er opbygget således:

# BYTE FUNKTION

1	LSB af pointer til begyndelsen af næste linje
2	MSB af pointer til begyndelsen af næste linje
3	LSB af linjenummer
4	MSB af linjenummer
5-255	maksimalt område for programtekst
256	0 - værdien indikerer afslutning på linjen.

Hver programlinje starter således med adressen på første byte i næste programlinje, og hver programlinje afsluttes med ASCII-værdien 0 (nul). Selve programlinje-teksten kan *aldrig* indeholde værdien nul, så det er en meget enkel måde, at konstatere afslutningen på en linje. Afslutningen af programmet indikeres på samme enkle måde. Af praktiske årsager, peger pointeren i sidste programlinje stadig mod det første, efterfølgende byte. Denne ikke-eksisterende programlinje, kan naturligvis ikke indeledes med adressen på første byte i næste linje, og derfor sætter computeren disse to adressebytes til nul. Møder den dem, så ved computeren, at der ikke findes flere linjer i programmet, og den stopper. På denne simple måde undgår computeren, at opfatte *variabelhukommelsen* som programlinjer. Dette er også årsagen til, at ethvert program *skal* afsluttes med 3 nuller - et nul, som markering for linjeslut i sidste programlinje, og to nuller som slutmarkering for programmet. Mangler disse tre nuller, vil der opstå problemer, når der indsættes nye programlinjer, idet computeren fortsætter ind i variabelhukommelsen, og prøver at få mening ud af indholdet. Resultatet kan under visse omstændigheder blive, at computeren ikke møder de to nuller fra den ikke-eksisterende linje, før den kommer op i input/output området, hvor enhver forandring kan få katastrofale følger. Så pas på, når du POKE'r i programområdet.

Selve tekstområdet i en programlinje kan som allerede vist bestå af op til 251 karakterer. Det er kun den *logiske skærmlinje*, der begrænser den effektive linjelængde til 80 karakterer. Ydermere er det ikke alt, der opbevares i den form, det indtastes.

Vi har allerede set, at et linjenummer omdannes til 2 bytes i hukommelsen. Dvs. at linjenummerets størrelse *ikke* har nogen betydning for pladskravene - men.... I kommandoer, som GOTO, GOSUB, RUN og THEN lagres linjenummeret eksakt, som det indtastes - derfor er det en god ide, at benytte små linjenumre. For det første sparer det plads, men allervigtigst er det dog, at computeren hurtigere kan læse linjenumre med få cifre!!

En anden vigtig forandring kaldes *tokenizing*. Alle BASIC kommandoer og funktioner opbevares som et byte i hukommelsen. Derfor spares absolut ingen plads ved at indtaste ? i stedet for PRINT. Efter konverteringen fylder begge dele lige meget - nemlig et byte! At det så kan være meget fornuftigt at indtaste ? i stedet for PRINT, er en helt anden sag. Det har jo noget at gøre med den begrænsning, der ligger i de 80 karakterer, som kan rummes på en *logisk skærmlinje*!

Er tokenizing af BASIC kommandoer og funktioner en fordel, set ud fra et ønske om en effektiv udnyttelse af hukommelsen, så medfører det dog også en række ulemper. Bl.a. er det årsagen til, at PRINT#, INPUT# og GET# kommandoerne *skal* indtastes uden mellemrum mellem f.eks. GET og #. Sker det

ikke, tror computeren, at der er tale om kommandoen GET efterfulgt af et mellemrum og tegnet # - og så ved vi jo alle, hvad computeren skriver på skærmen, når den møder programlinjen! Tokenizing er også årsagen til, at vi ikke må anvende variabelnavne, som på en eller anden måde rummer kommandonavne - f.eks. Total, KONTTO, sPRINT, fORan, cIF, bANDit osv. Computeren er ikke i stand til at afgøre, om der er tale om et variabelnavn eller ej! Den genkender blot en tegnfølge, der svarer til en kommando, og erstatter straks tegnfølgen med det tilsvarende token.

De tokens, som computeren anvender, har alle en kode, der ligger over 127 i ASCII-værdi. Det er det, der overhovedet sætter computeren i stand til at afgøre, om der er tale om variabelnavne eller tokens. Det eneste situation, hvor ASCII-værdier over 127 ikke har nogen effekt, er, når de sættes i anførselstegn. Det kan du let kontrollere ved at fremstille en REM sætning og en DATA sætning, som rummer de grafik-karakterer, der indtastes ved hjælp af Commodore-knappen! Lister du et sådant program, optræder der pludselig kommando-navne i stedet! Det kan naturligvis være en fornuftig ting, hvis man har svært ved at huske tast-kombinationerne, for de forkortede indtastninger. Man placerer blot to REM-linjer i starten af programmet. Den første indeholder normale tegn - adskilt af mellemrum - og den anden de tilsvarende "Commodore-tegn" - ligeledes adskilt af mellemrum. Kommer man i tvivl, behøver man blot at indtaste LIST-2, hvis man vil have listen op på skærmen. I forening med DATA-sætninger, kan dette fænomen dog volde problemer. Det vil ganske enkelt være umuligt at LIST'e de specielle grafik karakterer, medmindre de er sat i anførselstegn. Til gengæld risikerer man ikke, at tekster som PRINT, TOTAL osv. bliver "oversat" til tokens. Det kan let kontrolleres med:

```
100 DATA PRINT
110 READ A$
120 IF A$="PRINT" THEN PRINT "OK"
130 INPUT "SKRIV PRINT";B$
140 IF A$=B$ THEN PRINT "OK"
```

### 4.4.4 BASIC-program laver BASIC-program

Når man ved, hvordan computeren lagrer sine programmer internt, er det faktisk muligt, at fremstille BASIC-programmer med et BASIC-program. Normalt vil man ikke have det store behov - computeren er jo i ganske ferm til den opgave, så hvorfor lave alt det besvær?

Jo, i ganske specielle situationer, kan det faktisk være til nytte. Lad os antage, at du kun har en kassettebåndoptager, og en af dine venner har en diskstation med mange interessant maskinkoderutiner. Råder I ikke over en monitor, findes *ingen* let måde - bortset fra de tidligere nævnte maskinkode løsninger - til at overføre programmerne fra disk til kassette. Er maskinkoden beregnet til at blive placeret under KERNAL ROM'en vil heller ikke de tidligere viste maskinkode rutiner være til nogen hjælp. Det er til gengæld det følgende BASIC program.

Programmet læser en maskinkode programfil fra disk, og skriver den tilbage til disk igen, men i form af et BASIC program, hvor hvert byte optræder som decimalværdier i DATA sætninger. Når programmet er færdigt med sit arbejde,

eksisterer der to filer på disketten. Den ene med navnet "*originalnavn*" og den anden med navnet "*DAT.originalnavn*". Den sidste fil kan LOAD'es ind i computeren, og SAVE's til kassettebåndoptager som et almindeligt BASIC-program. Du skal blot huske at tilføje linjerne:

```
1 A=startadresse
2 READ B:IF A=999 THEN END
3 POKE A,B:A=A+1:GOTO 2
```

Startadressen optræder som første linjenummer i det konverterede program, så den bli'r ikke svær at finde. Program konverteren ser således ud:

```
110 REM MAKE LOADER
120 REM
130 ADD=2049:REM BASIC-START
140 PRINT CHR$(147)
150 INPUT "NAVN PAA FIL, DER SKAL KONVERTERES";FK$
160 OPEN 8,8,8,FK$+"",P,R"
170 NK$=LEFT$("DAT."+FK$,16)
```

Den nye datafil, der fremstilles i programmet, får navnet "*DAT.*" hængt foran det oprindelige navn.

```
180 OPEN 7,8,7,NK$+"",P,W"
190 HEX=ADD:GOSUB 430
200 GET#8,B$,C$
210 LINE=ASC(B$+CHR$(0))+256*ASC(C$+CHR$(0))
```

Første linje i det nye program får et linjenummer, der svarer til startadressen i maskinkoderutinen.

```
220 N=0
230 IF (N AND 7)<>0 THEN 280
240 IF N<>0 THEN PRINT#7,CHR$(0);:REM SLUT PAA LINJEN
250 HEX=ADD:GOSUB 430
260 HEX=LINE+N:GOSUB 430
270 PRINT#7,CHR$(131);:REM DATA KOMMANDO
280 GET#8,A$:SS=ST
290 A=ASC(A$+CHR$(0)):REM KONVERTER BYTE TIL DECIMALVAERDI
300 B=INT(A/100)
310 C=INT((A-B*100)/10)
320 D=A-B*100-C*10
330 PRINT#7,CHR$(B+48);CHR$(C+48);CHR$(D+48);:REM SKRIV TRE CIFRE
340 IF (N AND 7)<7 THEN PRINT#7,"";:REM SAET KOMMA IMELLEM
350 IF SS=0 THEN N=N+1:GOTO 230
```

Nu er den egentlige program-generator-rutine afsluttet. Tilbage står blot, at afslutte programmet med 999 fulgt af tre gange 0, der indikerer slutningen på et BASIC-program.

## Programmering i maskinkode

```
355 IF (N AND 7)=7 THEN PRINT#7,",";
360 PRINT#7,"999";
370 FOR N=1 TO 3
380 PRINT#7,CHR$(0);
390 NEXT N
400 CLOSE 7
410 CLOSE 8
420 END
```

Maskinkoden er nu konverteret til et BASIC-program, der udelukkende indeholder DATA-linjer. Subrutinen herunder konverterer decimaltal til hex-tal.

```
430 HI=INT(HEX/256)
440 LO=HEX-256*HI
450 PRINT#7,CHR$(LO);CHR$(HI);
460 RETURN
```

Konvertereren fremstiller et rent BASIC-program, som består af kommandoen DATA efterfulgt af maksimalt 8 værdier. Alle værdier består af tre cifre - nul skrives f.eks. som "000". Otte maskinkode bytes fylder ialt 37 bytes i BASIC, så den maksimale størrelse for et maskinkodeprogram, der skal konverteres ligger omkring 8400 bytes - men det er jo også et ganske respektabelt maskinkodeprogram.

### 4.4.5 Relokerbar maskinkode-loader

Hyppigt har man brug for at anvende flere forskellige maskinkode rutiner på samme tid. Problemet er blot, at de alle plejer at starte på adressen 49152 (hex C000), og det gør det jo lidt problematisk.

En stor del af de programmer, man har, volder det ingen problemer at relokere. Der stilles kun det ene krav, at programmerne ikke indeholder maskinkode ordrer som:

```
JSR adresse
JMP adresse
JMP (adresse)
```

som "peger ind i" selve maskinkoderutinen. Vær også opmærksom på pointere til adresseområder beliggende i rutinens arbejdsområde. Disse pointere skal normalt modificeres, og kun et nøje studie af maskinkoden med tilhørende dataområde kan fortælle hvordan. Rummer programmet ikke disse ordrer, kan programmet placeres frit i hukommelsen.

Indeholder programmet JSR og JMP ordrer, findes stadig en mulighed, men i det tilfælde er maskinkode ikke særligt praktisk. Programmet skulle jo gerne være universelt. Et program, der modificerer JSR og JMP adresser, virker kun pålideligt, hvis der ikke optræder "data" i den indlæste kode. Indirekte JMP ordrer volder problemer, idet maskinkoden skal relokeres, så de to adressebytes henviser til en lige adresse. Derfor foreslås følgende løsning (anvend eventuelt DATA-konverterings programmet vist tidligere):



Fremstil en BASIC udgave af maskinkode programmet, hvor koderne befinder sig i DATA sætninger. Omform alle adresse informationer til de rigtige adresser, men med modsat fortegn - f.eks. som her, hvor maskinkoden:

```
3011: 20 14 32 JSR $3214
3014: 40      RTS
```

ser således ud i BASIC formen:

```
510 DATA 32,20,50,96
```

indeholder adressen  $50 \times 256 + 20 (=12820)$ . Efter konverteringen vil sætningen se således ud:

```
510 DATA 32,-12820,96
```

Næste trin er at tilføje den oprindelige startadresse (her 12817 eller hex 3011) foran maskinkode-rutinen, og værdien 999 placeres efter, som her:

```
500 DATA 12817
510 DATA 32,-12820,96
520 DATA 999
```

Disse værdier udnyttes af relokatoren. Normalt indeholder DATA-sætninger med maskinkode kun positive værdier, så når vi møder en negativ værdi, ved vi, at der er tale om en værdi, der skal bearbejdes. Møder programmet værdien 999, stopper udførelsen med en udskrift af første ledige byte efter maskinkode rutinen, vi lige har læst ind. Det følgende BASIC-program er i stand til at relokere programmer med dette format overalt i computerens hukommelse:

```
100 PRINT CHR$(147)
110 INPUT "INDTAST RELOKERINGSADRESSE";RL
120 IF RL<0 OR RL>65535 THEN 100
130 READ OL:REM GAMMEL START ADRESSE
140 DIFF=RL-OL
150 READ BYTE:IF BYTE=999 THEN PRINT "FRIT FRA:";RL:END
160 IF BYTE>=0 THEN 210
170 AD=BYTE+DIFF
180 POKE RL,AD-256*INT(AD/256)
190 RL=RL+1
200 BYTE=INT(AD/256)
210 POKE RL,BYTE
220 RL=RL+1
230 GOTO 150
```

Programmet stopper automatisk, når det møder værdien 999, og den næste ledige adresse udskrives på skærmen. Næste program kan lagres herfra, om så ønskes. Lad os se på et eksempel:

Maskinkode programmet læser værdierne for en joystick. Startadressen er her 828, og programmet kaldes medUSR-funktionen med nummeret på det ønskede joystick, som parameter - f.eks. således:

## Programmering i maskinkode

```

100 FOR N=1 TO 2
110 POKE 785,828 AND 255:POKE 786,828/256
120 Y=USR(N):PRINT "JOYSTICK";N,
130 IF Y AND 16 THEN PRINT "FIRE ";
140 IF Y AND 8 THEN PRINT "RIGHT ";
150 IF Y AND 4 THEN PRINT "LEFT ";
160 IF Y AND 2 THEN PRINT "DOWN ";
170 IF Y AND 1 THEN PRINT "UP ";
180 PRINT
190 NEXT N
200 GOTO 100

```

Maskinkodeprogrammet ser således ud:

```

155: 033C                      *= 828
                                ;
190: 033C 20 F7 B7 USERIND JSR $B7F7
200: 033F 98                  TYA
210: 0340 08                  PHP
220: 0341 78                  SEI
230: 0342 29 01              AND #1
240: 0344 F0 09              BEQ JOYSTIK2
                                ;
290: 0346 20 6A 03 JOYSTIK1 JSR KEYBOFF
300: 0349 AD 00 DC            LDA 56320
310: 034C 4C 55 03            JMP EXIT
                                ;
340: 034F 20 6A 03 JOYSTIK2 JSR KEYBOFF
350: 0352 AD 01 DC            LDA 56321
360: 0355 29 1F              AND #31
370: 0357 A8                  TAY
380: 0358 20 A2 B3            JSR $B3A2
390: 035B A9 FF              LDA #255
400: 035D 8D 02 DC            STA 56322
410: 0360 AD 0E DC            LDA 56334
420: 0363 09 01              ORA #1
430: 0365 8D 0E DC            STA 56334
435: 0368 28                  PLP
440: 0369 60                  RTS
                                ;
460: 036A AD 0E DC KEYBOFF LDA 56334
470: 036D 29 FE              AND #254
480: 036F 8D 0E DC            STA 56334
490: 0372 A9 00              LDA #0
500: 0374 8D 02 DC            STA 56322
505: 0377 8D 03 DC            STA 56323
506: 037A 60                  RTS

```

Der findes tre kald til adresser i selve programmet - nemlig i linjerne 290, 310 og 340. Linjerne 290 og 340 kalder subrutinen på adresse 036A ( $106+256*3 = 874$ ) og linje 310 springer til adresse 0355 ( $85+256*3 = 853$ ) - adresseværdierne, der skal ændres, er mærket med en stjerne i DATA-listen herunder:

BASIC-programmet, der er fremstillet med vor maskinkode til DATA konverter, har dette udseende:

```
828 DATA 032,247,183,152,008,120,041,001
836 DATA 240,009,032,106*,003*,173,000,220
844 DATA 076,085*,003*,032,106*,003*,173,001
852 DATA 220,041,031,168,032,162,179,169
860 DATA 255,141,002,220,173,014,220,009
868 DATA 001,141,014,220,040,096,173,014
876 DATA 220,041,254,141,014,220,169,000
884 DATA 141,002,220,141,003,220,096,999
```

Efter konverteringen, ser det færdige program således ud - inklusive relokatoren:

```
100 PRINT CHR$(147)
110 INPUT "INDTAST RELOKERINGSADRESSE";RL
120 IF RL<0 OR RL>65535 THEN 100
130 READ OL:REM GAMMEL START ADRESSE
140 DIFF=OL+RL
150 READ BYTE:IF BYTE=999 THEN PRINT "FRIT FRA:";RL:END
160 IF BYTE>=0 THEN 210
170 AD=BYTE+DIFF
180 POKE RL,AD-256*INT(AD/256)
190 RL=RL+1
200 BYTE=INT(AD/256)
210 POKE RL,BYTE
220 RL=RL+1
230 GOTO 150
240 DATA 828
250 REM MASKINKODEDATA
828 DATA 032,247,183,152,008,120,041,001
836 DATA 240,009,032,-874*,173,000,220
844 DATA 076,-853*,032,-874*,173,001
852 DATA 220,041,031,168,032,162,179,169
860 DATA 255,141,002,220,173,014,220,009
868 DATA 001,141,014,220,040,096,173,014
876 DATA 220,041,254,141,014,220,169,000
884 DATA 141,002,220,141,003,220,096,999
```

Vær opmærksom på, at programmer, som selv beregner og modificerer adresser, normalt ikke kan relokeres uden videre. Det samme gælder programmer, som opbevarer adresserne i datatabeller, der læses som enkeltbyteinformationer! Den følgende rutine er et eksempel på dette:

```
LDA MSBADRESSE,Y
PHA
LDA LSBADRESSE,Y
PHA
RTS
```

I dette - og lignende tilfælde - findes der ingen enkel metode til relokering af koden.

## Kapitel 5

### Commodore PLUS/4 og maskinkode

De nye Commodore computere Plus/4 og C16 er begge udstyret med en *maskinkode monitor* - et program, som er specielt velegnet til arbejde med maskinkode. Monitoren gør det muligt at assemblere, disassemblere, ændre og undersøge maskinkode programmer, og er som sådan et virkeligt godt arbejdsredskab, hvis man har lyst at prøve kræfter på det felt.

Monitor'en, der kaldes fra BASIC med kommandoen *MONITOR* , rummer de følgende kommandoer, alle bestående af 1 bogstav:

A - Assemble	Assembler en linje til 6502 maskinkode
C - Compare	Sammenligner to dele af hukommelsen og melder om evt. afvigelser
D - Disassemble	Disassembler 6502 maskinkode
F - Fill	Fylder et specificeret område af hukommelsen med en byteværdi
G - Go	Starter udførelsen af et maskinkodeprogram - svarer til BASIC-kommandoen SYS
H - Hunt	Søger hukommelsen igennem efter specificeret byte
L - Load	LOAD'er en fil fra kassette eller diskette
M - Memory	Udskriver indholdet af hukommelsen
R - Registers	Viser registrenes indhold i 6502 processoren
S - Save	SAVE'r en fil til kassette eller diskette
T - Transfer	Flytter maskinkode fra et område til et andet
V - Verify	VERIFY'er et program på kassette eller diskette
X - Exit	Hop tilbage til BASIC

#### 5.1 Beskrivelse af de enkelte monitor kommandoer

##### A - ASSEMBLE

**FORMAL:** At indtaste en linje i assemblerkode, som computeren konverterer til maskinkode.

**ANVENDELSE:** A *adresse mnemonic operand*

*Adresse* er en 4-cifret hexadecimal værdi i området 0000 til FFFF, hvori koden skal placeres. *Mnemonic* er en standard mnemonic for 6502 processoren, f.eks. LDA, CLC, STY, BIT etc., og *operand* er, afhængigt af den benyttede

instruktion, data (1 byte) eller en adresse (1 eller 2 bytes), som noteres på en form, der svarer til den ønskede adresseringsmåde. \$-tegnet indikerer, at der er tale om hex-format, og et foranstillet #-tegn indikerer, at der er tale om data.

Når der trykkes på RETURN knappen oversættes det indtastede til maskinkode. Er indtastningen forkert, skrives et spørgsmålstegn på næste linje. Er indtastningen i orden, indledes næste linje med adressen på den næste "frie" plads i hukommelsen.

EKSEMPEL *kursiv*=indtastninger:

```
.A 4E0F LDA ($2F),Y
.A 4E11 PHA
.A 4E12
```

## C - COMPARE

FORMAL: Sammenligning af to områder af hukommelsen.

ANVENDELSE: C *adresse1* *adresse2* *adresse3*

*Adresse1* er startadressen, og *adresse2* slutadressen på det område af hukommelsen, der skal sammenlignes med et andet område, der starter fra *adresse3*.

Er de to områder identiske, vil der ikke ske andet, end at cursoren rykker en linje ned. Optræder der afvigelser, udskrives adresserne på de bytes, der ikke er i overensstemmelse med indholdet i det andet hukommelsesområde.

## D - DISASSEMBLE

FORMAL: At disassemblere maskinkode til mnemonics.

ANVENDELSE: D *startadresse* (*slutadresse*)

Det er ikke nødvendigt, at specificere en *slutadresse* for disassemblering. Indtastes kommandoen D, uden efterfølgende data, vises næste skærmside! Opdages fejl i koden, kan cursoren flyttes til pågældende linje, som kan rettes umiddelbart. Bemærk, at ikke-gyldige koder udskrives, som "???". Dette indikerer normalt, at der er tale om et dataområde, af en eller anden art!

EKSEMPEL *kursiv*=indtastninger:

```
D 1000 1FFA
. 1000 LDA #$0A
. 1002 ???
. 1003 BIT $1020
```

## F - Fill

FORMAL: At fylde et område af hukommelsen med en byte-værdi.

ANVENDELSE: F *startadresse* *slutadresse* *byte*

Området fra *startadresse* til og med *slutadresse* vil blive udfyldt med værdien

## Commodore PLUS/4 og maskinkode

*byte*. *Byte* skal være en to-cifret hex-værdi. Kommandoen anvendes især til initialisering af større RAM-områder, f.eks. til at slette skærmen m.m.

EKSEMPEL *kursiv*=indtastninger:

.F 0800 08FF 00 Fylder området fra hex 0800 til hex 08FF (256 bytes) med hex-værdien 00.

### G - GO

FORMAL: At starte kørslen af et maskinkode program.

ANVENDELSE: G *startadresse*

Udelades startadressen, begynder kørslen fra *nuværende programtæller* - PC. Værdien kan findes ved hjælp af R kommandoen. G kommandoen vil initialisere alle registrerne til de værdier, der vises i R - kommandoen. Lav altid en kopi af et nyt program, inden det udføres - det sparer en for de værste følger af *optimisme* i omgang med maskinkode!

### H - HUNT

FORMAL: Søger efter forekomsten af bestemte byte- eller karakterfølger i hukommelsen.

ANVENDELSE: H *startadresse slutadresse data...*

Kommandoen søger efter den specificerede datakombination i området fra *startadresse* til og med *slutadresse*. *Data* kan være hexadecimal værdier, eller karakterstreng. En karakterstreng *skal* indledes med en apostrof. Computeren udskriver alle de adresser, hvor den skitserede datafølge forekommer.

EKSEMPEL *kursiv*=indtastninger:

.H 1000 2000 'DATA  
1280 14FA 19AD  
.H 1000 2000 0F 03 07  
1A0B 1AC0 1FE0

### L - LOAD

FORMAL: At indlæse en fil i computerens hukommelse.

ANVENDELSE: L "*filnavn*",*device*

Kommandoen følger de samme regler som BASIC LOAD kommandoen, blot skal *device* specificeres som en 2-cifret hexadecimal værdi - f.eks. 01 for kassette, og 08 for disk-station. Filen læses til den startadresse, som angives i programheaderen. Dvs. at filerne *altid* placeres på den adresse, de blev SAVE't fra! Filen læses, indtil EOF mærket mødes.

### M - MEMORY DISPLAY

FORMAL: Viser indholdet i hukommelsen i både hex og ASCII-form.

ANVENDELSE: M *startadresse slutadresse*

Begge adressespecifikationer kan udelades. I det tilfælde udskrives en *skærmside* med informationer fra den sidst specificerede adresse. Indtastes kun startadressen, udskrives en *skærmside* herfra. ASCII-værdierne udskrives "NEGATIVT" (REVERSE). Formatet på udskriften er:

```
>1000 41 31 32 39 20 00 00 00 :A129 ...
  adresse      8 databytes      ASCII
```

## R - REGISTER DISPLAY

FORMAL: At vise indholdet af processorens registre.

ANVENDELSE: R

Kommandoen viser indholdet af processorens registre - bemærk at stack-pointeren (SP) vises i formen XX. Den reelle adresse i hukommelsen er 01XX

EKSEMPEL *kursiv*=indtastninger:

```
.R
  PC SR AC XR YR SP
1200 00 F2 A1 BC D0
```

Hvor PC er programtælleren, SR status register, AC A-registret, XR X-registret, YR Y-registret og SP er stack-pointeren.

## S - SAVE

FORMAL: At SAVE et program til diskette eller kassette.

ANVENDELSE: S "*filnavn*",*device*,*startadresse*,*slutadresse*

Kommandoen følger BASIC kommandoen SAVE, bortset fra at *device* skal specificeres som et 2-byte hex-ciffer - f.eks. 0A (*device* 10), og start- samt slutadressen på programmet *skal* opgives. Bemærk, at det byte, der står på slutadressen, *ikke* SAVE's!!! Filen, der fremstilles er en *programfil*!

EKSEMPEL *kursiv*=indtastninger:

```
.S "EN LILLE TEST",08,2A3B,2A9F
```

Kommandoen SAVE'r indholdet fra adresse 2A3B til og med adresse 2A9E (!) til *device* 8 som en *programfil* med navnet "EN LILLE TEST".

## T - TRANSFER

FORMAL: Overførsel af data fra en del af hukommelsen til en anden.

ANVENDELSE: T *startadresse* *slutadresse* *måladresse*

Programmet fra *startadresse* til og med *slutadresse* flyttes til området begyndende fra *måladresse*. *Måladresse* kan uden problemer være mindre end *startadresse*.

**EKSEMPEL** *kursiv*=indtastninger:

**.T** 1500 1600 1510

Flytter 257 bytes fra adresse 1500 til og med adresse 1600 til området startende fra adresse 1510. Som det ses, volder det heller ingen problemer, at områderne overlapper hinanden.

**V - VERIFY**

**FORMAL:** At VERIFY'e det SAVE'de program med indholdet i hukommelsen.

**ANVENDELSE:** V "filnavn",device

Kommandoen er identisk med den tilsvarende BASIC kommando, blot skal *device* opgives som en 2-cifret hex-værdi. Afviger filen fra hukommelsens indhold, gives en fejlmelding.

**X - EXIT**

**FORMAL:** At hoppe tilbage til BASIC interpreteren.

**ANVENDELSE:** X



## Kapitel 6

### Programmer på kassette

Selvom kassetten er et billigt lagermedie, er anvendelsen langt fra så fleksibel, som det kunne ønskes. For det første accepterer kassettebåndoptageren ikke alle data, og for det andet er den sløv, som bare ....

#### 6.1 Kassettebåndoptageren og data

Kassettebåndoptageren kan anvendes til lagring af data, men den kan på ingen måde konkurrere med en disk-station. For det første er kassettebåndoptageren kun i stand til at arbejde med to data-formater - nemlig programfiler og sekventielle filer. Fælles for dem begge er, at data kun kan læses og skrives sekventielt - dvs. i rækkefølge efter hinanden. Skal man have fat i et bestemt data-sæt, er det nødvendigt at læse filen, der indeholder de pågældende data, igennem, indtil de ønskede data er fundet. Ved læsning er det ikke så slemt. Problemet viser sig først for alvor, når man vil skrive data tilbage til båndet. Der eksisterer nemlig ingen metode til at skrive - lad os sige 30 bytes - tilbage i en datafil. Man er nødt til at læse hele filen ind først, og først herefter kan evt. ændringer foretages, og *hele* filen skrives tilbage.

En yderligere ulempe er, at kassettebåndoptageren ikke accepterer alle byte-koder. Ganske vist kan man udmærket *skrive* CHR\$(0) til kassette, men da dette byte signalerer *end of file* - slutningen af data-filen, er der ingen simpel måde at overføre f.eks. maskinkodeprogrammer eller data-elementer som indeholder denne karakter til kassette, hvis man ikke vil fremstille de nødvendige maskinkode rutiner (Se forrige kapitel side 64)) eller købe en monitor.

Denne utilstrækkelighed ved Commodores kassetteformat er den direkte årsag til, at maskinkode programmer normalt publiceres og lagres i form af DATA sætninger i et BASIC program. Metoden virker, men den er hverken hurtig eller særlig praktisk - mest fordi programstørrelsen dermed begrænses betydeligt i forhold til ren maskinkodelagring.

## 6.2 Kassettebåndoptagerens optageformat

Man kan sige mange negative ting om Commodores kassette-system, men en stor fordel har det dog - det er meget pålideligt. Hvis man renser tonehovederne med jævne mellemrum, og ellers kun anvender kassettebånd af rimelig kvalitet, er læsefejl meget sjældne. Årsagen skal søges i Commodores specielle optageformat.

Ulemperne skal også findes i det specielle optageformat. Hver gang et program SAVE's til kassette, fremstilles to kopier af programmet. Og det tager naturligvis to gange så lang tid som nødvendigt. Resultatet er, at kassettebåndoptageren kun er i stand til at læse eller skrive omkring 30 bytes pr. sekund - sammenlignet med disktestationens omkring 400 bytes, er det jo ikke noget at råbe hurra for.

Den anden optagelse af programmet er egentlig overflødig, men benyttes under LOAD til kontrol af programmet. Først indlæses programmet, hvorefter det sammenlignes med optagelse nr. to. Optræder der afvigelser mellem disse udgaver fås en fejlmelding, og programmet kan hverken listes eller køres. I hjemmecomputerens barndom kunne dette format have sin berettigelse, men i dag er det faktisk til mere skade end gavn. En evt. fejl kan jo lige så godt opstå i "kontrol optagelsen", som i den egentlige program optagelse, så Commodore har egentlig kun øget fejlmulighederne til det dobbelte.

Men nu er der jo ingen, der siger, at man skal af finde sig med Commodores måde at gøre tingene på. Selvom programmet umiddelbart virker tabt, så eksisterer der en ganske simpel måde, at kalde det frem på. Efter LOAD opbevares den adresse, der skal overføres til *Start of variables* pointeren i adresserne 174 og 175. Det eneste du skal gøre, er at indtaste følgende linje (uden linjenumre):

POKE 45,174:POKE 46,175:CLR

Hvorefter man i langt de fleste tilfælde vil have et perfekt program i hukommelsen. Men selv om der skulle være fejl i programmet, er ulykken normalt ikke særlig stor. Nu har man da i det mindste en chance for at rette på skaderne - og det er da ikke så lidt af en fordel.

Metoden virker for alle typer af programfiler, men kan normalt ikke anvendes på "købe" programmer, idet disse ofte foretager indlæsning i flere omgange. Ovenikøbet består første indlæsning oftest af et særligt maskinkodeprogram, som straks overtager kontrollen over systemet. Til gengæld findes der producenter, som udnytter lejligheden til at speed'e processen betydeligt op.

Er du interesseret i at vide, hvordan disse programmer arbejder, er det nødvendigt at vide, hvordan Commodore's kassettebåndoptager arbejder. Med det i hånden er det altid muligt, at læse den første blok fra kassetten - i den er producenterne *piskede* til at overholde Commodore's optageformat, ellers ville programmet *aldrig* kunne indlæses i Computeren. Der er dog ingen, der siger, at opgaven er let. Men det kan altså lade sig gøre.

Kassettefiler eksisterer i to udgaver, og optagelsen sker efter disse retningslinjer:

PROGRAMMER	DATAFILER
Programheader	Dataheader
Start/slutadresse+navn	Navn
Programheader (gentagelse)	Dataheader (gentagelse)
Program - een blok	Datablok
Program (getagelse)	Datablok (gentagelse)
Slutblok	Slutblok

Når optagelsen startes, overføres de nødvendige data til kassettebufferen, hvorefter de skrives som en samlet blok til kassetten. Fremgangsmåden er ens for både program- og datafiler. Den første blok, der skrives er *program-header'en* - en blok, der indeholder forskellige informationer om programmet af betydning for tilbagelæsningen. Header'en har følgende opbygning:

BYTE	BETYDNING
1	Headertype
2	LSB - startadresse
3	MSB - startadresse
4	LSB - slutadresse
5	MSB - slutadresse
5-21	Filnavn

Der findes fem forskellige headertyper:

- 
- 1 BASIC-programmer. LOAD'es fra *Start of BASIC*
  - 2 Datablok - byte 2 til 192 indeholder de egentlige data
  - 3 Maskinkode program - LOAD'es på den angivne startadresse
  - 4 Data-header - fortæller, at næste blok er en datablok
  - 5 Slutblok - fortæller, at filen er slut!
- 

Er der tale om en header af type 1 eller 3, indeholder byte 2 til 5 start-henholdsvis slutadressen for programmet, fulgt af programnavnet. Headertype 1 anvender kun startadressen, hvis det udtrykkeligt specificeres i LOAD-kommandoen:

LOAD "*programnavn*",1,1

Ellers placeres programmet på den adresse, der angives af *Start of BASIC* pointeren.

En type 2 header fortæller, at der er tale om en fil, der indeholder data, som er skrevet med PRINT# kommandoen. Data skal læses med enten GET# eller INPUT# kommandoen. Det ønskede antal bytes overføres direkte fra bufferen til den anvendte variabel. Når der ikke er flere bytes i bufferen, stoppes processen midlertidigt mens næste datablok indlæses i bufferen, hvorefter processen genoptages.

## Programmer på kassette

Når man SAVE'r et program til kassette fastlægges header typen. Der findes følgende muligheder:

### Programfiler:

SAVE "programnavn",1	SAVE's som normalt BASIC program
SAVE "programnavn",1,1	SAVE's som maskinkode (header 3)
SAVE "programnavn",1,2	som BASIC-program, men med EOT
SAVE "programnavn",1,3	som maskinkode, men med EOT

### Datafiler:

OPEN 1,1,0,"programnavn"	Abner fil for læsning
OPEN 1,1,1,"programnavn"	Abner fil for skrivning
OPEN 1,1,2,"programnavn"	do. men med EOT til slut.

EOT er en speciel blok, som indikerer *end of tape* - slutningen på båndet. Når computeren læser denne blok, fås fejlmeldingen **FILE NOT FOUND**. EOT mærket anvendes primært til at markere, at der ikke findes flere filer på båndet. Dermed undgås, at computeren søger videre på et bånd, som ikke indeholder flere informationer!

Når CLOSE kommandoen anvendes til lukning af en datafile, sættes automatisk et *end of file* mærke (CHR\$(0)), som fortæller computeren at eventuelle efterfølgende bytes i tape-bufferen ikke har nogen betydning.

## 6.3 STATUS variabel og kassette

Commodore computerne er udstyret med en særlig variabel - ST - der er forbeholdt til intern brug i computeren. Variablen kaldes også STATUS, og den reflekterer situationen efter *sidste* IN/OUTPUT operation - herunder kassette-operationer. F.eks. som herunder:

```
760 GET#2,A$:S2=ST
770 GET#3,A$:S3=ST
```

Arbejdes der med flere logiske filer på samme tid, bør man lagre STATUS variabelen efter hver IN/OUTPUT operation. ST variabelen kan ikke anvendes til lagring af data, ej heller kan den tildeles en værdi via et BASIC-program. Forsøges dette fås fejlmeldingen **SYNTAX ERROR**.

STATUS-variablen er en *intern kontrol-variabel* i computeren. Variablen afspejler en række helt specifikke tilstande. Variablen ST er en heltalsvariabel, hvor hvert bit afspejler en ganske bestemt tilstand, der er indtruffet. I forbindelse med kassettebåndoptageren kan den antage følgende værdier:

BIT	VÆRDI	BETYDNING
-	0	Alt er i orden
0	1	Anvendes ikke
1	2	Anvendes ikke
2	4	Kort blok
3	8	Lang blok
4	16	Alvorlig læsefejl
5	32	Check-sum fejl
6	64	End of file
7	-128	End of tape

EOF-bittet sættes til 1 (64), når det sidste data-byte i en fil læses. F.eks.

```

100 OPEN 7,1,2,"TEST"
110 PRINT#7,"Ø123456789"
120 CLOSE 7
130 N=Ø
140 OPEN 7,1,Ø"TEST"
150 GET#7,A$
160 S=ST
170 N=N+1
180 PRINT N,A$
190 IF S=Ø THEN 150 (eller IF (S AND 64)=Ø THEN 150)
200 CLOSE 7

```

Husk, at EOF indikerer, at sidste byte i filen (TEST i dette tilfælde) er indlæst - *ikke*, at der er læst forbi, sidste byte i filen. Derfor kontrolleres STATUS-variablen først i linje 190, hvor det afgøres om indlæsningen skal fortsætte eller ej. Bemærk også, at der ikke gives nogen form for fejlmelding, hvis man forsøger at læse forbi sidste byte i en fil! Det er derfor absolut nødvendigt at kontrollere statusvariablen, hvis man ikke er 100% sikker på, hvor lang den fil er, der skal indlæses!

EOT mærket anvendes udelukkende til indikation af, at der ikke findes flere filer på båndet, mens computeren søger efter en fil af et bestemt navn.

Fejltyperne *kort* og *lang blok*, fortæller at der er fejl i de datablokke, der findes på båndet. *Kort* blok fortæller, at computeren har fundet færre bytes end beregnet og *long* blok, at den indlæste blok er længere end tape-bufferen. En tape-blok har *altid* samme længde, men indholdet kan variere. Det aktuelle data-indhold kan være mindre end blok-længden, og det angives med *end of file* mærket. Fejlmeldingerne *kort* og *lang blok* kan opstå, hvis båndet ikke løber perfekt f.eks. p.gr. af "båndsalat" m.m.

Check-sum fejl fortæller, at der er afvigelser i de indlæste data i forhold til det ventede. Hver gang computeren skriver en blok til kassette, ledsages denne blok af et specielt testbyte - kaldet checksum. Indholdet i dette byte afhænger af de datainformationer, der findes i hver enkelt blok. Under indlæsningen vil enhver afvigelse fra det ventede give denne fejlmelding. Oftest vil denne fejl indikere, at der er fejl på båndet, eller at tonehovederne er snævsede, så de data, der indlæses, ikke svarer til dem, der reelt findes på båndet. Rens tonehovederne og prøv igen. Forsvinder fejlen ikke, tjah...så eksisterer der et lille problem - medmindre man har forsøgt at læse en programfil, som om der var tale om en datafil.

## Programmer på kassette

EOT-indikatoren fremtvinger fejlmeldingen **DEVICE NOT PRESENT**, og stopper straks programkørslen. Mærket har kun betydning for **LOAD** og **VERIFY** kommandoerne, men laver altså *ravage*, hvis vi prøver at læse EOT-blokken på kassette! Dette sker, uanset om man har en åben kanal til diskette eller printer. Commodore computeren er ikke i stand til at skelne, så lad være med at læse flere bytes, når du konstaterer at filen er slut *end of file*! Ellers kan det koste data-tab i disk-stationen!!!!

Den sidste indikator i **STATUS** variabelen fortæller, at der er noget grueligt galt, og indikatoren anvendes primært til et internt check under **LOAD** og **VERIFY**.

## Kapitel 7

### Commodore og disk

Springet fra kassettebåndoptager til disk er større end de fleste kassette-brugere forestiller sig. Er man først i besiddelse af en eller flere Commodore 1541 disk-stationer, åbner en helt ny verden sig. Programløsninger, man forud kun kunne drømme om, kan nu klares i en håndvending - sådan da.

Commodore 64 er nemlig ikke ligefrem udstyret med den mest bekvemme form for disk-programmering, men har man først lært de små tricks, der skal til, er mulighederne enorme. Er man den stolte ejer af en Commodore Plus/4 eller C16 er livet langt lettere. Disse computere er nemlig udstyret med samme disk-BASIC, som findes i Commodore's professionelle maskiner. Bortset fra dette, er der dog ingen større forskelle mellem Commodore 64 og de nyere Commodore-maskiner. F.eks. kan et BASIC-program, der er skrevet til Commodore 64 anvendes på de to andre computere uden problemer. Selvom Plus/4 og C16 er udstyret med langt mere bekvemme disk-kommandoer, sluger maskinerne også de mere "primitive" Commodore 64 kommandoer uden at kny.

Commodore's 1541 disk-station er en såkaldt 'intelligent' disk-station. Det betyder, at disk-stationen er udstyret med sit eget "programsprog", og at den arbejder uafhængigt af computeren. Har man overført sine data og kommandoer til disk-stationen, kan computeren fortsætte sit arbejde, uafhængigt af disk-stationen. Da disk-stationen ikke ligefrem hører til de hurtigste i denne verden, er det til tider en stor fordel - men ulemper findes også.

Den største fordel er, at diskoperativsystemet - det program, som styrer disk-stationen - ikke optager plads i computerens hukommelse. Kun de allermest nødvendige rutiner til kommunikation med disk-stationen, findes internt i Commodore 64 - resten er placeret i disk-stationens 16K ROM. I Commodore Plus/4 og C16 er der også tænkt lidt på programmørens bekvemmelighed, men bortset fra dette, er der ingen forskelle.

Dette kapitel beskriver de mere generelle sider i anvendelsen af Commodore's 1541 disk-station, mens de to følgende kapitler beskæftiger sig med de detaljer, som er specifikke for henholdsvis BASIC 2.0 (Commodore 64) og BASIC 3.5 (Commodore Plus/4 og C16). Sidstnævnte afsnit kan også anvendes som inspirationskilde for ejere af de større Commodore-maskiner, der er udstyret med BASIC 4.0.

### 7.1 Commodore 1541 - specifikationer

Commodores disk-station er også i dag markedets billigste, og sammenlignet med mange af konkurrenterne er kapaciteten god. Eneste større ulempe er overførselshastigheden. Kun omkring 400 bytes pr. sekund kan overføres mellem computer og disk-station. Årsagen er, at Commodore har valgt at benytte en seriel kommunikation mellem disk-station og computer, i stedet for det normale - parallel overførsel.

#### Tekniske specifikationer:

Maximal kapacitet .....	178.848 bytes pr. diskette
- ved sequentielle filer ....	168.656 bytes pr. diskette
Relative filer - maksimalt ....	167.132 bytes
- max. antal poster .....	65.535 pr. fil
Directory .....	10.192 bytes pr. diskette
- max. antal filer .....	144 stk. pr. diskette
Antal spor (tracks) .....	35 stk./single sided
Antal sektorer pr. spor .....	17-21 stk. pr. spor
Antal blokke maksimalt .....	683 stk. pr. diskette
- excl. directory .....	664 stk. pr. diskette
Bytes pr. sektor/blok .....	256 stk. max.

Microprocessor .....	6502	(1 stk.)
Porte, timere m.m. ....	6522 VIA	(2 stk.)
In-/output-buffer .....	2K RAM	
Operativ-system .....	16K ROM	
Kommunikation .....	IEC-seriel	

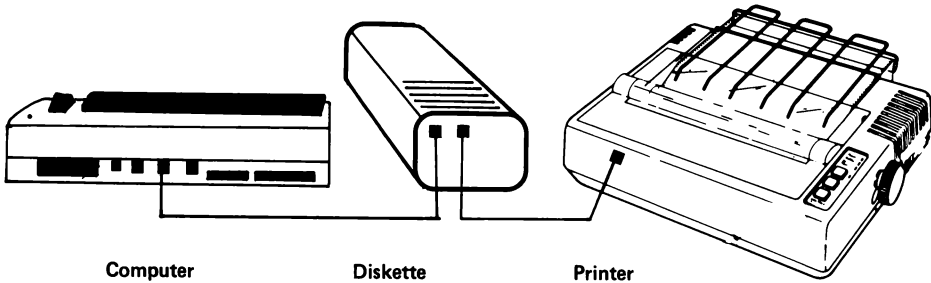
Diskette-type: 35/40 tracks, single sided, single density

### 7.2 Devicenumre og disk

Når du modtager din Commodore 1541 disk-station, er den indstillet til at arbejde som device nummer 8 - ligesom Commodores større disk-stationer (læs mere om devicenumre på side 25 i Commodore TASTATUR og SKÆRM).

Det er ikke noget problem, at koble flere disk-stationer sammen på en Commodore 64. Hver Commodore 1541 disk-station er udstyret med to bøsninger på bagsiden - og der er ingen forskel på, hvilken af de to bøsninger, du benytter til kablet mellem din Commodore 64 og disk-stationen. Den anden bøsning kan benyttes til forbindelseskablet til den næste disk-station, der igen kan forbindes til en ny disk-station osv. Sidst i kæden tilsluttes printeren, hvis du har en sådan.





### Flere disk-stationer og en printer med Commodore 64

Benytter du mere end en disk-station, skal den ene af de to disk-stationer tildeles et andet device nummer - ellers får du problemer med kommunikationen. Har du kun lånt den ene disk-station, kan du nøjes med at ændre device nummeret i software. Er du den stolte ejer af to disk-stationer, er fremgangsmåden lidt for besværlig i det lange løb. I det tilfælde kan du ændre devicenummeret permanent, ved at afbryde en loddebro internt i den ene disk-station. Du kan læse mere herom i afsnittet **Brug af flere disk-stationer** i det følgende kapitel.

#### 7.2.1 Kommunikationen mellem forskellige enheder

Denne simple serieforbindelse af flere perifere enheder, skyldes anvendelse af en særlig kommunikations-protokol mellem de forskellige apparater. Denne kommunikations-protokol er egentlig ikke andet end et simpelt sprog, der tillader udveksling af meddelelser mellem de enkelte enheder i systemet.

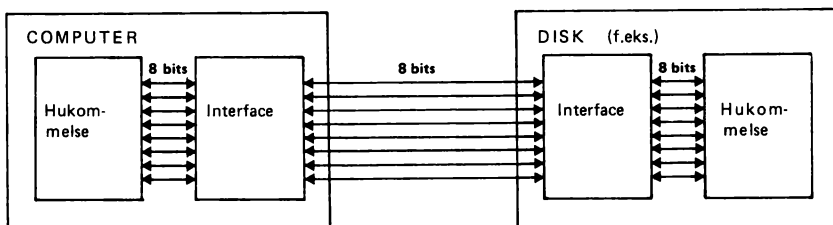
Alle enheder "lytter" på en speciel signal-linje, der signalerer, om der udsendes meddelelser eller ej. Er der tale om en meddelelse, læses denne af *alle* tilsluttede enheder, men kun den enhed, der adresseres (device nummer) reagerer på meddelelsen. De øvrige enheder ignorerer den efterfølgende kommunikation (data), indtil der igen udsendes en meddelelse.

Alle enheder i systemet kan udsende meddelelser til brug for alle andre enheder i systemet, men den overordnede styring og kontrol ligger altid hos Commodore 64 (eller Plus/4 og C16).

Det anvendte kommunikations-system har store ligheder med den protokol (det sprog), som de professionelle Commodore datamater benytter. Eneste større forskel er, at "de store" maskiner anvender parallel kommunikation.

### 7.2.2 Parallel kommunikation

Ved parallel kommunikation overføres alle bits i et byte (læs mere om bits og bytes i kapitlet **Programmering i maskinkode**) samtidigt mellem computeren og det tilsluttede udstyr - f.eks. en printer, en disk-station eller et måleinstrument. Dette kræver mindst 8 signal-ledninger, samt diverse kontrol-ledninger, som sikrer, at kommunikationen foregår efter hensigten. Der findes flere standardiserede former for parallel-forbindelser - bl.a. IEC/IEEE 488, som Commodore anvender på de større datamater og Centronics-porten, der udelukkende anvendes til envejs-kommunikation med printere og plottere (Se afsnittet **Centronics interface** senere i bogen).

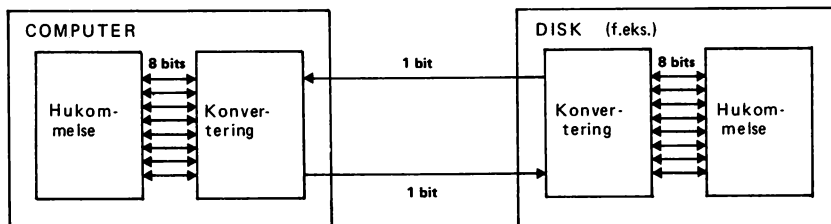


#### Princippet bag parallel kommunikation

Selvom en parallel-forbindelse er enklere at styre - set fra et software-synspunkt - er den dyrere at benytte, idet kabler og dertil hørende stik, samt den nødvendige ekstra elektronik i både computer og disk-station jo også skal betales. Commodore valgte en billigere løsning - nemlig:

### 7.2.3 Seriel kommunikation

Ved seriel overførsel af data benyttes i princippet kun en signal-ledning for overførsel af data i een retning. Hvert byte oversættes 'oversættes' til den tilsvarende følge af bits, som sammen med en række kontrolbits, sendes enkeltvis mellem f.eks. computer og disk-station. Dette medfører naturligvis, at kommunikationen foregår væsentligt langsommere end den tilsvarende parallelle kommunikation - typisk omkring 10 gange langsommere.



### Princippet bag seriel kommunikation

Seriel kommunikation anvendes til en lang række formål, og flere standarder findes på området. Blandt de mest kendte er RS-232, som i vid udstrækning anvendes til kommunikation mellem computere og printere, computere og modems - ja endda i kommunikation mellem to computere (Se afsnittet RS-232 interface senere i bogen).

### 7.3 Disketter som lagermedie

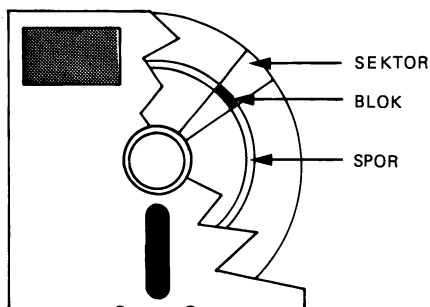
Både disketter og kassetter er magnetiske lagermedier, og har i princippet de samme egenskaber. Forskellene skyldes i praksis udelukkende de fysiske dimensioner - og ikke de til orientalsk mystik grænsende egenskaber, som mange "halvstuderende røvere" (både i og udenfor branchen) elsker at udbasunere for at demonstrere deres fænomenale kundskaber. Der er ingen hindring for at formatte og benytte kassettebånd på samme måde som disketter, blot er det ikke særligt praktisk.

Kassetter er bedst egnede til lagring af informationer, som skal afspilles ud i et, efter hinanden. Om der er tale om musik eller computer-data er i den forbindelse underordnet. Ønsker man at springe rundt mellem informationerne i vilkårlig rækkefølge, er en plade mere praktisk. Det er meget enkelt at afspille melodierne på en LP i lige nøjagtig den rækkefølge, man ønsker det, hvor det er en helt anderledes besværlig sag, at piske en kassettebåndoptager frem og tilbage mellem forskellige melodier - især, hvis man ikke gider at vente, mens båndoptageren spoler frem og tilbage.

### 7.3.1 Disketten inddeles i spor

En disk-station er ikke andet end en meget avanceret, automatisk pladespiller. En pladespiller, som hurtigt og præcist kan flytte nålen frem og tilbage mellem de forskellige melodier - eller brudstykker af en melodi. I disk-stationen er pick-up'en dog erstattet af et magnethoved, og pladen (disketten) er ikke udstyret med riller, men kun en glat magnetbelægning. I stedet for riller, er armen udstyret med en motor, som med stor nøjagtighed placerer hovedet et antal forudbestemte steder på disketten. Disse kaldes spor - på engelsk tracks.

Hvert af disse spor, svarer til en ring i diskettens overflade. Sporene er ikke afmærkede på diskettens overflade, men anvendes udelukkende til at illustrere det antal positioner, som disk-stationens magnethoved kan indtage. Commodore 1541 disk-stationen er udstyret med 35 spor, mens andre disk-stationer kan have f.eks. 40, 77, 80 eller 96 spor. Commodore 1541 benytter kun den ene side af disketten, men disk-stationer, som udnytter begge sider af disketten, er ingen sjældenhed i dag.



### Diskettens inddeling i spor og sektorer

Disketterne produceres i flere kvaliteter, afhængigt af de krav, disk-stationen stiller. Holder man sig til et anerkendt mærke, vinder man egentlig ikke noget ved at købe disketter af bedre kvalitet. Man kan ikke undgå sikkerhedskopiering, disk-stationen bliver ikke mere pålidelig - det eneste man opnår, er i grunden kun et større hul i tegnebogen. En anden ting er så, at enkelte fabrikanter ikke leverer disketter af single-sided type. Grundene hertil er udelukkende praktiske - flere typer er lig med større lageromkostninger.

### 7.3.2 Diskette-typer

De forskellige diskettekvaliteters betegnelser er opført i tabellen herunder:

sider	spor	forkortelse	betegnelse
1	35/40	SS SD	single sided/single density
2	35/40	DS SD	double sided/single density
1	77/96	SS DD	single sided/double density
2	77/96	DS DD	double sided/double density

Enkelte diskettetyper er udstyret med en særlig tæt pigmentering, som tillader lagring af meget store datamængder - typisk omkring 1 megabyte eller mere. Ofte bærer disse betegnelsen "Quad-density".

Disketter fås i flere størrelser. Den type du skal have, har en diameter på 5 1/4 tomme - også kaldet mini-disketter, for at adskille dem fra 8 tommer disketten, der er disketternes "grand old man". På det seneste er også typer med mindre diameter dukket op i handelen, men heller ikke disse micro-disketter kan anvendes på en 1541 disk-station.

### 7.3.3 Formattering af disketter

Før en diskette kan bruges, skal den formatteres. Når en diskette formatteres, indspilles en række kontrolinformationer på diskettens overflade. Alle tidligere informationer slettes!

Under formatteringen opdeles hvert spor på disketten i en række sektorer eller blokke, som hver kan indeholde et forud fastlagt antal bytes. På Commodore 1541 kan hver sektor rumme 256 bytes, men dette er ikke nogen universel værdi. Andre disk-stationer kan anvende sektorstørrelser på 128 bytes, 512 bytes eller 1024 bytes (se illustrationen på side 94).

Hver sektor indeholder foruden det egentlige dataområde en række kontrolinformationer, som letter hovedets positionering - bl.a. indspilles spor og sektornummeret under formatteringen, så disk-stationen altid er klar over, hvor hovedet befinder sig! Mangler disse informationer, giver disk-stationen en fejlmelding, og medmindre du selv er i stand til at programmere disk-stationens controller (absolut kun noget for virkeligt erfarne maskinkode programmører), er alle data på diskettens tabt!!!

Med alt dette in mente, er det ikke svært at indse, hvorfor man ikke uden videre kan tage en diskette fra en anden computer, og aflæse den i et Commodore 1541 diskdrev. Det hører faktisk til undtagelserne, at noget sådant kan lade sig gøre - også mellem Commodores disk-stationer. En næsten problemfri udveksling af disketter kan kun lade sig gøre mellem de følgende Commodore disk-stationer:

Commodore 1540  
Commodore 1541  
Commodore 2031  
Commodore 4040

Disketter "indspillet" på Commodore 2040 kan "afspilles" på et 1541 diskdrev - det omvendte er ikke tilfældet.

#### 7.4 Vær god ved dine disketter

Disketter er følsomme naturer. De finder sig ikke i alt. Ved den mindste form for overlast, strejker de - oftest med det resultat, at indholdet ikke kan læses. Det er aldrig særlig rart at opleve den situation - og da slet ikke, hvis man har "glemt" at fremstille en sikkerhedskopi. Derfor, hvis du vil undgå alt for megen spænding og ophidselse i omgangen med disketter (og kassetter for den sags skyld), så:

**HUSK, at fremstille sikkerhedskopier -  
hver gang du foretager ændringer på en diskette!**

Udover denne sikkerhedsforanstaltning, der i høj grad er skabt, fordi vi mennesker har en tendens til først at tænke os om, *efter* vi har trykket på RETURN-knappen, findes en række regler, der sikrer disketten maksimal levetid, og derfor kan være nok så fornuftige at følge.

##### **Diskettens fjender:**

##### **Støv**

Læg altid disketten tilbage i lommen, når den ikke anvendes - allerhelst tilbage i opbevaringsboksen.

##### **Fedt og fugt**

Berør aldrig diskettens overflade med fingrene. Ligeledes er det en dårlig ide, at placere fedtemadder, kaffekopper, askebægre o.l. ovenpå disketten - uanset om den er i lommen eller ej. Husk, at hovederne i disk-stationen skal renses med jævne mellemrum. Specielle rensedisketter kan købes til formålet.

##### **Temperatur**

Undgå at opbevare disketter i meget varme omgivelser, direkte sollys m.m. Stærke temperatursvingninger er heller ikke af det gode. Husk også, at disketter opfører sig ligesom briller, når de flyttes fra kølige omgivelser ind i varmen. De dugger, og så må de ikke anvendes, før

duggen er fordampet igen.

### **Mekanisk påvirkning**

Disketter må ikke bøjes, knækkes, vrides eller på anden måde udsættes for mekanisk overlast. Skal du skrive på etiketten, må det ikke ske med kuglepen eller blyant - brug en blød filtpen i stedet.

### **Magnetisme**

Disketter må ikke placeres ovenpå dit TV, din radio, dine højttalere eller ovenpå en strømforsyning. Det er heller ikke nogen god ide, at lægge disketten ovenpå disk-stationen. Ligeledes bør du undgå, at tænde eller slukke for disk-stationen, mens der er en diskette i drevet.

### **Nysgerrighed**

Disketten må aldrig tages ud af sit hylster.

Skulle uheldet alligevel ske, så kasser disketten med det samme. En beskadiget diskette udgør en akut fare for disk-stationens liv og helbred. Der er jo ingen grund til at risikere en kostbar reparation, bare fordi du vil spare 30-40 kr. Du har selvfølgelig husket at fremstille en back-up, så den side af sagen skulle jo være unødvendig - ikke?

Den sidste fjende, du møder, retter sig ikke så meget mod selve disketten, som mod indholdet. Fjenden hedder *dårlig hukommelse*. Sørg altid for, at dine disketter er forsynet med en etiket, som fortæller en lille smule om indholdet. Så undgår du - normalt - at formatere en diskette, som rummer vigtige programmer og data. Som en ekstra sikkerhed, kan du foretage en "write protect" af de disketter, som er særligt værdifulde. Du skal blot klæbe et af de medfølgende små mærker over den lille udskæring i siden. Så kan disketten ikke slettes eller formatteres. Hvis du løber tør for mærker, må du ikke anvende almindeligt tape - enkelte af typerne "sveder" nemlig, når de får det varmt, og indeni 1541 disk-stationen er der altid varmt.

Følger du **ALLE** disse regler, vil du næppe opleve problemer med dine disketter!

## Kapitel 8

### Lagring af programmer på diskette

#### 8.1 Directory og filer

Når et program lagres på disketten, opføres forskellige informationer om programmet i diskettens directory. Det er disse informationer, som gør det muligt for diskoperativsystemet at finde programmet igen, når det skal indlæses i computeren.

Diskettens directory er operativsystemets telefonbog. I directory'et findes programmets navn og placering på disketten (adressen), samt yderligere oplysninger om fil-type (program, sekventiel, user eller relativ fil) og det antal blokke, som programmet belægger.

I en særlig del af directory markeres hver enkelt blok på disketten, som er belagt med filer. Denne del kalder Commodore for *Block availability map* - forkortet til BAM. BAM'ens mission her i livet er konstant at holde diskoperativsystemet orienteret om, hvilke sektorer eller blokke, som ikke må indspilles med nye informationer.

Enhver fejl i BAM'en kan have særdeles ubehagelige konsekvenser, hvis man ikke opdager fejlen i tide. Programmer og data kan blive helt eller delvist overspillede med andre programmer eller data, og det er ikke nogen større fornøjelse, hvis man lige har tilbragt timer, måske dage med indtastningen. Derfor den megen snak om sikkerhedskopiering eller back-up, som det også kaldes. Det kan ikke siges kraftigt nok:

#### Lav en sikkerhedskopi af alle dine disketter.

Directory'en, der er placeret i spor 18, kan rumme 144 optegnelser. Hver optegnelse kaldes en fil, og består af ialt 30 bytes. Hver sektor eller blok kan rumme 8 sådanne optegnelser, og til dette formål er der afsat 18 sektorer i directory'en, startende fra sektor 1. Sektor 0 er helliget BAM, diskettens navn og ID-nummer (Se Appendix C Disk format på Commodore 1541).



## 8.2 Læsning af diskettens indhold

Den hyppigst benyttede disk-kommando er den, der tillader os at læse diskettens directory (indholdsfortegnelse). Kommandoen ser således ud:

```
LOAD "$",8
```

hvor 8 er disk-stationens device-nummer (læs mere om devicenumre på side 25 i Commodore TASTATUR og SKÆRM). Kommandoen læser diskettens directory ind i computerens *programhukommelse* - dvs. at *ethvert* BASIC-program overskrives. Directory kan kaldes frem på skærmen med kommdoen:

```
LIST
```

Resultatet kunne se således ud på skærmen:

```
0  "1541TEST/DEMO"  " ZX 2A
13 "HOW TO USE"     PRG
5  "HOW PART TWO"   PRG
4  "VIC-20 WEDGE"   PRG
1  "C-64 WEDGE"     PRG
4  "DOS 5.1"        PRG
11 "COPY/ALL"       PRG
9  "PRINTER TEST"   PRG
4  "DISK ADDR CHANGE" PRG
4  "DIR"            PRG
6  "VIEW BAM"       PRG
4  "CHECK DISK"     PRG
14 "DISPLAY T&S"    PRG
9  "PERFORMANCE TEST" PRG
5  "SEQUENTIAL FILE" PRG
13 "RANDOM FIAL"     PRG
558 BLOCKS FREE.
```

Vær opmærksom på, at disse oplysninger lagres i computerens hukommelse på samme måde, som programlinjer. Derfor skal du *altid* huske at slette hukommelsen med kommandoen **NEW**, inden, inden du starter på indtastning af et program. Indlæses et program med **LOAD**-kommandoen, opstår ingen problemer.

Er der tale om lange programlistninger, kan hastigheden af skærmudskriften bremses ved at holde CTRL-knappen nede. Slippes knappen, forgår listning i normalt tempo igen.

Kommandoen **LOAD "\$",8** kan også anvendes på Commodore Plus/4 og C16, men disse to maskiner er udstyret med en særlig kommando, der tillader udskrivning af diskettens directory direkte til skærmen - *uden* at indholdet i computerens hukommelse påvirkes. Kommandoen ser således ud:

### DIRECTORY (device,drive,"mønster")

Parametrene i parantesen kan udelades. I det tilfælde udskrives directory fra device nummer 8. Parameteren drive, specificerer det ønskede drev i en dobbelt disk-station. På en Commodore 64 ser den tilsvarende komplette kommando således ud:

LOAD "\$mønster",device

Det er ikke muligt at specificere forskellige drive-numre i forbindelse med \$-kommandoen på Commodore 64.

### 8.3 Brug af WILDCARDS eller JOKERE

Udtrykket "mønster", der er anvendt i det foregående, antyder, at det er muligt kun at liste et lille udvalg af filerne på en diskette. Udvalget bestemmes af det mønster af karakterer, brugeren fastlægger.

Indeholder disketten f.eks. en samling beslægtede filer, der er navngivet FAKTURA01, FAKTURA02, FAKTURA03 osv., er det muligt at foretage en listning af disse alene. Til det formål skal vi anvende wildcards - også kaldet jokere - der kan erstatte de karakterer, der er forskellige i de ønskede filer. Kommandoerne:

LOAD "\$FAKTURA\*",8<sup>1</sup> eller LOAD "\$FAKTURA??",8<sup>2</sup>

vil begge bevirke, at kun de filer, der indeholder teksten FAKTURA i filnavnet, vil blive læst ind i computeren fra diskettens directory. Selvom virkningen kan forekomme ens, er der forskelle mellem de to former. I eksempel 1 vil alle filer med teksten FAKTURA blive læst, uanset filnavnets længde - altså også en fil med navnet: "FAKTURA TIL C&B"! I eksempel 2 vil læsningen blive begrænset til de filer, der indeholder 9 tegn og teksten FAKTURA i de første 7 tegn.

De to wildcard eller joker karakterer således forskellig virkning, og deres anvendelse kan sammenfattes til:

\* - joker

Benyttes i alle de situationer, hvor de filnavne, der søges, kan have forskellig længde, eller hvor længden af filnavnet er ukendt. Karakteren "\*" kan erstatte fra 0 til 16 karakterer (det maksimale antal karakterer, der kan anvendes i et filnavn). F.eks. vil kommandoen LOAD "\$FAK\*",8 læse filer med navne som FAK, FAKULTET, FAKTURA, FAKT.003 etc. fra diskettens directory. En ekstrem anvendelse af "\*" - jokeren er kommandoen LOAD "\$\*",8, der er identisk med kommandoen LOAD "\$",8,

som læser alle navne i diskettens directory ind i computerens hukommelse.

? - joker

Benyttes i stedet for een og kun een karakter i filnavnet. Placeringen i filnavnet kan vælges frit. F.eks. vil kommandoen `LOAD "$S?X",8` læse alle de filer, der indeholder tre tegn, har "S" som første bogstav, og "X" som sidste bogstav - f.eks. filnavne, som SAX, SIX osv. Et filnavn, som SAXO vil ikke blive læst!

### Kombination

Ved at kombinere disse to jokere, kan virkningen forstærkes. F.eks. vil kommandoen `LOAD "$S?X*",8` også læse filnavnet SAXO ind i computerens hukommelse.

Wildcards eller jokere kan anvendes i alle disk-kommandoer, men de skal anvendes med omtanke. Ubetænksom anvendelse - evt. forårsaget af en lille trykfejl, kan få katastrofale konsekvenser i forening med "\*" - jokeren. F.eks. vil følgende kommando:

```
OPEN "SØ:*K",15,8,15
```

slette ALLE filer på en diskette - uden varsel! At meningen var at slette alle filer, hvis navne begyndte med "K" (K\* i stedet for \*K), kan disk-stationen jo ikke vide. Eksemplet illustrerer også, at alle de tegn, der følger en "\*" joker, ignoreres - de er helt uden betydning! (OPEN og S - `SCRATCH` - kommandoerne vil blive behandlet lidt senere i dette kapitel).

#### 8.3.1 Forslag til navngivning af filer

Filnavne kan indeholde enhver karakter, bortset fra tegnene ":", ",", "\*" og "?". Hvert filnavn kan indeholde op til 16 karakterer (tegn, tal eller bogstaver). Benyttes et længere filnavn, sker der ikke andet, end at de overskydende karakterer ignoreres. F.eks. vil følgende kommando:

```
SAVE "DETTE NAVN ER LANGT",8
```

ikke give nogen fejlmelding, men filen vil blive opført i diskettens directory under navnet:

```
"DETTE NAVN ER LA"
```

Bemærk iøvrigt, at mellemrum opfattes som en fuldgældig karakter på linje med karakterer, som 2 og A.

Når filerne skal navngives, bør du følge nogle få, enkle grundregler. Disse

## Lagring af programmer på diskette

er:

- Anvend aldrig kommando-navne - f.eks. LOAD, SAVE, NEW osv. som fil-navne. Navnene er i og for sig gyldige, men en lille trykfejl kan føre mange problemer med sig. Husk gamle Murphy's første lov, der siger "Hvis noget kan gøres forkert, vil en eller anden på et eller andet tidspunkt gøre det". Derfor bør du ikke tillade muligheden!
- Anvend fornuftige, beskrivende filnavne. Efter et par måneder, har du sandsynligvis glemt alt om, hvad filnavnet "RSØ1" betyder, hvorimod navnet "REGNSKABØ1-1984" også kan forstås om tre år.
- Beslægtede filer, bør navngives, så de er lette at skille ud fra diskettens directory med en simpel stjernekommando. Det er f.eks. en god ide, at reservere de første 3 eller 4 karakterer i filnavne til disse generelle formål. Her er et par eksempler, på hvordan en sådan løsning kunne se ud:

BRV.xxxxxxx	for breve
FKT.xxxxxxx	for fakturaer
TXT.xxxxxxx	for almindelige tekstfiler
ASM.xxxxxxx	for assembler source-kode-filer
BAS.xxxxxxx	for BASIC-programmer
osv.	

Ønsker du herefter at få en liste af alle dine brev-filer på en diskette, skal du blot indtaste kommandoen:

LOAD "\$BRV\*"

Enkelt og ligetil, men kun hvis du er konsekvent. Ellers er systemet nærmest værdiløst. Benyttér du teknikken, udebliver belønningen ikke. Om du fremstiller dit eget system, eller bygger videre på de viste eksempler, er i den forbindelse underordnet.

### 8.4 Filtyper

Commodore's disk-operativsystem skelner mellem 4 forskellige fil-typer. Disse er:

PRG eller P filer

Programfiler. Kan være BASIC eller maskinkode programmer.

## SEQ eller S filer

Sekventielle filer. Normalt tekst eller datafiler.

## USR eller U filer

USER-filer. Et levn fra Commodore's større disk-stationer og maskiner. På Commodore 1541 har disse ingen særlig mission.

## RRL eller L filer

Relative filer. En særlig filtype, som især anvendes til lagring af data. Der anvendes en særlig adresseringsmetode for lagring og læsning af indholdet i filerne.

Udover disse fil-typer eksisterer der i Commodore's terminologi en femte type - *random files*. Her er der dog ikke tale om nogen egentlig filtype, men om en særlig adresseringsform, som muliggør direkte læsning eller lagring af informationer i hver enkelt sektorblok.

### 8.4.1 Wildcards og filtyper

Ikke i alle situationer er det ønskeligt at indlæse directory for en hel diskette. Udvælgelse med wildcards eller jokere kan hjælpe en hel del, men hvis man kun er interesseret i at vide hvilke programfiler, der befinder sig på disketten, må vi anvende en anden fremgangsmåde.

Commodore 1541 disk-stationen rummer en mulighed for at udvælge bestemte filtyper til directory-listning. Fremgangsmåden er, ligesom mange af de øvrige disk-kommandoer, forbigået i tavshed i Commodore's vejledninger, men derfor er der jo ingen grund til at afstå fra brugen.

Når vi kun ønsker en udskrift af en bestemt filtype i directory'en, kan det ske med følgende kommando:

```
LOAD "$*=filtype",device
```

Filtypen fastlægges efter følgende retningslinjer:

*=S eller *=SEQ	Sekventielle filer.
*=P eller *=PRG	Programfiler.
*=R eller *=RRL	Relative filer.
*=U eller *=USR	User (bruger) filer.

Ønsker vi kun at se hvilke programfiler, der befinder sig i directory, kan vi indtaste følgende kommando:

```
LOAD "$*=P",8
```

## Lagring af programmer på diskette

og kun programfilerne på disketten vil blive indlæst i computeren.

Denne udvælgelsesprocedure kan kombineres med wildcards for filnavnet, hvis vi ønsker en yderligere indskrænkning af informationerne fra direktory. F.eks. vil den følgende kommando udvælge de programfiler, hvis første bogstav er lig med Q:

```
LOAD "$Q*:*=P",8
```

Bemærk, at der *skal* anvendes et semikolon til at adskille parametrene for filnavn og filtype.

### 8.5 Programfiler

Programfiler er principielt identiske med sekventielle filer, blot er indeholdet programmer, af en eller anden art. Programfilerne indtager dog en særstilling, idet computeren automatisk finder ud af placeringen i computerens programhukommelse (Læs mere herom i kapitel 4 side 66). Kommandoerne, der benyttes til indlæsning eller lagring af programmer i forbindelse med en disk-station, er stort set de samme, som når man benytter en kassettebåndoptager, som lagermedie. Selve lagringsprocessen er blot langt hurtigere.

#### 8.5.1 SAVE til diskette

Når et program skal lagres på en diskette sker det med kommandoen:

```
SAVE "drive:programnavn",device
```

hvor *device* i de fleste tilfælde skal være 8, medmindre man benytter flere disk-stationer, eller af en eller anden grund har tildelt disk-stationen et andet device-nummer (Se afsnittet Brug af flere disk-stationer senere i dette kapitel). Benyttes en Commodore 1541 disk-station, skal *drive* vælges til 0, men betegnelsen kan i dette tilfælde udelades helt. Udelades *device*-nummer, forsøger computeren at lagre programmet på kassettebåndoptageren. Device-nummeret skal derfor altid specificeres, hvis man ønsker lagring på diskette. Benyttes en dobbelt disk-station, er det en god ide altid at specificere *drive*-betegnelsen, selvom *drive* 0 altid vælges, hvis intet andet er specificeret. Kommandoerne:

```
SAVE "PROGRAMNAVN",8
```

og

SAVE "Ø:PROGRAMNAVN",8

er således identiske. Eksisterer der allerede et program af samme navn på disketten, giver den røde lampe på disk-stationen sig til at blinke, og en læsning af fejlkanalen, vil give meldingen:

63 FILE EXISTS ØØ ØØ

I modsætning til kassettebåndoptageren, er disk-stationen udstyret med en beskyttelse mod utilsigtet sletning af allerede bestående programmer, men *kun* når der er tale om en SAVE eller WRITE kommando. Ønsker man imidlertid at erstatte et bestående program med en nyere version, kan dette gennemtvinges med kommandoen:

SAVE "@Ø:PROGRAMNAVN",8

hvor tegnet "@" fortæller disk-stationen, at programmet *PROGRAMNAVN* skal lagres på diskette, og hvis et program med samme navn allerede findes på disketten, så skal det erstattes med det nye. Bemærk, at det gamle program først slettes, når det nye er lagret på disketten. Man vil altså få en fejlmelding, hvis der ikke er tilstrækkeligt med plads på disketten til den nye udgave af programmet, før den gamle udgave slettes. I det tilfælde vil den gamle udgave af programmet, dog vedblivende eksistere på disketten, og kan om ønsket slettes med en SCRATCH-kommando, før lagring af den nye udgave af programmet foretages. Den fornuftigste løsning vil dog oftest være, at skifte til en ny diskette.

På Commodore Plus/4 findes en særlig kommando til lagring af programmer på diskette:

DSAVE "*programnavn*"(,D*drive*,U*device*)

hvor *drive* er drivenummeret (0 eller 1), og *device* er devicenummeret - f.eks. 8 - på den benyttede disk-station. Benyttes disk-stationen Commodore 1541, er det ikke nødvendigt at specificere *drive* og *device*.

## 8.5.2 VERIFY af lagret program

Selvom mange sikkert vil føle, at der er tale om overdreven nervøsitet, er det *altid* en god ide, at foretage en efterkontrol af et program, der lige er blevet lagret på en diskette - især efter at SAVE"@Ø:...." kommandoen har været i anvendelse. Trods disketternes større pålidelighed sammenlignet med kassettebånd er der stadig mange ting, der kan kikse. Og det sker gerne, når man mindst venter det! Bitter erfaring, taler på dette punkt! Udføres kommandoen:

**VERIFY "drive:programnavn",device**

umiddelbart efter hver SAVE-kommando, får man klar besked, om lagringen er gået godt eller ej. Har man først slukket for maskinen, eller indlæst et andet program i mellemtiden, er der ikke noget at gøre, hvis uheldet har været ude!

Når VERIFY kommandoen udføres, sammenlignes det lagrede program byte for byte med programmet i computerens hukommelse. Enhver afvigelse vil resultere i fejlmeldingen **VERIFY ERROR**. VERIFY kommandoen kan kun anvendes til kontrol af BASIC-programmer. Sekventielle filer, USER-filer, relative filer og random filer må kontrolleres af det program, som foretager lagringen.

Langt de fleste monitor-programmer til Commodore 64 tillader VERIFY, af lagrede maskinkode programmer. Det er også tilfældet med den indbyggede monitor i Commodore Plus/4, hvor V-kommandoen opfylder dette formål.

### 8.5.3 LOAD fra diskette

Når et program skal indlæses i computerens hukommelse, benyttes kommandoen:

**LOAD "PROGRAMNAVN",device,(virkning)**

hvor *device* er nummeret på den benyttede disk-station - normalt 8 - og *virkning* specificerer, hvordan indlæsningen skal foregå.

Udelades parameteren *virkning* eller sættes den til nul, indlæses programmet, som et normalt BASIC-program - dvs., at adressen specificeret i de første to bytes i programfilen (se kapitlet **Program-filer**) ignoreres. Programmet indlæses automatisk fra starten af BASIC-området.

Er der tale om et maskinkode program, som kræver lagring på en specifik adresse, sættes parameteren *virkning* til 1, hvorefter programmet vil blive indlæst, startende fra den adresse, som er specificeret i de første to bytes i programfilen. F.eks. som i dette eksempel:

**LOAD "PROGRAMNAVN",8,1**

På Commodore Plus/4 findes en særlig kommando til læsning af BASIC-programmer fra diskette (maskinkode programmer skal indlæses på den forud beskrevne måde):

**DLOAD "programnavn" (,Ddrive,Udevice)**

hvor *drive* er drivenummeret (0 eller 1), og *device* er devicenummeret - f.eks. 8 - på den benyttede disk-station. Benyttes disk-stationen Commodore 1541, er det ikke nødvendigt at specificere *drive* og *device*.



## 8.6 Læsning af filer

Da Commodore 1541 er udstyret med sin egen "intelligens", foregår en stor del af disk-stationens arbejde helt uafhængigt af computeren.

Fordelen ved dette system er, at kommunikationen mellem disk-station og computer kan begrænses til udveksling af styringskommandoer og overførsel af de egentlige data. Ser vi bort fra nogle få specielle kommandoer, er det disk-stationen, og den alene, der afgør, hvordan og hvor på disketten data placeres. I forhold til disk-stationen arbejder computeren nærmest som en simpel terminal.

Ulempen ved arrangementet er, at evt. fejl og deraf følgende fejlmeldinger ikke automatisk overføres til computeren, i det øjeblik de opstår. Det er helt og holdent programmørens opgave, at sørge for at evt. fejl konstateres, og at fejlmeldinger læses, så programmet kan reagere herpå. Sker det ikke, er mulighederne for tab af data meget store.

### 8.6.1 Kanaler for data-overførsel

Disk-stationen råder over 16 kanaler, der kan benyttes til kommunikation med computeren. Tre af disse er forbeholdt specielle formål.

KANAL	FORMÅL
0	LOAD
1	SAVE
2-14	data-overførsel
15	Fejlkanal

Kanalnumrene 0 og 1 anvendes af operativ-systemet til overførsel af programmer mellem computer og disk-station, og bør ikke anvendes til andre formål. Fejlkanalen kaldes også kommandokanalen, idet samme kanalnummer benyttes til at sende særlige meddelelser til disk-stationen - f.eks. ved brug af relative filer eller direkte kontrol af disktestationen. De øvrige kanalnumre (2-14) kan benyttes frit til overførsel af data.

### 8.6.2 Åbning af en disk-kanal

Før der kan læses eller skrives til disk-stationen, skal der åbnes en kanal. Dette sker med OPEN kommandoen, der har følgende format, når vi taler om disk-kommunikation:

**OPEN *lf,device,kanal,"PROGRAMNAVN,filtype,virkning"***

hvor *lf* er det logiske filnummer og *device* er disk-stationens nummer (læs mere om *device* side 25) - f.eks. 8. Når en kanal er åbnet, indikeres det ved, at den røde lampe i diskstationen tændes. Det logiske filnummer kan vælges frit i området 1 til 255, men i praksis bør kun numrene 1 til 127 benyttes. Vælges et logisk filnummer over 127, tilføjer computeren automatisk et *line feed* (linjeskift) til hvert *carriage return* ("vogn retur"). Denne funktion udnyttes kun i forbindelse med printere og ved listning til disk, for senere udskrift til printer (se evt. også side 14).

*PROGRAMNAVN*ET, *filtypen* og *virkning* skal altid specificeres, medmindre der er tale om kommunikation til fejkanalen, hvor kanalen åbnes således:

**OPEN *lf,device,15***

Selvom brugeren råder over ialt 13 kanaler til kommunikation med disk-stationen, må kun fire kanaler være åbne på samme tid. Normalt er det kun muligt at åbne tre kanaler for læsning og skrivning til disk, udover fejkanalen. Benyttes *random files* kan alle fire kanaler udnyttes.

FILTYPE	PLADSKRAV	MAX.ANTAL
sekventielle	1 kanal	3 filer
relative filer	2 kanaler	1 fil
random filer	1 kanal	4 filer

Tabellen herover viser, hvor mange kanaler, der belægges af de forskellige filtyper - programfiler, der læses sekventielt, kan betragtes som sekventielle filer. Det samme gælder *USR*-filer.

Relative filer beslaglægger altid 2 kanaler, og der kan derfor aldrig åbnes mere en een relativ fil ad gangen på en Commodore 1541 disk-station. Man har dog mulighed for at åbne een sekventiel fil sammen med en relativ fil.

Filtypen specificeres således i *OPEN*-kommandoen:

sekventielle filer	S eller SEQ
userfiler	U eller USR
programfiler	P eller PRG
relative filer	L eller REL
random files	se disse

Virkemåden beskriver om en fil skal åbnes for læsning eller skrivning. Udover disse to standard-kommandoer kan man yderligere specificere A for *append* (tilføj til slutningen af en fil) og M, der er en special-kommando, som gør det muligt at læse filer, der af en eller anden grund ikke er blevet lukkede på korrekt måde - f.eks. på grund af strømsvigt, fejlbetjening m.m.

---

W	for	WRITE	skrivning
R	for	READ	læsning
A	for	APPEND	tilføjelse
M	for	?	redningsaktion

---

Ønsker man f.eks. at åbne en sekventiel fil for skrivning, ser programsætningen således ud:

`OPEN lf,device,kanal,"FILNAVN,S,W"`

Selve filnavnet kan bestå af op til seksten karakterer, som allerede nævnt.

#### 8.6.2.1 Abning af fil med replace

Eksisterer filen allerede på disketten, kan den ikke umiddelbart overskrives, undtagen hvis der er tale om *random* eller *relative* filer. Ønsker man at overskrive eller ændre programfiler, sekventielle filer eller user filer, kan man enten slette filen, før skrivning af det nye indhold, eller man kan benytte *replace*-tekniken, som er beskrevet nærmere på side 105. Benyttes *replace*, skal `OPEN` kommandoen se således ud:

`OPEN lf,device,kanal,"@Ø:PROGRAMNAVN,filtype,virkning"`

#### 8.6.3 Skrivning til disk

Ønsker vi at skrive data til en fil, skal vi benytte `PRINT#` kommandoen. Bemærk, at kommandoen *altid* skal staves fuldt ud. Det er *ikke* tilladt at benytte forkortelsen "?" i stedet for ordet `PRINT` i `PRINT#` kommandoen! Kommandoen benyttes således:

`PRINT#lf,data`

hvor *lf* er det *logiske* filnummer, der blev benyttet i `OPEN` sætningen. Er filen (`USR`, `PRG` eller `SEQ`) åbnet som en læse-fil (`R`) vil en skrivning til filen give en fejlmelding.

`PRINT#` kommandoen benyttes iøvrigt analogt med den almindelige `PRINT` kommando. Der er således ingen problemer i at kombinere flere variabler eller data i een sætning, og skrivning til disketten sker i helt samme format, som til skærmen - også når der benyttes semikolon eller komma. Herunder er et par eksempler ("\*" repræsenterer et tomt felt (mellemrum eller `CHR$(32)`, og "<" en *carriage return* - `CHR$(13)`).

```
Variabelværdier: AN=23.4567
                  B%=324
                  C$="KARLBØRGE"
-----
PRINT#2,A        *23.4567*<
PRINT#2,-1*A     -23.4567*<
PRINT#2,A;B%     *23.4567**324<
PRINT#2,C$       KARLBØRGE<
PRINT#2,"EJ";C$  EJKARLBØRGE<
PRINT#2,B%;C$    *324*KARLBØRGE<
PRINT#2,"X";"Y"  XY<
PRINT#2,"X","Y"  X*****Y<
-----
```

Bemærk, at beskrivelsen i den engelske vejledning til Commodore 1541 er ukorrekt!!!!

### 8.6.3.1 Listning til disk

I en række situationer, kan man have behov for at liste et program til diskette. Bl.a. anvendes teknikken ofte, hvis en programlistning skal anvendes i et tekstbehandlingsprogram.

Normalt lagres programmer i samme format, som benyttes, når programmet ligger i computerens hukommelse (se side 71). Dette specielle programformat omdannes til klart sprog, når et program listes. Et tekstbehandlingsprogram kan ikke omdanne computerens interne format til klart sprog, og derfor er det nødvendigt at liste programmet til disketten, før det kan anvendes. Fremgangsmåden er meget enkel:

```
100 OPEN 2,8,3,"FILNAVN,S,W"
110 CMD 2
120 LIST
130 CLOSE 2
```

Kommandoen CMD omdirigerer i dette tilfælde udskriften fra skærm til disk-station.

I eksemplet skrives teksten i hver programlinje efterfulgt af *carriage return*. Skal tekstfilen anvendes til senere udskrift på en printer, som kræver at hver linje afsluttes med både *carriage return* og *line feed*, kan det logiske fil-nummer (her 2) i linjerne 100, 110 og 130 erstattes med en værdi, der er større end 127. I langt de fleste tilfælde er eksemplet herover dog at foretrække, idet teksten kan anvendes både til printere, der klarer sig med et *carriage return* alene, og til printere, der kræver et *line feed* også. I sidstnævnte tilfælde skal printerporten blot åbnes med et logisk fil-nummer større end 127, som i eksemplet herunder:

```

100 REM UDSKRIFT AF TEKST FRA DISK TIL
110 REM PRINTER MED EKSTRA LINE FEED
120 OPEN 2,8,3,"FILNAVN,S,R"
130 OPEN 128,4,7:REM PRINTER
140 GET#2,A$
150 S=ST      (Læsning af STATUS - se dette)
160 PRINT#128,A$;
170 IF S=0 THEN 140
180 CLOSE 128
190 CLOSE 2
200 END

```

I linje 170 kontrolleres om værdien i status-variablen (ST) er forskellig fra nul, hvilket bl.a. indikerer at den sekventielle fil ikke rummer flere data!

#### 8.6.4 Læsning fra disk

Filer læses fra disk efter samme retningslinjer, som fra kassettebåndoptager. Blot eksisterer der ikke de samme restriktioner angående data - f.eks. hersker der ingen nævneværdige problemer med at tilbagelæse ASCII-værdien 0.

##### 8.6.4.1 INPUT# kommandoen og disk

INPUT# kommandoen benyttes til at tilbagelæse data fra disketten, og indlæsningen i variabler sker analogt til INPUT kommandoen. Bemærk, at alle tegn i kommandoen INPUT# har betydning, og skal indtastes fuldt ud, hver gang! Den forkortede udgave af INPUT efterfulgt af tegnet #, vil give en fejlmelding. Computeren bliver også forvirret, hvis der optræder et mellemrum mellem ordet INPUT og tegnet #.

INPUT# kommandoen har følgende format:

INPUT#If,variabel,variabel,variabel....

Kommandoen kræver, at følgende betingelser er opfyldt:

- Numeriske variabler kan kun indlæse talværdier. Er der tale om integer variabler eller arrays, skal værdien befinde sig i området -32767 til +32767.
- Numeriske variabler skal være lagret med efterfølgende komma eller carriage return - CHR\$(13) - for at kunne tilbagelæses til numeriske variabler. Sker dette ikke, kan computeren ikke skelne de enkelte talværdier, og man får en fejlmelding. I disse tilfælde kan tilbagelæsning kun ske med GET# kommandoen eller til strengvariabler. F.eks. kan numeriske variabler, der er skrevet til en fil på en af de

efterfølgende måder, tilbagelæses uden problemer:

```
100 PRINT#2,A
110 PRINT#2,A%
120 CR$=CHR$(13)
130 PRINT#2,A;CR$;B%;CR$;C
140 PRINT#2,A;"",B;"",C;"",D
```

Numeriske variabler, der er lagret på følgende måde, vil enten føre til en fejlmelding, eller give fejlagtige resultater under tilbagelæsningen!

```
100 PRINT#2,A;B;C%;D
110 PRINT#2,A,B,C%,D
```

Bemærk, at kommaadskillelsen i linje 110 *ikke* medfører, at computeren kan skelne variablerne under tilbagelæsningen, selvom det fremgår af manualen!!!! Du kan selv efterprøve sagen med dette lille program:

```
100 OPEN 2,8,3,"TEST,S,W"
110 FOR N=1 TO 4
130 PRINT#2,N,N+2
140 NEXT N
150 CLOSE 2
160 OPEN 2,8,3,"TEST,S,R"
170 FOR N=1 TO 4
180 INPUT#2,X
190 PRINT X
200 NEXT N
210 CLOSE 2
```

- Ikke-numeriske variabler kan kun indlæses i strengvariabler eller -arrays.
- Den maksimale strenglængde, der kan indlæses er 88 tegn - dvs. at senest tegn nr. 89 skal være en *carriage return*. Indeholder teksten, der skal tilbagelæses et komma, vil kun tegnene, *indtil* kommaet kunne udskrives. Men er der mere end 88 tegn til *carriage return*, fås stadig fejlmeldingen **STRING TOO LONG**. Følgende lille program, illustrerer problemet:

```

100 OPEN 2,8,4,"TEST,S,W"
110 PRINT#2,"AAA,";
120 FOR N=1 TO 85
130   PRINT#2,"A";
140 NEXT N
150 PRINT#2,CHR$(13)
160 CLOSE 2
170 OPEN 3,8,4,"TEST,S,R"
180 INPUT#3,A$,B$
190 PRINT A$,B$
200 CLOSE 3

```

Ændres linje 120 til:

```

120 FOR N=1 TO 84

```

opstår problemet ikke! Som det ses, virker et komma, som adskillelse under tilbagelæsningen, men alle karakterer frem til *carriage return* læses ind i bufferen! Kun indlæsningen i variablerne berøres af kommaet!!! Konklusionen herpå må være:

Anvend udelukkende *carriage return* som skillekarakter sammen med PRINT# og INPUT#

#### 8.6.4.2 GET# kommandoen og disk

Har man behov for at læse data, der også indeholder karaktererne *carriage return* og komma, eller omfatter data karakterstreng, der er længere end 88 tegn, skal kommandoen:

GET#*lf*,*strengvariabel*, *strengvariabel*....

benyttes. Betegnelsen *lf* står for det *logiske filnummer*, der er anvendt i OPEN kommandoen. GET# kommandoen indlæser en karakter ad gangen, men til gengæld er det muligt at indlæse alle karakterer fra ASCII 1 til 255. Kun karakterkoden 0 ignoreres af GET#-kommandoen, men det problem omgår man let, hvis man altid anvender en programlinje af formen:

```

10 GET#2,A$:IF A$="" THEN A$=CHR$(0)

```

Af hensyn til program-hastigheden, bør GET# kommandoen kun anvendes, hvor det er absolut nødvendigt - enten for at spare plads på disketten, eller fordi den ønskede programløsning kræver anvendelse af GET# kommandoen. Hastighedsforskellen kan konstateres med dette program:

## Lagring af programmer på diskette

```
100 A$="AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
110 OPEN 2,8,4,"TEST,S,W"
120 FOR N=1 TO 250
130   PRINT#2,A$
140 NEXT N
150 CLOSE 2
160 OPEN 2,8,4,"TEST,S,R"
170 TI$="000000"
180 FOR N=1 TO 250
190   INPUT#2,A$
200 NEXT N
210 PRINT "TID MED INPUT# ";TI$
220 CLOSE 2
230 OPEN 2,8,4,"TEST,S,R"
240 TI$="000000"
250 FOR N=1 TO 10000
260   GET#2,A$
270 NEXT N
280 PRINT "TID MED GET# ";TI$
290 CLOSE 2
300 END
```

Indlæsningen tager rundt regnet fire gange så lang tid med GET# kommandoen, så der er virkelig tid at spare, hvis man planlægger sine datafiler, så INPUT# kommandoen kan anvendes.

### 8.6.4.3 Praktisk brug af GET# kommando - læsning af directory

En ting, man ofte har brug for, er at kunne indlæse diskettens directory i computerens hukommelse. Det muliggør f.eks. brugerstyret udvælgelse blandt flere programmer, data-filer i en database osv. Opbygningen af directory er vist i detaljer i Appendix C.

Directory kan maksimalt rumme 144 optegnelser, og skal vort program være helt idiot-sikkert bør vi reservere plads til alle, hvis vi kan afse pladsen. Det bør gøres i starten af programmet, f.eks. således:

```
100 HF=144
110 DIM DR$(HF)
120 DIM PL$(HF)
130 DIM TP$(HF)
```

Array'et DR\$ anvendes til lagring af filnavnene, PL\$ til lagring af fil-længden, og TP\$ til lagring af fil-typen. Der reserveres plads til ialt 145 optegnelser. Frygter du pladsmangel, kan færre værdier benyttes, men ikke udnyttede elementer i de to strengarrays, optager ikke megen plads i hukommelsen. Ønsker du alligevel, at reservere mindre plads, skal du blot ændre variablen HF til det ønskede antal filer *plus* een.

Elementet DR\$(0) anvendes til lagring af diskettens navn, og variablen PL\$(0) benyttes til summering af længderne af de indlæste filnavne. TP\$(0) anvendes til lagring af diskettens ID og betegnelsen for operativ-systemet (På 1541 er den 2A). Det egentlige program er udformet, som en subrutine, som kan



kaldes på to måder.

Første del benyttes kun, når det er nødvendigt at indlæse indholdet i diskettens directory, f.eks. fordi diskettens indhold er ændret. Anden del af rutinen kaldes hver gang, en udskrift af diskettens indhold ønskes!

```
500 DR$(Ø)="
510 OPEN 8,8,8,"$"
520 FOR N=2 TO 143
530 : GET#8,I$
540 NEXT N
```

Først læser vi forbi alle "uinteressante data". Diskettens navn finder vi mellem byte xxxx og xxxx, blot skal vi huske at ignorere alle bytes med værdien CHR\$(1??). De benyttes kun som fyldkarakterer.

```
550 FOR N=144 TO 161
560 : GET#8,I$:DR$(Ø)=DR$(Ø)+I$
570 NEXT N
```

Nu skal vi have læst diskettens ID og DOS-versionen, som befinder sig på pladserne fra 162 til og med 165 - adskilt af SHIFT-space i første sektor af directory.

```
580 TP$(Ø)="
590 FOR N=162 TO 166
600 : GET#8,I$
610 : TP$(Ø)=TP$(Ø)+I$
620 NEXT N
```

Resten af sektoren indeholder ingen oplysninger af interesse, så dem overspringer vi nu.

```
630 FOR N=167 TO 255:GET#8,I$:NEXT N
```

Nu begynder den egentlige directory. Til indlæsning af denne, benyttes en løkke, som fortsætter indtil STATUS-variablen melder, EOI.

```
640 TL=Ø
650 FOR N=1 TO 18
```

Diskettens directory optager ialt 19 sektorer. Sektor 0 indeholder BAM, diskettens navn, ID osv. Vi skal derfor maksimalt indlæse 18 sektorer. Fil-tælleren TL nulstilles.

De første to bytes i hver sektor, peger mod den næste sektor, der er belagt. Disse to bytes indlæses ikke af dette program.

```
660 : FOR DN=1 TO 8
```

Hver sektor i directory indeholder 8 optegnelser, som vi læser med denne løkke.

## Lagring af programmer på diskette

```
670 : GET#8,I$:IF I$="" THEN I$=CHR$(0)
```

Første karakter beskriver hvilken fil-type, der er tale om. Men før vi tolker denne, skal vi sikre, at vi ikke har læst forbi filens "endestop". Det sker således.

```
680 : IF ST<>0 THEN DN=9:N=19:GOTO 880
```

Ved at sætte løkke-tællerne DN og N til værdierne 9 og 19 er vi sikre på, at løkkerne forlades på en civiliceret måde, hvis vi er nået til enden af filen!!!!

Nu kan vi begynde at tolke fil-typen:

```
690 : A=ASC(I$) AND 7:IF A=0 THEN  
FOR SK=1 TO 29:GET#8,I$:NEXT SK:GOTO 860
```

En fil, der er slettet giver værdien nul, men vi skal huske at overspringe alle bytes i fil-optegnelsen. Er filen ikke slettet, kan der være tale om følgende muligheder:

```
700 : TL=TL+1:REM OPSKRIV FILTÆLLER  
710 : IF A=1 THEN TP$(TL)="SEQ"  
720 : IF A=2 THEN TP$(TL)="PRG"  
730 : IF A=3 THEN TP$(TL)="USR"  
740 : IF A=4 THEN TP$(TL)="REL"
```

De følgende to bytes indeholder spor og sektor nummer på første blok i filen. Informationerne udnyttes ikke.

```
750 : GET#8,I$,I$
```

De følgende 16 bytes indeholder filnavnet, udfyldt med SHIFT mellemrum. Informationerne indlæses direkte. Først sikres, at det array-element, der skal rumme filnavnet, er tomt.

```
760 : DR$(TL)=""  
770 : FOR B=3 TO 18  
780 : GET#8,I$  
790 : DR$(TL)=DR$(TL)+I$  
800 : NEXT B
```

Filnavnet er nu indlæst. De følgende bytes indeholder data af betydning for *relative filer* og nogle felter m.m., som ikke anvendes. Vi springer dem over.

```
810 : FOR B=19 TO 27:GET#8,I$:NEXT B
```

De sidste to felter i optegnelsen indeholder filens størrelse i formatet LO-byte/HI-byte. Husk problemet med indlæsning af en *nul*karakter.

```

820 : GET#8,I$:LO=ASC(I$+CHR$(0))
830 : GET#8,I$:HI=ASC(I$+CHR$(0))
840 : PL$(TL)=LO+256*HI
850 : PL$(0)=PL$(0)+PL$(TL)

```

Vi har nu fundet den samlede længde af alle filer til dette punkt, og længden af den fil, der er opført i directory. Vi er parat til indlæsning af næste optegnelse. Først skal vi overspringe to tegn, der adskiller hver optegnelse, hvorefter vi kan afslutte løkken.

```

860 : IF DN<8 THEN GET#8,I$,I$
870 : NEXT DN

```

Når alle otte optegnelser i en sektorblok er indlæst, kan vi starte på en ny sektorblok.

```

880 NEXT N:CLOSE 8

```

Nu er alle eksisterende filnavne indlæst, og vi kan udskrive en liste over navnene. Den følgende rutine kan kaldes, hver gang en listning af filnavnene ønskes.

```

890 P=0
900 PRINT CHR$(147);"TRYK PAA O FOR OP, N FOR NED OG S FOR STOP":PRINT
910 IF (P+20)<=TL THEN 930
920 P=TL-20
930 FOR N=P TO P+20
940 : PRINT PL$(N),DR$(N);" ";TP$(N)
950 NEXT N
960 GET A$:IF A$="" THEN 960
970 IF A$="O" THEN P=P+20:GOTO 900
980 IF A$="S" THEN RETURN
990 IF A$<>"N" THEN 960
1000 P=P-20:IF P<0 THEN 890
1010 GOTO 900

```

Som du kan se, er det slet ikke så svært endda. Ønsker du flere fil-oplysninger, f.eks. spor og sektor nummer på første blok i hver fil, er det en smal sag at ændre rutinen, så dette også medtages.

### 8.6.5 STATUS-variablen og disk

Commodore computerne er udstyret med en særlig variabel - ST - der er forbeholdt til intern brug i computeren. Variablen kaldes også STATUS, og den reflekterer situationen efter sidste IN/OUTPUT operation - herunder også disk-operationer. F.eks. som herunder:

```

760 GET#2,A$:S2=ST
770 GET#3,A$:S3=ST

```

Arbejdes der med flere logiske filer på samme tid, bør man lagre STATUS variabelen efter hver IN/OUTPUT operation. ST variabelen kan ikke anvendes til lagring af data, ej heller kan den tildeles en værdi via et BASIC-program. Forsøges dette fås fejlmeldingen SYNTAX ERROR.

STATUS-variablen er en intern kontrol-variabel i computeren. Variablen afspejler en række helt specifikke tilstande, men kan iøvrigt ikke anvendes som en egentlig fejlindikator i forbindelse med disk-stationen.

Variablen ST er en heltalsvariabel, hvor hvert bit afspejler en ganske bestemt tilstand, der er indtruffet. I forbindelse med disk-stationen kan den antage følgende værdier:

BIT	VÆRDI	BETYDNING
-----		
-	0	Alt er i orden
0	1	Time-out læsning
1	2	Time-out skrivning
2-5	?	Ikke definerede
6	64	EOI (end of data)
7	-128	Device not present
-----		

EOI-indikatoren har stor lighed med EOF-indikatoren for kassettebåndoptageren, blot påvirkes EOI ikke af en nul karakter - CHR\$(0) - i de indlæste data. EOI-bittet sættes til 1 (64), når det sidste data-byte i en fil læses. F.eks.

```
100 OPEN 7,8,9,"TEST,S,W"
110 PRINT#7,"0123456789"
120 CLOSE 7
130 N=0
140 OPEN 7,8,9,"TEST,S,R"
150 GET#7,A$
160 S=ST
170 N=N+1
180 PRINT N,A$
190 IF S=0 THEN 150 (eller IF (S AND 64)=0 THEN 150)
200 CLOSE 7
```

Husk, at EOI indikerer, at sidste byte i filen (TEST i dette tilfælde) er indlæst - ikke, at der er læst forbi, sidste byte i filen. Derfor kontrolleres STATUS-variablen først i linje 190, hvor det afgøres om indlæsningen skal fortsætte eller ej. Bemærk også, at der ikke gives nogen form for fejlmelding, hvis man forsøger at læse forbi sidste byte i en fil! Det er derfor absolut nødvendigt at kontrollere statusvariablen - uanset filtype - hvis man ikke er 100% sikker på, hvor lang den fil er, der skal indlæses!

Læser man forbi sidste byte i en fil, forbliver EOI-bittet sat (64), men nu ændres også TIME-OUT for læsning (bit 1) fra 0 til 1. STATUS-variablen antager nu værdien 66 (2+64). Om ønsket kunne man således teste for værdien 66 umiddelbart efter læsning af et byte fra disk, og afslutte alle videre operationer herefter. F.eks. som her:

```
730 OPEN 5,8,6,"TEST,S,W"
740 GET#5,A$
750 IF ST=66 THEN CLOSE 5:RETURN
760 PRINT A$
770 GOTO 740
```

**DEVICE NOT PRESENT** bittet (-128) afslører om en IN/OUTPUT kommando til et device er lykkedes! Fejlmeldingen er ganske naturlig, hvis man har glemt at tilslutte eller tænde for disktestationen, men indikatoren fortæller en hel del mere.

Efter læsning fra en fil (f.eks. GET#) vil STATUS-variablen antage værdien -128, hvis filen ikke eksisterer på disketten, og efter første PRINT# kan man fastslå, om der allerede eksisterer en fil med samme navn! Afhængigt af situationen kan det indikere, at den forkerte diskette i diskdrevet, at man skulle have anvendt *replace* i en OPEN kommando, eller at en "gammel" fil først skal slettes, inden man kan lagre nye data!

Anvendelsen af **DEVICE NOT PRESENT** indikatoren er dog ikke nogen patentløsning. Så længe der skrives til et device opstår der ingen problemer. Gøres der derimod et forsøg på at læse (GET# eller INPUT#) fra et device, der ikke er "aktivt" (f.eks. disk-stationen), mens et andet device er aktivt (f.eks. printer) på den serielle port, vil systemet gå i baglås! I det tilfælde er der kun een udvej, nemlig RUN/RESTORE knap-kombinationen.

Commodore 64 er i stand til at konstatere om et device er i stand til at modtage data, men ikke om det er i stand til at sende data, hvis der er andet udstyr på den serielle port! I det tilfælde giver computeren sig bare til at vente på et svar - uanset hvor lang tid det skal tage! Computeren er, som allerede nævnt i forordet, dum som et brædt!

Er der ikke tilsluttet andet udstyr på den serielle port, kan computeren konstatere **DEVICE NOT PRESENT** tilstanden, inden man prøver at læse fra et ikke-eksisterende device!

Begrundelsen for **WRITE TIME OUT** kendes ikke, idet et forsøg på at skrive til et ikke-eksisterende device, enten sætter **DEVICE NOT PRESENT** indikatoren, eller får computeren til at gå i baglås. Indikatoren ser således ikke ud til at have den store betydning i praksis!

#### 8.6.6 Lukning af en disk-kanal

Når man er færdig med at arbejde med en fil, skal den lukkes. Det sker med kommandoen:

*CLOSE logisk filnummer*

Bemærk, at der ikke opstår nogen fejlmelding, hvis du lukker en fil, der ikke har været åbnet, så det er altid en god ide, at kontrollere, om den røde lampe i diskstationen slukker ved samme lejlighed!

Er filen åbnet som en R (READ) fil, sker der ingen ulykker, hvis man glemmer

at lukke filen. Det gør der derimod, hvis man har skrevet til en fil.

Hver gang, der skrives til en fil, overføres data fra computeren til en buffer i disk-stationen. Hver gang bufferen er fuld, skrives indholdet til disketten. Kommandoen CLOSE tvinger disk-stationen til at skrive indholdet i bufferen til disketten, selvom bufferen ikke skulle være helt fuld. Undlader man at lukke en program-, user eller sekventiel fil, har man ikke blot mistet de data, der befandt sig i bufferen - hele filen er blevet ulæselig (se dog M-kommandoen i afsnittet **Specialkommandoer for fil-håndtering**). Dette markeres med tegnet "\*" foran filtypen, når diskettens indhold listes.

Er der tale om *relative* eller *random* filer, er ulykken ikke helt så stor. I disse tilfælde mistes kun de data, som befinder sig i bufferen. Men det kan jo også være alvorligt nok. Derfor:

### HUSK, altid at CLOSE en åben fil!

Vær opmærksom på, at kommandoerne RUN og CLR lukker *alle* åbne filer, men disk-stationens røde lampe slukker ikke. I praksis betyder det, at disk-stationen har skrevet indholdet i alle benyttede buffere til disketten, men at den *ikke* har fået at vide, at computeren ikke har lyst til videre kommunikation. Ønsker man at afbryde den røde lampe, kan man kort åbne og lukke den benyttede kanal. Det er ikke nødvendigt, at skrive til eller læse fra kanalen. Alternativt kan man tage disketten ud af disk-stationen. Så slukker den røde lampe automatisk.

## 8.7 Læsning af fejlkanalen

Udover at indikere, at disk-stationen er aktiv, benyttes den røde lampe også til at advare om fejl, der opstået under læsning fra eller skrivning til disk-stationen. I den situation giver den røde lampe sig til at blinke, og nu er "dyre råd gode"!

### 8.7.1 Læsning af fejlkanalen på Commodore 64

Denne bemærkning er særlig aktuel i forbindelse med Commodore 64, hvis man ikke har indlæst DOS 5.1, idet fejlkanalens indhold kun kan læses med GET# eller INPUT# kommandoer. Da disse *ikke* kan benyttes "direkte fra tastaturet", må man fremstille et lille program, *inden* indholdet i fejlkanalen kan læses. Programmet kan se således ud:

```
100 OPEN 2,8,15
110 INPUT#2,A$,B$,C$,D$
120 PRINT A$;" ";B$;" ";C$;" ";D$
130 CLOSE 2
```

En alternativ udgave er:

```
100 OPEN 2,8,15
110 INPUT#2,A,B$,C,D
120 PRINT A;B$;C;D
130 CLOSE 2
```

Programmerne indlæser *fejlnummeret* (A\$ eller A), *selve fejlmeldingen* (B\$) - f.eks. **FILE EXISTS** - samt *spor* (C eller C\$) og *sektor* (D\$ eller D) for operationen, hvor disse har betydning. I enkelte disk-meldinger, f.eks. efter sletning af filer, indeholder variablerne C\$ eller C antallet af de slettede filer (Se Appendix B Commodore 1541 fejlmeldinger).

### 8.7.2 Læsning af fejlkanalen på Commodore Plus/4

Ejere af Commodore Plus/4 råder over en enklere metode til læsning af fejlkanalen. Computeren reserverer to variabler udelukkende til fejlmeldinger, nemlig DS og DS\$. Ønsker man en udskrift af indholdet i fejlkanalen, kan det gøres således:

```
PRINT DS,DS$
```

hvor DS rummer fejlkoden og DS\$ selve fejlmeldingen (Se Appendix B Commodore 1541 fejlmeldinger). Metoden tillader læsning af fejlkanalen direkte fra tastaturet, og er således langt mere fleksibel end fremgangsmåden for Commodore 64. Bemærk dog, at det ikke er tilladt at anvende egne variabler med tegnene DS, i dine programmer. Variabelnavne som DS, DS\$, DS... og DS...\$ er reserveret til computerens interne brug!

### 8.8 Specialkommandoer for disk

Udover de almindelige kommandoer, til ind- og udlæsning af data fra en fil, rummer Commodore 1541 disk-stationen et antal kommandoer, som i høj grad forenkler omgangen med disk-stationen. Et fællestræk for alle disse kommandoer er, at de alle skal sendes til disk-stationens kommandokanal. Den normale fremgangsmåde er:

```
100 OPEN 1f,8,15
110 PRINT#1f,"disk-kommando"
120 CLOSE 1f
```

men om ønsket, kan denne alternative fremgangsmåde anvendes:

```
100 OPEN 1f,8,15,"disk-kommando"
110 CLOSE 1f
```

Virkningen er eksakt den samme. I begge tilfælde står betegnelsen *1f* for

*logisk filnummer*, og denne betegnelse vil blive benyttet i de følgende eksempler.

### 8.8.1 Formattering af disketter

Den vigtigste er NEW-kommandoen, som ikke må forveksles med BASIC kommandoen NEW. Kommandoen bevirker, at disketten formatteres, så den kan udnyttes til lagring af data. Denne proces skal altid udføres, inden en diskette kan anvendes. Kommandoen kan anvendes på to måder - enten:

```
11Ø PRINT#lf,"NØ:disk-navn,ID"
```

eller:

```
11Ø PRINT#lf,"NØ:disk-navn"
```

Den første udgave af kommandoen formatterer en diskette fra grunden - dvs. at disketten opdeles i spor og sektorer, der markeres med særlige styreinformationer, som gør det muligt for disk-stationen at placere læse/skrive-hovedet eksakt på det ønskede sted på disketten. Processen sletter alle informationer på disketten, tildeler disketten et navn, og udstyrer den med en ID-kode.

ID-koden kan bestå af en valgfri kombination af to tegn. Denne kode gør det muligt for disk-stationen at afgøre, om disketten er blevet udskiftet med en anden.

ID-koden lagres sammen med diskettens navn på spor 18, sektor 0, som også indeholder en "oversigt" over hvilke af diskettens blokke, der allerede er udnyttet, og hvilke der kan benyttes frit af andre programmer eller data filer. Denne oversigt kaldes BAM eller *Block Availability MAP*. Denne BAM læses ind i disk-stationens hukommelse, og opdateres hver gang, der tilføjes eller slettes en blok på disketten. Skiftes disketten ud med en anden, kan disk-stationen kun konstatere dette, hvis ID-koden er forskellig - ellers tror disk-stationen, at der er tale om samme diskette. Det er derfor meget vigtigt, at disketterne udstyres med *forskellige* ID-koder, ellers risikerer man, at disk-stationen benytter en "forkert" BAM eller indholdsfortegnelse, med deraf følgende katastrofer!

Den alternative udgave af NEW-kommandoen kan benyttes, når en diskette allerede er formatteret - på en Commodore 1541!! I dette tilfælde ændres kun indholdet af spor 18, sektor 0 - dvs. at disketten tildeles et nyt navn, og i BAM'en markeres alle blokke som frie. ID-koden forbliver uændret! Den eneste fordel er den tid, der kræves for operationen. En fuld formattering tager omkring 80 sekunder, mens den alternative fremgangsmåde er overstået på få sekunder.



### 8.8.2 Initialisering af BAM

Benytter man flere disketter med samme ID-kode, er det nødvendigt at meddele det til disk-stationen, når disketterne ombyttes. Det sker med kommandoen:

```
PRINT#1f,"I0"
```

Kommandoen tvinger disk-stationen til at indlæse diskettens BAM i hukommelsen, og er således en sikkerhed mod fejlagtig overskrivning af belagte sektorer. Selvom en forskel i ID-koden mellem forskellige disketter normalt udløser denne proces automatisk, kan det varmt anbefales, at man altid benytter I-kommandoen, når en fil åbnes for *skrivning*. Automatikken er ikke 100 procent pålidelig, så denne fremgangsmåde sparer megen ærgrelse og ophidselse i det lange løb. Især kan det anbefales at udføre kommandoen efter SAVE eller OPEN med replace!

### 8.8.3 Opdatering af BAM

Anvendelsen af replace funktionen (@) medfører ofte, at diskettens indholdsfortegnelse (BAM) kommer i uorden, og skal BAM'en igen afspejle de faktiske forhold på disketten, er det nødvendigt at genskabe den fra bunden. Kommandoen:

```
100 PRINT#1f,"V0"
```

gennemlæser alle aktive filer - uanset type - og markerer hver blok, der er belagt, i BAM'en - både i hukommelsen og på selve disketten. Kommandoen skal også benyttes til at fjerne filer, der ikke er blevet lukkede på korrekt måde - f.eks. fordi man har glemt CLOSE-kommandoen efter en PRINT#-kommando, eller fordi computeren er "gået i bro", strømmen er blevet afbrudt midt i det hele osv. SCRATCH-kommandoen må ikke benyttes til det formål.

OBS: VALIDATE sletter alle random filer fra diskettens BAM!

### 8.8.4 Sletning af filer

Ofte ønsker man at fjerne ældre eller ikke brugte udgaver af programmer eller data filer fra en diskette. Dette kan ske med kommandoen:

```
100 PRINT#1f,"S0:filnavn"
```

## Lagring af programmer på diskette

som sletter en fil - uanset type - med det pågældende navn. Det er muligt at anvende wildcards eller jokere i denne kommando, men det frarådes stærkt, idet følgerne kan være helt uoverskuelige, ved selv den mest banale form for "tyrkfejl". F.eks. vil følgende kommando:

OPEN 15,8,15,"SØ:\*K"

slette ALLE filer på en diskette - uden varsel! At meningen var at slette alle filer, hvis navne begyndte med "K" (K\* i stedet for \*K), kan disk-stationen jo ikke vide.

OBS: SCRATCH-kommandoen må ikke benyttes til at slette ikke lukkede filer. Anvend VALIDATE i stedet.

### 8.8.5 Ændring af filnavnet

Ønsker man at ændre navnet på en fil - uanset type - benyttes kommandoen:

11Ø PRINT#lf,"RØ:nyt navn=Ø:gammelt navn"

Bemærk, at det vil resultere i en fejlmelding, hvis det nye navn allerede eksisterer i forvejen, selvom filtypen skulle være en anden. Generelt gælder det, at der ikke kan eksistere to ens navne på disketten, selvom filtyperne er forskellige.

### 8.8.6 Kopiering af filer

Uanset om man er i besiddelse af en eller flere Commodore 1541 disk-stationer, er det kun muligt at kopiere filer indenfor samme diskette. F.eks. således:

11Ø PRINT#lf,"CØ:ny fil=Ø:gammel fil"

Kommandoen fremstiller en eksakt kopi af den oprindelige fil, blot under et andet navn. Copy-kommandoen rummer dog en ganske særlig mulighed, som kun sjældent benyttes. Det er nemlig muligt at sammenkopiere flere filer under et nyt filnavn (Se også afsnittet om APPEND senere i dette kapitel). Metoden forudsætter naturligvis, at der er plads nok på disketten til at rumme den nye fil, og er kun praktisk anvendelig med sekventielle og user filer, der indeholder tekst. Programfiler kan ikke "sammenkobles" korrekt på denne måde (læs mere om programfiler i et følgende afsnit), selvom alle informationerne samles i en fil!

Fremgangsmåden er:

110 PRINT#1f,"C0:ny fil=0:gammell,0:gammel2, osv."

Det maksimale antal filer, der kan kombineres på denne måde, bestemmes af den ledige plads på disketten og af det maksimale antal tegn, der kan sendes på een gang til kommando kanalen - maksimalt 40 tegn mellem anførselstegnene!

### 8.8.7 Sikkerhedskopiering af disketter

Der findes *ingen* enkel metode til at fremstille en sikkerhedskopi på een 1541 disk-station, og selv med to 1541 disk-stationer er processen ikke helt problemfri.

BACUP-programmet, der gennemgås i det følgende, er i stand til at fremstille en eksakt kopi af en diskette - uanset indholdet. Alle filtyper kopieres uden problemer, slette-beskyttede filer (se et følgende afsnit i dette kapitel) vil også være slettebeskyttede på kopi-disketten, ligesom ikke-lukkede filer og *random filer*, vil være uforandrede på kopidisketten (dog kun hvis linje 52 udelades!). Programmet gør det muligt at fremstille back-up- eller arbejds-kopier med kun een 1541 disk-station. Kopieringen tager omkring 20 minutter - uanset diskettens indhold!!!!

Enkelte programdisketter er beskyttede fra fabrikantens side. Denne kopibeskyttelse vil normalt også være virksom overfor dette back-up-program, men ikke altid!! Det skader aldrig at gøre et forsøg. Lad dig ikke forskrække af et evt. forbigående blinkeri fra diskteststationen - lad blot processen fortsætte, så længe det er muligt. Man kan jo være heldig en gang imellem.

Lad det straks være sagt:

**BACK-UP programmet er kun tiltænkt til fremstilling af sikkerhedskopier til rent personligt brug. Det er ikke lovligt, at fremstille kopier til vennerne eller med salg for øje, medmindre du selv har fremstillet de programmerne, du distribuerer til andre. Dette gælder både privatpersoner, firmaer og naturligvis også undervisningsinstitutioner!!!**

Vær opmærksom på, at der ikke må befinde sig maskinkodeprogrammer i computerens hukommelse, når kopi-programmet startes. Alle former for indstiksmodule (ROM-cartridges) skal fjernes, *inden* programmet anvendes! Det er *altid* en god ide, at afbryde computeren et kort øjeblik, *inden* BACK-UP-programmet indlæses. Hverken NEW eller RUN/RESTORE kan klare opgaven.

Efter anvendelse af kopi-programmet, skal computeren *enten* afbrydes kort, eller følgende programsætning indtastes (RUN/RESTORE er heller ikke tilstrækkeligt i denne situation):

POKE 56,160:CLR

## Lagring af programmer på diskette

•

BACKUP-programmet begrænser BASIC-hukommelsen til 2K (både program og variabler)!

```
1 PRINT CHR$(147);"KOPI-PROGRAM":PRINT
```

Hukommelses-området, der kan udnyttes af BASIC, fastlægges. Området fra adresse 4096 (hex 1000) til 65535 (hex FFFF) reserveres til lagring af data, der indlæses fra disketten. Linje 2 beskytter dette område mod overskrivning!

```
2 POKE 56,16:POKE 55,0:CLR
3 DIM S$(35),T(3)
```

Topværdierne for track (spor) for hver gennemløb indlæses (11, 23 og 35).

```
4 FOR N=1 TO 3
5 : READ T(N)
6 NEXT N
```

Det maksimale sektorantal for hvert spor (track) indlæses. Vær opmærksom på, at arrayet S skal være dimensioneret som et integer eller heltals array - ellers er der ikke plads nok til både program og variabler i computerens BASIC-hukommelse.

```
7 FOR N=1 TO 35
8 : READ A:S$(N)=A
9 NEXT N
```

Maskinkoden, der sørger for maksimal hastighed i ind- og udlæsningen af data fra eller til disketten, placeres i kassettebufferen.

```
10 FOR N=828 TO 916
11 : READ X:POKE N,X
12 NEXT N
```

Der kan vælges mellem backup med kun een disk-station eller to disk-stationer. Devicenumrene fastlægges i linje 18 (to disk-stationer) henholdsvis linje 14 (een disk-station). Variablen "O" indeholder device-nummer, for den disk-station, der indeholder original-disketten, der skal kopieres FRA, og variablen "K" indeholder device-nummer for disk-stationen, der rummer kopi-disketten! Denne skal være formatteret, før kopieringen kan begynde. Diskettens navn og ID-nummer er uden betydning, idet kopien tildeles eksakt samme navn og ID-nummer, som original-disketten.

```
13 PRINT"EEN DISKSTATION (J/N)?"
14 GOSUB 57:IF A$="J" THEN O=8:K=8:GOTO 19
15 IF A$<>"N" THEN 14
16 PRINT:PRINT"PLACER ORIGINAL I DREV 8 OG MAAL-DISKET-TEN I DREV 9"
```

Subrutinen i linje 56 afventer et knaptryk.

```
17 GOSUB 56
18 O=8:K=9
```

Det første spor (track), der læses fra disketten fastlægges.

19 T1=1

Linje 20 fastlægger, at der skal foretages ialt 3 gennemløb (ind/udlæsninger af maksimalt 60 K).

20 FOR N=1 TO 3

Bufferen, der skal rumme indtil 60 kilobytes, der indlæses fra disketten, initialiseres - dvs. at både læse- og skrive-adressen sættes til starten af bufferen. Hver læsning til bufferen eller skrivning til diskette fra bufferen opdaterer automatisk læse og skrive adresserne, så de peger mod næste 256 byte område i hukommelsen.

21 : SYS 828

22 : PRINT CHR\$(147)

Benyttes to disk-stationer (device 8 og 9 - fastlagt i linje 18) udskrives kun een meddelelse, når kopieringen begynder.

23 : IF N=1 AND K<>0 THEN PRINT:PRINT "KOPIERING BEGYNDER"

Benyttes kun en disk-station, startes hvert gennemløb med en opfordring til at indsætte original-disketten i disk-stationen. Subrutinen, der starter i linje 56, afventer et "bekræftende tastetryk".

24 : IF O=K THEN PRINT:PRINT"INDSÆT ORIGINAL":GOSUB 56

Der åbnes en kanal for *random access* til hver enkelt blok på disketten.

25 : OPEN 15,0,15,"IØ"

26 : OPEN 8,0,8,"#"

Track- (spor-) tællerens (TR) arbejdsområde fastlægges. I første gennemløb indlæses sporene 1 til 11, i andet gennemløb indlæses spor 12 til 23, og i tredje gennemløb indlæses spor 24 til 35.

27 : T2=T(N)

28 : FOR TR=T1 TO T2

Sektortællerens (SE) arbejdsområde fastlægges i overensstemmelse med det aktuelle spor-nummer. Afhængigt af det valgte spor indlæses 21 (0-20), 19 (0-18), 18 (0-17) eller 17 (0-16) blokke fra disketten.

29 : FOR SE=Ø TO S\$(TR)

Track- (spor) og sektor-nummer for den blok, der skal indlæses, indikeres på skærmen.

30 : GOSUB 55

I linje 31 indlæses 256 byte fra spor (track) TR, sektor SE til disk-stationens

## Lagring af programmer på diskette

buffer.

```
31 : PRINT#15,"U1";8;0;TR;SE
```

Buffer-pointeren justeres, så den med garanti peger mod første byte (byte 0) i disk-bufferen!

```
32 : PRINT#15,"B-P";8;0
```

En blok på 256 bytes indlæses fra spor TR, sektor SE til det reserverede område af Commodore 64' hukommelse. Efter indlæsningen af en blok, sørger rutinen automatisk for, at indlæsningsadressen for den efterfølgende blok er korrekt. Kaldes rutinen f.eks. 240 gange (det maksimale antal) vil alle 60K RAM fra adresse 4096 (hex 1000) til og med adresse 65535 (hex FFFF) være fyldt med informationer fra original-disketten.

```
33 : SYS 841
```

Sektortælleren opdateres. Maksimalværdien afhænger af arrayet SX (se linje 29), der indeholder det antal blokke, der findes for hver af de 35 spor (tracks) på disketten. Husk, at der tælles fra nul!

```
34 : NEXT SE
```

Spor- (eller track-) tælleren opdateres. Når værdien 11 (første gennemløb), 23 (andet gennemløb) eller 35 (sidste gennemløb) overstiges, er alle blokkene indlæst i Commodore 64'ers hukommelse.

```
35 : NEXT TR
```

```
36 : CLOSE 8:CLOSE 15
```

Foretages BACK-UP på een disk-station (variablerne O og K er identiske, skal original-disketten udskiftes med kopi-disketten. Subrutinen i linje 56 afventer en bekræftelse i form af et tryk på en af tastaturets knapper.

```
37 : IF O=K THEN PRINTCHR$(147):PRINT:PRINT "INDSAET KOPIDISK":GOSUB 56
```

I den følgende løkke udlæses de blokke, der er indlæst i hukommelsen via programlinjerne 25 til 36.

```
38 : OPEN 15,K,15
```

```
39 : OPEN 8,K,8,"#"
```

```
40 : FOR TR=T1 TO T2
```

```
41 : FOR SE=0 TO S*(TR)
```

Udskrift af track og sektornummer for den 256-byte blok, der skrives til disketten.

```
42 : GOSUB 55
```

Buffer-pointeren stilles på første byte i blokken (byte 0).

```
43 : PRINT#15,"B-P";8;Ø
```

En blok skrives til disketten. Hvert maskinkode-kald skriver en ny blok til disk-stationens buffer.

```
44 : SYS 879
```

Linje 45 medfører, at indholdet i disk-stationens buffer skrives til disketten på track (spor) TR, sektor SE.

```
45 : PRINT#15,"U2";8;Ø;TR;SE
```

Sektor-tælleren opdateres i linje 46.

```
46 : NEXT SE
```

Spor- (eller track-) tælleren opdateres. Når værdien 11 (første gennemløb), 23 (andet gennemløb) eller 35 (sidste gennemløb) overstiges, er alle blokkene, der er indlæst i Commodore 64'eren's hukommelse, overført til kopi-disketten.

```
47 : NEXT TR
```

```
48 : CLOSE 8:CLOSE 15
```

TR har værdien 12, 24 eller 36 efter henholdsvis første, andet og tredje gennemløb. 12 og 24 er start-sporene (tracks) for henholdsvis gennemløb to og gennemløb tre. De tilsvarende maksimalværdier er henholdsvis 23 og 35, som det fremgår af DATA-sætningen i linje 59.

```
49 : T1=TR
```

Gennemløbstælleren opdateres.

```
50 NEXT N
```

Det kan være en fordel, at fremstille to udgaver, af programmet. Een udgave med linje 52, som kan kaldes "BACKUP", og en udgave uden linje 52, der kunne kaldes "KATASTROFE". Uden linje 52 og VALIDATE-kommandoen, vil kopi-disketten altid være 100 procent identisk med original-disketten.

```
51 PRINT CHR$(147);"OPDATERING ";
```

```
52 OPEN 15,K,15,"VØ":CLOSE15
```

```
53 PRINT"FINISHED:"
```

```
54 PRINT:END
```

Subrutinen herunder udskriver track- og sektor-nummer for hver blok, der læses fra eller skrives til disketten.

```
55 PRINT CHR$(19);"T:";TR;CHR$(157);" S:";SE;CHR$(157);" ":";RETURN
```

## Lagring af programmer på diskette

Den følgende rutine afventer tryk på en knap (WAIT-kommandoen i linje 57), og indlæser værdien i variabelen A\$. Linjen må ikke forandres!

```
56 PRINT"TRYK PAA EN KNAF, NAAR DU ER PARAT"  
57 WAIT 203,63:GET A$:RETURN
```

DATA-sætningen i linje 59, indeholder det højeste track-nummer, der ind- eller udlæses under de tre gennemløb af læse/skrive-rutinen.

```
58 REM TRACKS  
59 DATA 11,23,35
```

DATA-sætningerne 61 og 62 indeholder de højeste sektornumre (der tælles altid fra 0), for de første 11 spor (tracks) - ialt 231 blokke på 256 bytes.

```
60 REM SEKTORER 1.KOPI  
61 DATA 20,20,20,20,20,20  
62 DATA 20,20,20,20,20
```

DATA-sætningerne 64 og 65 indeholder de tilsvarende maksimale sektornumre for spor (track) 12 til 23 - ialt 240 blokke.

```
63 REM SEKTORER 2.KOPI  
64 DATA 20,20,20,20,20,20  
65 DATA 18,18,18,18,18,18
```

DATA-sætningerne 67 og 68 indeholder de tilsvarende maksimale sektornumre for spor (track) 24 til 35 - ialt 212 blokke.

```
66 REM SEKTORER 3.KOPI  
67 DATA 18,17,17,17,17,17  
68 DATA 17,16,16,16,16,16
```

Linjerne 70 til 81 rummer maskinkoderutinen, der læses ind i kassette-buffere med løkken i linjerne 10 til 12. Vær meget omhyggelig med indtastningen, idet selv den mindste fejl kan få helt uoverskuelige følger!!

Alle elementerne i DATA-sætningerne består af tre cifre, for at gøre listningen nemmere at overskue. Ledende nuller (f.eks. i Ø16) kan uden problemer udelades, men vær ekstra omhyggelig med kontrollen af det indtastede program, hvis denne beslutning tages - man kan så let komme til at slette et ciffer for meget - eller et komma. Det sidste vil ikke altid resultere i en fejlmelding! Vær også opmærksom på, at programmet (incl. variablerne) fylder praktisk taget hele BASIC-området, så der er ikke plads til de store krumspring - medmindre du fjerner alle "unødige" mellemrum, REM'er etc.



```

69 REM IND/UD
70 DATA 169,016,133,252,133,254,169,000
71 DATA 133,251,133,253,096,169,000,133
72 DATA 144,168,162,008,032,198,255,032
73 DATA 207,255,008,120,072,165,001,041
74 DATA 252,133,001,104,145,251,165,001
75 DATA 009,003,133,001,040,200,208,231
76 DATA 230,252,096,169,000,133,144,168
77 DATA 162,008,032,201,255,008,120,165
78 DATA 001,041,252,133,001,177,253,170
79 DATA 165,001,009,003,133,001,040,138
80 DATA 032,210,255,200,208,231,230,254
81 DATA 096

```

Bemærk, at programmet ikke undersøger, om du har placeret den rette diskette i disk-stationen. Det ville ikke have nogen mening, idet original-diskettens navn og ID-nummer kopieres uforandret til kopi-disketten. Det samme gælder alle fejl og ikke-lukkede filer. Intet forandres, hvis linje 52 ikke findes i kopi-programmet! Dette er især en fordel, hvis der er opstået "kludder" i filerne på en diskette. Så har du en mulighed for at fremstille en eksakt kopi, som du kan benytte til en evt. "redningsaktion".

Skulle du ved en fejltagelse få rodet rundt i original- og kopi-disketterne, kan du til enhver tid afbryde processen på det tidspunkt, hvor programmet beder dig om at skifte disketter. Det værste, der kan ske, er at kopien ikke er brugbar - og så kan du jo blot fremstille en ny!!! Kommer du ved en fejltagelse til at skrive data på originaldisketten, sker der i 99,9% af tilfældene ikke andet, end at originaldiskettens data skrives uforandret tilbage på eksakt det samme sted, som de blev læst fra!!! Den sidste 0,1% tager højde for alle de fejl, der altid kan opstå på grund af snavsede hoveder, slid, strømudfald m.m.

Herunder præsenteres maskinkode-delen af programmet. Udskriften følger de sædvanlige LIST-konventioner for maskinkodeprogrammer til 6502/6510 processorer - dvs. at første kolonne rummer linjenummer, anden kolonne (startende med 033C) rummer adressen, tredje kolonne rummer maskinkoden i hex (max. 3 hex bytes), og herefter følger selve assembler-udskriften.

Første del af programmet fastlægger adresserne på de LABELS, der benyttes i programmet.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
181:					CHKOUT = \$FFC9
182:					CHKIN = \$FFC6
183:					CHROUT = \$FFD2
184:					CHRIN = \$FFCF
185:					STATUS = \$90

Programmets startadresse sættes til 828 (33C hex), hvilket er lig med kassettebufferen, som jo alligevel ikke benyttes, mens kopieringen pågår.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
190:	033C				*= 828

## Lagring af programmer på diskette

INIT-rutinen initialiserer hhv. læse- og skrive-bufferens start til hex 1000. Læsebufferens startadresse opbevares i hukommelsen i byte FB og FC (251 og 252), og skrivebufferens startadresse opbevares på pladserne FD og FE (253 og 254) i hukommelsen.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
220:	033C	A9	10		INIT LDA #\$10
230:	033E	85	FC		STA \$FC
240:	0340	85	FE		STA \$FE
250:	0342	A9	00		LDA #\$00
260:	0344	85	FB		STA \$FB
270:	0346	85	FD		STA \$FD
280:	0348	60			RTS

Den følgende rutine læser 256 bytes fra disketten (1 blok). Blokkens spor- og sektornummer fastlægges i BASIC-programmet. Når rutinen forlades opskrives start-adressen på læse-bufferen med 256 bytes (linje 620), således at den næste blok, der indlæses fra disketten ikke kommer til at overskrive den blok, vi allerede har indlæst. Der kan maksimalt indlæses 240 blokke på 256 bytes i en "opvask" (60 kilobyte), før læse-bufferen skal tømmes. Det er BASIC-programmets opgave at sørge for, at denne værdi ikke overskrides. Vær meget opmærksom på dette punkt, idet maskineriet bryder ned, hvis det sker.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
320:	0349	A9	00		READ LDA #\$00
330:	034B	85	90		STA STATUS
340:	034D	A8			TAY

Computeren oplyses om, at vi ønsker at indlæse fra det logiske filnummer 8.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
350:	034E	A2	08		LDX #\$08
360:	0350	20	C6	FF	JSR CHKIN

Selve indlæsningen fra disketten sker i denne løkke (linje 370 til 610). CHRIN-subrutinen henter et byte fra disketten.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
370:	0353	20	CF	FF	LOOP1 JSR CHRIN

Her forberedes udkobling af BASIC- og KERNAL-ROM'erne, samt indkobling af RAM fra adresse D000 til DFFF hex, således at hele hukommelsesområdet fra hex 1000 til hex FFFF består af RAM. Alle interrupts afbrydes i linje 390. Sker det ikke, vil computeren "gå død".

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
380:	0356	08			PHP
390:	0357	78			SEI

Linje 450 til 470 styrer indkoblingen af RAM-området.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
----	-----	-----	-----	-----	-----
440:	0358	48			PHA
450:	0359	A5	01		LDA \$01
460:	035B	29	FC		AND #\$FC
470:	035D	85	01		STA \$01
480:	035F	68			PLA

Det byte, der er indlæst fra disketten, lagres nu på adressen "Y+256\*(FC)".

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
----	-----	-----	-----	-----	-----
520:	0360	91	FB		STA (\$FB),Y

Computerens hukommelse kobles igen tilbage i normal-tilstanden, så fortsat læsning fra disketten kan foregå.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
----	-----	-----	-----	-----	-----
560:	0362	A5	01		LDA \$01
570:	0364	09	03		ORA #\$03
580:	0366	85	01		STA \$01
590:	0368	28			PLP
600:	0369	C8			INY
610:	036A	D0	E7		BNE LOOP1

Når 256 bytes er indlæst (register Y har igen værdien 0) opskrives startadressen med 256 bytes, og programmet returnerer til BASIC.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
----	-----	-----	-----	-----	-----
620:	036C	E6	FC		INC \$FC
630:	036E	60			RTS

WRITE-rutinen udlæser igen indholdet af computerens hukommelse til disketten. Udlæsningen sker i blokke på 256 bytes - svarende til en sektorblok på disketten. Spor- og sektornummer fastlægges i BASIC-programmet, der også skal holde styr på det antal blokke, der er blevet indlæst. Rutinen udlæser slavisk 256 bytes til den logiske fil nummer 9, hver gang den kaldes, og flytter automatisk startadressen 256 bytes frem efter hver udlæsning. Rutinen svarer stort set til READ-rutinen.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
----	-----	-----	-----	-----	-----
670:	036F	A9	00	WRITE	LDA #\$00
680:	0371	85	90		STA STATUS
690:	0373	A8			TAY

Computeren informeres om, at der nu skal skrives til den logiske fil nr.9. (Filen skal være åbnet af BASIC-programmet, ligesom det også er BASIC-programmets opgave, at sørge for at disk-stationens buffer-pointer står på nul.

## Lagring af programmer på diskette

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
700:	0374	A2	09		LDX #\$08
710:	0376	20	C9	FF	JSR CHKOUT

Hovedløkken, der udskriver 256 bytes til disketten.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
750:	0379	08			LOOP2 PHP
760:	037A	78			SEI

RAM-område fra hex 1000 til hex FFFF indkobles.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
800:	037B	A5	01		LDA \$01
810:	037D	29	FC		AND #\$FC
820:	037F	85	01		STA \$01

Et byte læses fra computerens hukommelse.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
860:	0381	B1	FD		LDA (\$FD),Y
870:	0383	AA			TAX

Computeren kobles tilbage i normalstillingen.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
910:	0384	A5	01		LDA \$01
920:	0386	09	03		ORA #\$03
930:	0388	85	01		STA \$01

Et byte skrives til disk-stationen (linje 980).

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
960:	038A	28			PLP
970:	038B	8A			TXA
980:	038C	20	D2	FF	JSR CHR0UT
990:	038F	C8			INY
1000:	0390	D0	E7		BNE LOOP2

Udskriftsbufferens startadresse opskrives med 256 bytes, og der hoppes tilbage til BASIC-programmet. Næste kald til WRITE-rutinen, udskriver den efterfølgende 256-byte blok til disketten.

LIN:	ADDR	MC	MC	MC	ASSEMBLER-KODE
1010:	0392	E6	FE		INC \$FE
1020:	0394	60			RTS

Dette program er dokumentationen til de DATA-sætninger, der befinder sig i

linjerne 70 til 81 i BASIC-programmet, og er tænkt som inspirationskilde for dem, der ønsker at foretage tilpasninger til egne formål. Interesserer du dig ikke for maskinkode - hvad der er yderst forståeligt - er der ingen grund til at udforske den dybere mening i listningen. Programmet bliver hverken værre eller bedre af den grund.

## 8.9 Commodore Plus/4 og C16 disk-kommandoer

Det er muligt at benytte de samme disk-kommandoer, som på Commodore 64, men Plus/4 og C16 computerne råder også over en del af de disk-kommandoer, som findes på Commodores professionelle computere. En detalje, der sikkert vil blive værdsat af mange brugere.

Herunder findes en kort forklaring på de betegnelser, der vil blive benyttet i den følgende gennemgang af BASIC 3.5 kommandoerne.

**Ddrive** Drive-nummer i dobbelt disk-station (0 eller 1).  
Commodore 1541 har *altid* nummeret 0!  
**Udevice** Device-nummeret på den tilsluttede disk-station.

I forbindelse med en 1541 disk-station (device 8) er det ikke nødvendigt, at specificere disse parametre!

### BACKUP

Muliggør fremstilling af en kopi af en diskette på en dobbelt diskstation. Kommandoen kan *ikke* benyttes til kopiering mellem to Commodore 1541 disk-stationer. Kommandoen ser således ud:

BACKUP Ddrive TO Ddrive (,Udevice)

### COLLECT

Kommandoen svarer til VALIDATE kommandoen i forbindelse med Commodore 64. Anvendelse:

COLLECT Ddrive,Udevice

### COPY

Fremstiller en kopi af en fil under et nyt navn (*samme drive*) eller samme navn (*andet drive*). Kopieringen kan kun foretages indenfor samme disk-station (*device-nummer*). Det er tilladt at anvende *wildcards* eller *jokere*. Kommandoen svarer fuldstændig til den førnævnte COPY-kommando; kun syntax'en er anderledes:

COPY (Ddrive,)"gl.navn" TO (Ddrive,)"nyt navn"(,Udevice)

### DIRECTORY

## Lagring af programmer på diskette

Udskriver diskettens directory på skærmen, uden at påvirke indholdet i hukommelsen. Formatet er:

DIRECTORY Ddrive,Udevice,"joker-udvalg"

Ingen af parametrene er absolut nødvendige, hvis man kun anvender een 1541 disk-station (device 8).

### HEADER

Samme funktion, som NEW disk-kommandoen i forbindelse med Commodore 64. Syntax'en er:

HEADER "disk-navn",ID,Ddrive,Udevice

### RENAME

Kommandoen er identisk i virkning med den forud beskrevne RENAME-kommando, blot er syntax'en anderledes:

RENAME "gl.navn" TO "nyt navn"(Ddrive,Udevice)

### SCRATCH

Samme funktion, som SCRATCH disk-kommandoen i forbindelse med Commodore 64.

SCRATCH "filnavn",Ddrive,Udevice

## 8.10 Sekventielle filer

Sekventielle filer benyttes som betegnelse for filer, der læses og skrives som een lang sammenhængende følge af karakterer. Der findes kun een måde, at finde frem til et bestemt element i filen - nemlig ved at læse *alle* foranliggende data.

Programfiler og user filer er også organiseret sekventielt. Indholdsmæssigt adskiller programfilerne sig dog på en lille detalje i den første blok i filen. Bortset herfra, er formaterne identiske, og en programfil kan læses og skrives efter de samme retningslinjer, som almindelige sekventielle filer - i modsætning til kassettebåndoptager.

Når en fil skrives til disketten, sker det i form af blokke, hver på 256 bytes.

Dette sker automatisk, hver gang disk-stationens buffer er fuld, men for at sikre, at sidste blok i filen skrives til disketten, selvom den ikke rummer 256 bytes, er det nødvendigt at benytte CLOSE-kommandoen.

Hver blok, der skrives til filen, noteres som belagt i BAM'en , og track og sektor numrene på den første blok i filen, noteres i directory (Se Appendix C). Disse informationer er dog ikke tilstrækkelige, til at disk-stationen kan finde alle blokke i en fil. Derfor benyttes de første 2 bytes i hver blok på 256 bytes som *sektor-pointer* til næste blok i filen. Formatet ser således ud:

BYTE	SEQ-fil	PRG-fil
-----		
1.blok:		
-----		
0	track på næste blok	
1	sektor på næste blok	
2-3	data	startadresse*
4-255	data	program
2.blok:		
-----		
0	track på næste blok	
1	sektor på næste blok	
2-255	data	program
sidste blok:		
-----		
0	0 (nul)	
1	antal bytes i blokken	
2-???	data	program incl. tre nuller
-----		

De med \* mærkede bytes, er de eneste, der adskiller almindelige sekventielle filer fra programfiler - bortset fra den særlige mærkning i directory optegnelsen.

Af tabellen fremgår det tydeligt, at en sekventiel fil kun rummer 254 bytes i hver blok. De to første bytes er normalt utilgængelige for brugeren, og anvendes kun til den interne styring i disk-stationen.

#### 8.10.1 Sekventiel læsning af programfiler

Da programfiler i praksis kan opfattes som sekventielle filer, kan man anvende eksakt de samme kommandoer, som til almindelige SEQ-filer. Blot skal man være opmærksom på følgende detaljer:

- Da programmerne opbevares i *tokenized* form med linjenumre og pointere til næste linje i binært format, kan læsning ikke forgå med INPUT# - kommandoen.
- Da CHR\$(0) er en fast bestanddel af hver BASIC-programlinje, må man

huske, at benytte følgende programlinje, når data læses fra en programfil:

```
340 GET#If,A$:IF A$="" THEN A$=CHR$(0)
```

ellers går der kludder i sagerne. Dette gælder også, hvis der læses maskinkode programmer fra disk.

- Når der skrives til disketten, skal man huske at anvende *semikolon* efter alle PRINT# - kommandoer. Sker det ikke vil resultatet næppe stå mål med forventningerne, idet der indsættes en RETURN-karakter eller ekstra mellemrum (efter komma) i filen. Især kan effekten på maskinkode programmer være meget overraskende.

### 8.10.2 Chaining af programmer

En særlig fordel ved Commodore's LOAD-kommando, er at den kan anvendes til at sammenkæde programmer - både når der er tale om indlæsning fra diskette og kassette.

Benyttes denne fremgangsmåde, skal man dog være opmærksom på følgende detaljer:

- *End of BASIC* pointeren justeres ikke, når et nyt program indlæses. Fordelen er, at alle variabler overføres uændret til det indlæste program, hvis det indlæste program er mindre end det oprindelige program. Er det omvendte tilfældet, opstår der problemer, idet det nye program overskriver en del af variabel området. Derfor bør man - i egen interesse - benytte følgende fremgangsmåde i disse tilfælde:

```
100 REM PROGRAM 2
110 POKE 45,PEEK(174)
120 POKE 46,PEEK(175)
130 CLR
```

hvorefter alle pointere igen er justerede.

- Er det indlæste program mindre end det "kaldende" program, kan der dog stadig opstå problemer. Det er tekststrengene, der volder problemer i disse tilfælde. Når BASIC'en møder udtryk, som:

```
100 A$="DETTE GAAR GALT"
```

placeres værdien for A\$ ikke i strengområdet. I stedet justeres *streng-pointeren*, så den peger ned i programområdet til teksten. Indlæser man nu et nyt program, peger pointeren helt forkert, hvilket let kan efterprøves med disse to små programmer:

```
100 REM FØRSTE PROGRAM
110 A$="DENNE TEKST SKAL UDSKRIVES"
120 LOAD "ANDET PROGRAM",8
```



```
100 REM PROGRAM 2
120 PRINT A$
```

Vil man undgå dette problem, skal man blot erstatte linjer, som linje 110 i eksemplet herover, med:

```
110 A$="DENNE TEKST SKAL UDSKRIVES"+""
```

Denne fremgangsmåde *tvinger* teksten op i området for strengvariabler, og problemet er løst.

- Når en LOAD-kommando udføres i et program, starter BASIC'en automatisk forfra med første linje. Mens det er en fordel, når man *chainer* almindelige BASIC programmer, giver det mange problemer med indlæsning af maskinkodeprogrammer. F.eks. vil følgende program fortsætte med at udføre linje 100 i det uendelige:

```
100 LOAD "MASKINKODE 1",8,1
110 LOAD "MASKINKODE 2",8,1
120 LOAD "MASKINKODE 3",8,1
```

Anvender man i stedet denne fremgangsmåde:

```
100 IF A=0 THEN A=1:LOAD "MASKINKODE 1",8,1
110 IF A=1 THEN A=2:LOAD "MASKINKODE 2",8,1
120 IF A=2 THEN A=3:LOAD "MASKINKODE 3",8,1
```

er problemet løst. Eneste forudsætning er, at variablen A starter med værdien 0!

### 8.10.3 APPEND af filer

Som vist tidligere, er det forholdsvis let at APPEND'e programfiler (se side 68). Er der derimod tale om større mængder data indeholdt i en sekventiel fil, er det både en tidrøvende og pladskrævende proces, at læse *alle* data fra een fil og overføre dem til en anden, blot for at tilføje nogle få bytes til enden. I disse tilfælde, kan man benytte den særlige APPEND-kommando, som flytter *pegepinden* for næste skrivning til en fil hen til første ledige byte, efter filen, hvorefter man kan tilføje data med en almindelig PRINT# kommando. Fremgangsmåden er:

```
100 REM FREMSTILLING AF SEKVENTIEL FIL
110 OPEN 8,8,8,"TESTFIL,S,W"
120 PRINT#8,"DETTE ER EN TEST"
130 CLOSE 8
140 OPEN 8,8,8,"TESTFIL,S,A":REM APPEND
150 PRINT#8,"DETTE ER EN TEST"
160 CLOSE 8
```

## Lagring af programmer på diskette

```
170 OPEN 8,8,8,"TESTFIL,S,R":REM LÆSNING AF INDHOLDET
180 FOR N=1 TO 2
190 : INPUT#8,A$
200 : PRINT A$
210 NEXT N
220 CLOSE 8
```

Fordelen ved denne fremgangsmåde er, at vi kan udnytte al ledig plads på en diskette til sekventielle filer, medens den traditionelle fremgangsmåde med indlæsning og succesiv skrivning til en anden fil, kun tillader filstørrelser på maksimalt halvdelen af pladsen på en diskette. At der også er tale om en betydelig hastighedsgevinst, er kun et ekstra plus.

Metoden er ikke anvendelig til at APPEND'e en programfil til en anden, da to af de afsluttende nul-bytes i den første fil, og de to bytes, der repræsenterer startadressen i den anden fil, ikke fjernes. Den nye fil har ganske vist den "rigtige" længde, men computeren accepterer ikke de bytes, som følger efter de tre nuller i den første programdel, som hørende til BASIC programmet, selvom alle bytes indlæses korrekt.

### 8.10.4 Ikke lukkede filer

Filer, der ikke er lukkede korrekt, markeres med en stjerne efter filtypen, f.eks. således:

```
0 "1541TEST/DEMO " ZX 2A
13 "HOW TO USE" PRG
5 "HOW PART TWO" PRG
4 "VIC-20 WEDGE" PRG*
```

Disse filer kan ikke læses på normal måde, og V-kommandoen fjerner den pågældende fil - både fra directory og BAM. Denne situation er jo aldrig særlig heldig, hvis det er den eneste udgave af programmet, man er i besiddelse af. Men der er hjælp at hente, selvom den kommando, der skal benyttes, ikke omtales af Commodore noget sted i de medfølgende manuals. Kommandoen ser således ud:

```
100 OPEN If,"filnavn,filtype,M"
```

og muliggør læsning af filer, der ikke er blevet lukket korrekt.

Nu er problemet med de ikke-lukkede filer ikke kun et spørgsmål om at lukke dem igen. For det første, kan der mangle større eller mindre dele af den oprindelige fil, og for det andet er der normalt vrøvl med *sektor pointerne*. Dette resulterer normalt i, at filen er mere omfattende end "nødvendigt". Derfor er det nødvendigt, at undersøge indholdet manuelt efter "reparationsarbejdet".

Det følgende program indlæser *hele* indholdet i en *ikke-lukket* fil, foretager en *VALIDATE* og skriver indholdet tilbage til disk under samme navn. Første del af dette "nye" program indeholder *alle* de data, der er *skrevet* til

disketten, inden fejlen opstod. Indholdet *skal* undersøges manuelt bagefter, og den "anvendelige" del skal skrives til en ny fil. Programmet kan let modificeres til egne formål, f.eks. kan streng-array'ets størrelse normalt øges uden problemer. Bemærk, at programmet *ikke* kan anvendes til at redde *relative filer*.

```

100 DIM A$(200)
110 OPEN 8,8,8,"PROGRAM,P,M"
120 N=0:P=1
130 GET#8,A$:IF A$="" THEN A$=CHR$(0)
140 A$(N)=A$(N)+A$
150 IF ST<>0 THEN 200
160 P=(P+1)AND127
170 IF P=0 THEN N=N+1:IF N>200 THEN 200
180 GOTO 130
200 CLOSE 8

```

Diskettens directory og BAM'en genskabes, så de afspejler de faktiske forhold. Den plads, som den ikke-lukkede fil lagde beslag på, frigøres:

```

210 OPEN 15,8,15,"V0:":CLOSE 15

```

Det "reddede" program skrives tilbage til disketten igen.

```

220 OPEN 8,8,8,"00:PROGRAMNAV,N,P,W"
230 FOR X=0 TO N
240 PRINT#8,A$(X);
250 NEXT X

```

Er der tale om en programfil, skal vi sikre, at den *altid* afsluttes med 3 *nuller*. I sekventielle og user filer, kan linjerne 260 til 280 udelades.

```

260 FOR N=0 TO 255
270 PRINT CHR$(0);
280 NEXT N
290 CLOSE8
300 END

```

### 8.11 Relative filer

Relative filer adskiller sig grundlæggende fra sekventielle filer. Filtypen tillader direkte adgang til enhver del af filen, blot man kender placeringen af data. Det er således muligt at arbejde med meget store datamængder, uden at den tid, det tager at finde de enkelte dataelementer, påvirkes synderligt.

Brugere, der er vant til at arbejde med f.eks. CP/M, MS-DOS eller PC-DOS baserede computere, kan tage tingene helt afslappet. Der er ikke tale om noget helt nyt og ukendt for dem. Filtypen er identisk med betegnelsen "random file" på disse maskiner. Commodore anvender betegnelsen *random*

*files* om alle disk-operationer, som foretager direkte lagring eller læsning af informationer i de enkelte sektorblokke på en diskette. Dette betegnes oftest som *direct disk access* eller *sector adressing* på andre computere. Pas på, at du ikke kommer til at rode rundt i begreberne.

Før en relativ fil kan anvendes, skal der træffes en række beslutninger. En relativ fil er opbygget af blokke med en fast længde - kaldet *poster* eller *records*. Mens det ikke volder noget problem at skrive færre data til en sådan post, kan længden aldrig overskrides.

Længden af hver enkelt post fastlægges første gang, filen åbnes, og kan ikke forandres senere. Den maksimale længde for hver post er 255 bytes, men iverdigt kan længden fastlægges frit, ligesom det er muligt at benytte et valgfrit antal poster (op til 32737). Den maksimale størrelse af en relativ fil bestemmes af den ledige plads på disketten, minus den plads disk-stationen reserverer til styring af filen - 6 blokke pr. relativ fil.

Når en relativ fil åbnes (uanset om det er første gang eller ej), sker det med kommandoen:

```
OPEN If,device,kanal,"navn,L,("+CHR$(længde))
```

Der skelnes ikke mellem skrivning eller læsning, begge dele kan foretages frit. Selvom det ikke er strengt nødvendigt at anvende *længde* parameteren, når den relative fil har været åbnet een gang, kan det være en fordel at anvende den "komplette" form hver gang, idet en evt. fejlfinding lettes betydeligt!

Inden man kan skrive til, eller læse fra en relativ fil, skal man have positioneret *fil-pointeren* - dvs. at man skal meddele disk-stationen, hvilken *post* man ønsker adgang til, og fra hvilket byte i posten, man ønsker at læse hhv. skrive. Positioneringen sker med kommandoen:

```
PRINT#If,"P";CHR$(kanal);CHR$(lo);CHR$(hi);CHR$(byte)
```

hvor *If* er det logiske filnummer på kommando kanalen, *kanal* er den kanal, der er åbnet til den relative fil, *lo* og *hi* beskriver postnummeret ( $=256*hi+lo$ ). Parameteren *byte* beskriver det byte (1 til *længde*), som *fil-pointeren* skal pege mod. Den sidste parameter kan udelades, men det foregående semikolon skal altid medtages - ellers opfatter disk-stationen den efterfølgende RETURN-karakter, som en del af kommandoen, og placerer *fil-pointeren* på byte nr. 13!!! Den komplette procedure for skrivning til en relativ fil kan se således ud (GET# og INPUT# kommandoerne anvendes analogt hertil):

```
100 OPEN 1,8,15:REM KOMMANDOKANAL SKAL ALTID AABNES
110 OPEN 2,8,3,"FILNAVN,L,("+CHR$(32)
120 A$="1234567890123456789012345678901"
130 RN=257:REM RECORD NUMMER
140 HI=INT(RN/256)
150 LO=RN-256*HI
160 PRINT#1,"P";CHR$(3);CHR$(LO);CHR$(HI);CHR$(1)
170 PRINT#2,A$
180 CLOSE 8
190 CLOSE 15
```

Bemærk, at der også skal afsættes plads til *RETURN* karakteren, hvis vi ønsker at foretage tilbagelæsning af postens indhold med en *INPUT#* kommando.

Når vi skriver til en *record* eller *post* for første gang, oprettes denne samt alle *mellemliggende* poster - dvs. at disk-stationen noterer udnyttelsen i sin specielle arbejdsfil - kaldet *side sectors* - og udfylder alle ikke tidligere benyttede poster med *CHR\$(255)*. Tager vi udgangspunkt i det viste eksempel, og antager vi, at det højeste postnummer, der tidligere har været benyttet, var nummer 26, så vil *PRINT#* kommandoen i linje 160 medføre, at alle posterne fra 27 til og med 256 oprettes automatisk, når der skrives til post 257! Posterne 1 til og med 26 påvirkes *ikke*.

En relativ fil vokser i takt med den tilgang af poster, der sker. Denne egenskab udnyttes dog kun sjældent, idet det kræver unødigt megen tid, at oprette en ny post (og evt. *mellemliggende* poster), hver gang, der skrives til en relativ fil. I stedet vælger man normalt, at *dimensionere* filen til den forventede størrelse fra starten. Senere spares denne tid igen, når man opretter en ny post.

Lad os se på et eksempel, som også kan udnyttes i praksis.

#### 8.11.1 Database baseret på relative filer

I det følgende gennemgås et færdigt database-program. Programmet kan anvendes til at finde adresser og telefonnumre på enkeltpersoner eller firmaer, og kan tilpasses individuelle behov uden større problemer.

Inden programmet fremstilles, skal informationernes format og omfang fastlægges. Programmet er fastlagt til at rumme 6 felter med følgende indhold i hver post:

FELT	LÆNGDE	INDHOLD
1	10 tegn	Fornavn
2	29 tegn	Efternavn/Firmanavn
3	29 tegn	Adresse
4	4 tegn	Postnummer
5	24 tegn	Bynavn og evt. land
6	12 tegn	Telefon nummer
108 tegn		ialt

I dette eksempel skal data læses med *INPUT#* kommandoen, og vi er derfor nødt til at reservere 6 bytes mere til *RETURN* karakterer - alt 114 karakterer!

I det følgende program specificeres denne værdi i variablen *FELT*. Programmet opretter ialt 10 poster, men der er *intet* i vejen for at oprette 100, 300 eller 1000 poster.

## Lagring af programmer på diskette

```
100 REM
110 REM OPRET DATABASE
120 REM
130 ANTAL=10:REM SAMME SOM DATABASE
140 FELT=114:REM TOTAL INCL. RETURN'S
150 OPEN 8,8,8,"00:EMPTY,S,W"
160 PRINT#8,ANTAL
170 FOR N=1 TO ANTAL
180 : PRINT#8,N
190 NEXT N
195 CLOSE 8
```

Af praktiske årsager oprettes en sekventiel fil, som indeholder numrene på de poster, der ikke er udnyttede. Den første værdi i denne fil er antallet af ledige poster - i dette tilfælde 10. Udover denne "husholdningsfil" oprettes en speciel fil - en *keyword* eller *nøgle* fil, ligeledes sekventiel - som skal indeholde *indholdsfortegnelsen*. Første værdi fortæller hvor mange "aktive" poster, der eksisterer - udgangspunktet er 0 (nul).

Denne fil tjener et vigtigt formål senere. I stedet for at gennemlæse alle enkelt-poster i databasen - meget tidskrævende, hvis vi har 1000 poster - foretager vi i stedet et opslag i *indholdsfortegnelsen*, som består af de første 4 tegn i efter/firmanavnet fulgt af postnummeret. Har vi en fil, der rummer 1000 poster, er det helt umuligt at rumme alle posterne i computeren, og en gennemlæsning - post for post - tager år og dag. I stedet kan vi slå op i *indholdsfortegnelsen*, og nøjes med at undersøge de poster, der starter med de samme 4 tegn - f.eks. HANS for HANSEN og PETE for PETERSEN.

Fordelen ved denne fremgangsmåde er, at kun *indholdsfortegnelsen* skal være i alfabetisk rækkefølge - placeringen på disketten er fuldstændig underordnet, så vi slipper for en yderst langsommelig sortering af diskettens indhold. Sorteringen (meget simpel) kan foregå i computerens hukommelse i stedet, og computeren har rigelig plads til *indholdsfortegnelsen* - hver optegnelse kræver maksimalt 8 bytes (plus de bytes, der anvendes til hvert element i et stringarray).

```
200 OPEN 8,8,8,"00:KEYWORDS,S,W"
210 PRINT#8,0
220 CLOSE 8
```

Programmet opretter kun 10 poster, men antallet styres af variablen ANTAL, og kan ændres efter behov. Det lave tal er udelukkende valgt, så du ikke skal vente for længe - f.eks. på grund af manglende plads på dine disketter - inden programmet kan afprøves.

```
230 OPEN 15,8,15
240 OPEN 8,8,8,"TLF-ADR,L,"+CHR$(FELT)
250 HI=INT(ANTAL/256)
260 LO=ANTAL-256*HI
270 PRINT#15,"P";CHR$(8);CHR$(LO);CHR$(HI);CHR$(0):REM OBS
280 PRINT#8,"TOM"
290 CLOSE 8
300 CLOSE 15
310 END
```

Ved at skrive teksten "TOM" i den 10. post, sikrer vi at *alle* poster oprettes fra starten. Bemærk, at første gang, der skrives til en post, reagerer disk-stationen med fejlmedlingen 50 RECORD NOT PRESENT 00 00. Det vil også ske i foregående program, men fejlen er uden betydning i praksis. Prøv at køre programmet en gang til, og du vil se, at fejlmedlingen ikke opstår igen.

Op nu til selve database programmet. Variablerne ANTAL og FELT *skal* være de samme, som i oprettelsesproceduren. Fremgangsmåden med adskilte programmer er valgt, så man ikke ved en fejltagelse kan komme til at slette selve databasen!

```

100 REM
110 REM DATA-BASE
120 REM
130 ANTAL=10:FELT=114:.SY=0
140 DIM KEY$(ANTAL),PO(ANTAL)
150 DIM LX(5),KY(5),IND$(5),TXT$(5),DM$(5)
160 PRINT CHR$(147);"DATA-BASE":PRINT"-----"
170 GOTO 280

```

Variablerne dimensioneres. De følgende linjer indeholder indlæsningsrutiner og cursorstyring. Fremgangsmåden i linje 190 er valgt fremfor metoden i linje 240, fordi den sikrer "dynamisk" indtastning - dvs. at der ikke skal trykkes på en knap, for at læse karakterer fra *tastatur bufferen* (læs mere om denne i kapitlet TASTATUR og SKÆRM).

```

180 POKE 204,0:REM CURSOR ON
190 GET I$:IF I$="" THEN 190
200 IF PEEK(207) THEN 200
210 POKE 204,1:REM CURSOR OFF
220 RETURN
230 REM GET KARAKTER
240 WAIT 203,63:GET I$:RETURN
250 REM SET CURSOR
260 POKE 214,L:POKE 211,K:SYS58732:RETURN

```

Felternes betegnelser og placeringen af første tegn i indtastningen fastlægges:

```

270 REM SET VARIABEL-INDHOLD
280 TXT$(0)="FORNAVN...":LX(0)=5:KY(0)=10
290 TXT$(1)="EFTERNAVN...":LX(1)=6:KY(1)=10
300 TXT$(2)="ADRESSE...":LX(2)=7:KY(2)=10
310 TXT$(3)="POSTNR...":LX(3)=8:KY(3)=7
320 TXT$(4)="BY...":LX(4)=8:KY(4)=15
330 TXT$(5)="TLF...":LX(5)=9:KY(5)=4

```

Indtastningsformatet fastlægges - rutinen anvender den tidligere præsenterede metode, hvor indholdet på skærmen læses direkte fra skærmen. De følgende variabler fastlægger indholdet i *indtastningsfelterne*. Punktummerne erstattes senere med mellemrum (linje 1150). Stjernerne anvendes for at fremtvinge en indtastning af fire cifre i postnummeret.

## Lagring af programmer på diskette

```
340 DM$(0)="....."
350 DM$(1)="....."
360 DM$(2)="....."
370 DM$(3)="****"
380 DM$(4)="....."
390 DM$(5)="....."
```

Indlæsning af de egentlige styringsvariabler foretages. PO(0) kommer til at indeholde antallet af ledige poster. KEYS kommer til at indeholde antallet af "aktive" poster.

```
400 OPEN 8,8,8,"EMPTY,S,R"
410 INPUT#8,PO(0):IF PO(0)=0 THEN 450
420 FOR N=1 TO PO(0)
430 : INPUT#8,PO(N)
440 NEXT N
450 CLOSE 8
460 OPEN 8,8,8,"KEYWORDS,S,R"
470 INPUT#8,KEYS:IF KEYS=0 THEN 510
480 FOR N=1 TO KEYS
490 : INPUT#8,KEY$(N)
500 NEXT N
510 CLOSE 8
```

Abningsmenuen skrives på skærmen. Normalt findes kun 3 muligheder, nemlig OPRETtelse af en ny post, SØGning efter en eksisterende post, eller EXIT, som skal benyttes, når programmet skal forlades. Dette er meget vigtigt, idet tilføjelser og sletninger først registreres på disketten, når EXIT-rutinen udføres.

SLETning eller RETTelser kan kun udføres på filer, der er fundet ved hjælp af SØG rutinen. Postens indhold befinder sig i så fald på skærmen!

```
520 PRINT CHR$(147)
530 L=12:K=15:GOSUB 260:PRINT"F1 = OPRET"
540 L=13:K=15:GOSUB 260:PRINT"F3 = RET  "
550 L=14:K=15:GOSUB 260:PRINT"F5 = SØG  "
560 L=15:K=15:GOSUB 260:PRINT"F7 = SLET  "
570 L=16:K=15:GOSUB 260:PRINT"F2 = EXIT  "
580 L=18:K=12:GOSUB 260:PRINT"INDTAST DIT VALG"
590 GOSUB 240:IF I$<CHR$(132)THEN 590
600 ON ASC(I$)-132 GOSUB 660,750,1290,1670,1810
610 IF SY=0 THEN 520
620 GOTO 530
```

Oprettelse af en ny fil udnytter i store træk samme rutiner, som rettelse. Forskellene i programforløbet afgøres af variablen SY, som vil være nul ved oprettelse, og indeholde nummeret på det element i indholdsfortegnelsen KEY\$(SY), som indeholder nummeret på den post, der skal rettes.



```

630 REM
640 REM OPRET
650 REM
660 IF PO(0)=0 THEN RETURN
670 FOR N=0 TO 5
680 : IND$(N)=DM$(N)
690 NEXT N
700 SY=0
710 GOTO 760
720 REM
730 REM RET
740 REM
750 IF SY=0 THEN RETURN
760 GOSUB 1940:REM PRINT

```

Indtastningsrutinen. Cursor op og cursor ned, flytter cursoren til starten af foregående eller efterfølgende felt. Flytningen godkendes kun, hvis feltet indeholder korrekte indtastninger! Iøvrigt virker cursor til højre hhv. venstre, som normalt, og RETURN-knappen afslutter indtastningen, og indleder lagringen af data på disketten - forudsat at indtastningerne opfylder betingelserne, der er fastlagt og kontrolleres i linjerne 1090 til 1250.

```

770 N=0
780 CU=0
790 LW=LEN(IND$(N))-1
800 L=LX(N):K=KY(N)+CU:GOSUB 260:GOSUB 180
810 IF (ASC(I$)AND127)>31 THEN 880
820 IF I$=CHR$(29) THEN 890
830 IF I$=CHR$(157) THEN IF CU>0 THEN CU=CU-1
840 IF I$=CHR$(145) THEN GOSUB 1090:IF X<90 AND N>0 THEN N=N-1:GOTO 780
850 IF I$=CHR$(17) THEN GOSUB 1090:IF X<90 AND N<5 THEN N=N+1:GOTO 780
860 IF I$=CHR$(13) THEN 940
870 GOTO 800
880 PRINT I$;
890 IF CU<LW THEN CU=CU+1
900 GOTO 800

```

Det undersøges om indtastningen er korrekt.

```

910 REM
920 REM INDTASTNING SLUT ?
930 REM
940 FOR N=0 TO 5
950 : GOSUB 1090
960 : IF X>90 THEN P=N:N=5
970 NEXT N
980 IF X>90 THEN N=P:GOTO 790

```

Er der tale om en rettelse opdateres postens indhold, og der springes tilbage til hoved-menuen. Ellers søges først den korrekte placering i indholdsfortegnelsen (GOSUB 2090), hvorefter denne opdateres og data skrives til en ledig post (GOSUB 2430).

## Lagring af programmer på diskette

```
990 IF SY>0 THEN GOSUB 2520:SY=0:RETURN
1000 SØG$=LEFT$(IND$(1),4):GOSUB 2090:REM FIND PLADS
1010 IF SØG$>LEFT$(KEY$(SY),4) THEN SY=SY-1
1020 IF SY<1 THEN SY=1
1030 GOSUB 2430:REM SKRIV RECORD
1040 SY=0
1050 RETURN
1060 REM
1070 REM KONTROL AF INDFASTNINGEN
1080 REM
1090 IF N=0 OR N=1 OR N=4 THEN LL$="A":LU$="Z"
1100 IF N=3 OR N=5 THEN LL$="Ø":LU$="9"
1110 R=LEN(IND$(N)):IND$(N)=""
1120 L=LX(N):K=KY(N):GOSUB 260
1130 OPEN 1,3
1140 FOR Q=1 TO R
1150 : GET#1,A$:IF A$="." THEN A$=" "
1160 : IND$(N)=IND$(N)+A$
1170 NEXT Q
1180 CLOSE1:CU=0
1190 FOR X=1 TO LEN(IND$)
1200 : X$=MID$(IND$(N),X,1)
1210 : IF N=2 AND X$>="Ø" AND X$<"9" THEN 1240
1220 : IF X$=LL$ AND X$<LU$ THEN 1240
1230 : CU=X-1:X=99
1240 NEXT X
1250 RETURN
```

Rutinen indlæser efternavnet på den søgte person. De første fire tegn benyttes til opslag i indholdsfortegnelsen (KEY\$).

```
1260 REM
1270 REM SØG EFTER MATCHING ENTRY
1280 REM
1290 N=1
1300 PRINT CHR$(147);"INDTAST DET SØGTE NAVN:"
1310 L=LX(N):K=0:GOSUB 260:PRINT TXT$(N);
1320 IND$(N)=DM$(N)
1330 L=LX(N):K=KY(N):GOSUB 260:PRINT IND$(N)
1340 CU=0:LW=LEN(IND$(N))-1
1350 L=LX(N):K=KY(N)+CU:GOSUB 260:GOSUB 180
1360 IF (ASC(I$)AND127)>31 THEN 1410
1370 IF I$=CHR$(29) THEN 1420
1380 IF I$=CHR$(157) THEN IF CU>0 THEN CU=CU-1
1390 IF I$=CHR$(13) THEN GOSUB 1090:IF X<90 THEN 1470
1400 GOTO 1350
1410 PRINT I$;
1420 IF CU<LW THEN CU=CU+1
1430 GOTO 1350
```

Indholdsfortegnelsen gennemses og samstemmende værdier indlæses fra diskette, hvorefter det afgøres, om det søgte navn, også findes på disketten i virkeligheden. Mulige emner er indeholdt i KEY\$ fra nr. SÝ til nr. SZ. Evt. overensstemmelser med det søgte navn præsenteres på skærmen (f.eks. kan flere hedde HANSEN), hvorefter brugeren må afgøre, om der er tale om den

søgte værdi. Svares ja, forlades rutinen, og der hoppes tilbage til hoved-menuen. Postens indhold forbliver på skærmen, og SLET eller RET mulighederne kan udnyttes. Svares nej, søges videre, indtil der ikke findes flere muligheder, hvorefter der vendes tilbage til hovedmenuen med tom skærm!

```

1440 REM
1450 REM GYLDIG INDFASTNING
1460 REM
1470 SØG$=LEFT$(IND$(1),4)
1480 GOSUB 2090
1490 IF OK=0 THEN L=23:K=0:GOSUB 260:PRINT "FINDES IKKE":SY=0:RETURN
1500 Q$=IND$(1):OK=0
1510 FOR SY=SY TO SZ
1520 : GOSUB 2300:REM INDLÆS RECORD
1530 : IF IND$(1)<>Q$ THEN 1600
1540 : GOSUB 1940:REM PRINT MATCH
1550 : L=23:K=0:GOSUB 260:PRINT "ER DETTE DEN ØNSKEDE POST?"
1560 : GOSUB 240
1570 : IF I$="N" THEN 1600
1580 : IF I$<>"J" THEN 1560
1590 : Q=SY:SY=9999:OK=1
1600 NEXT SY
1610 IF SY<9999 THEN OK=0:SY=0:RETURN
1620 SY=Q
1630 RETURN

```

En post slettes. I praksis slettes posten kun fra indholdsfortegnelsen, og postnummeret overføres til tabellen over ledige filer.

```

1640 REM
1650 REM SLETNING
1660 REM
1670 IF SY=0 THEN RETURN
1680 RN=VAL(RIGHT$(KEY$(SY),LEN(KEY$(SY))-4))
1690 IF SY=KEYS THEN 1730
1700 FOR N=SY+1 TO KEYS
1710 : KEY$(N-1)=KEY$(N)
1720 NEXT N
1730 KEYS=KEYS-1
1740 PO(0)=PO(0)+1
1750 PO(PO(0))=RN
1760 SY=0
1770 RETURN

```

Programmet forlades, og indholdsfortegnelse samt tabellen over ledige poster skrives til disketten. Først nu er alle ændringer registrerede til senere brug.

```

1780 REM
1790 REM EXIT
1800 REM
1810 OPEN 8,8,8,"00:EMPTY,S,W"
1820 FOR N=0 TO PO(0)
1830 : PRINT#8,PO(N)
1840 NEXT N
1850 CLOSE 8

```

## Lagring af programmer på diskette

```
1860 OPEN 8,8,8,"00:KEYWORDS,S,W"
1870 PRINT#8,KEYS
1880 FOR N=1 TO KEYS
1890 : PRINT#8,KEY$(N)
1900 NEXT N
1910 CLOSE 8
1920 PRINT CHR$(147);"SLUT"
1930 END
```

Generel rutine til skærmudskrifter.

```
1940 L=4:K=0:GOSUB 260
1950 PRINT"-----";
1960 FOR N=0 TO 5
1970 : K=0:IF N=4 THEN K=11
1980 : L=LX(N):GOSUB 260
1990 : PRINT TXT$(N);
2000 : L=LX(N):K=KY(N):GOSUB 260
2010 : PRINT IND$(N);
2020 NEXT N
2030 L=10:K=0:GOSUB 260
2040 PRINT"-----";
2050 RETURN
```

Rutinen søger den korrekte placering i indholdsfortegnelsen for en tilføjelse. Variablen SY indeholder nummeret på første element, der er større eller lig med den søgte værdi. Ved søgning indeholder SZ nummeret på det største element i indholdsfortegnelsen, som er lig med variablen SØG\$, der indeholder de første fire tegn på det indtastede efternavn. Den anvendte søgerutine er en modificeret binær søgning.

```
2060 REM
2070 REM FIND MATCHING ENTRY
2080 REM
2090 SX=1:SZ=KEYS:OK=0
2100 SY=INT((SX+SZ)/2)
2110 IF (SY-SX)<1 AND (SZ-SY)<1 THEN 2250:REM CHECK FOR 1 MATCH
2120 IF SØG$<LEFT$(KEY$(SY),4) THEN SZ=SY-1:GOTO 2100
2130 IF SØG$>LEFT$(KEY$(SY),4) THEN SX=SY+1:GOTO 2100
2140 REM
2150 REM CHECK FOR FLERE ENTRIES
2160 REM
2170 OK=1
2180 SY=SY-1
2190 IF SØG$=LEFT$(KEY$(SY),4) THEN 2180
2200 SY=SY+1:SZ=SY
2210 SZ=SZ+1
2220 IF SØG$=LEFT$(KEY$(SZ),4) THEN 2210
2230 SZ=SZ-1
2240 RETURN
2250 IF SØG$=LEFT$(KEY$(SY),4) THEN OK=1:SZ=SY
2260 RETURN
```

Indlæsning af en post, udvalgt på basis af det postnummer, der findes i variablen KEY\$(SY) - de første 4 tegn er selve opslagsværdien (4 første tegn

i efternavn).

```

2270 REM
2280 REM INDLÆS RECORD - 4 FØRSTE TEGN=NØGLE
2290 REM
2300 RN=VAL(RIGHT$(KEY$(SY),LEN(KEY$(SY))-4))
2310 OPEN 15,8,15
2320 OPEN 8,8,8,"TLF-ADR,L,"+CHR$(FELT)
2330 GOSUB 2670:REM POSITIONER PAA RN
2340 FOR N=0 TO 5
2350 : INPUT#8,IND$(N)
2360 NEXT N
2370 CLOSE 8
2380 CLOSE 15
2390 RETURN

```

En post tilføjes til databasen.

```

2400 REM
2410 REM ADD A NEW RECORD
2420 REM
2430 RN=PO(PO(0)):PO(0)=PO(0)-1
2440 FOR N=KEYS TO SY STEP-1
2450 : KEY$(N+1)=KEY$(N)
2460 NEXT N
2470 KEYS=KEYS+1
2480 KEY$(SY)=SØG$+STR$(RN)
2490 REM
2500 REM SKRIV EN RECORD
2510 REM
2520 RN=VAL(RIGHT$(KEY$(SY),LEN(KEY$(SY))-4))
2530 OPEN 15,8,15,"IØ"
2540 OPEN 8,8,8,"TLF-ADR,L,"+CHR$(FELT)
2550 GOSUB 2670:REM POSITIONER PAA RN

```

Bemærk den følgende fremgangsmåde. Benyttes fremgangsmåden:

```

2570 FOR N=0 TO 5
2580 : PRINT#8,IND$(N)
2590 NEXT N

```

i stedet, vil indholdet i posten ikke være korrekt! Det er nødvendigt at benytte positioneringskommandoen (se linje 2590) forud for hver skrivning til disketten, ellers "driller" disk-stationen, og der opstår rod i data.

```

2560 P$=""
2570 FOR N=0 TO 5
2580 : P$=P$+IND$(N)+CHR$(13)
2590 NEXT N
2600 PRINT#8,P$;
2610 CLOSE 8
2620 CLOSE 15
2630 RETURN

```

## Lagring af programmer på diskette

Generel rutine til positionering på en ønsket post - både ved indlæsning og skrivning.

```
2640 REM
2650 REM POSITIONER PÅ RECORD RN
2660 REM
2670 HI=INT(RN/256)
2680 LO=RN-256*HI
2690 PRINT#15,"P";CHR$(8);CHR$(LO);CHR$(HI);CHR$(1):REM OBS!!!
2700 RETURN
```

### 8.12 Random filer

Til langt de fleste formål vil de forud beskrevne kommandoer opfylde alle de krav, man med rimelighed kan stille. I enkelte situationer kan der dog opstå behov for mere specialiserede disk-funktioner - f.eks. når man skal redde beskadigede filer, når man benytter helt specielle fil-strukturer etc.

Til disse formål råder man over en serie kommandoer, som tillader direkte arbejde på blok-niveau (B- eller U- kommandoer), eller programmering af tillægsfunktioner m.m. direkte i disk-stationen (M- og U-kommandoer).

Funktionerne giver maksimal fleksibilitet og frihed i omgangen med disketter, men rummer samtidig alle ønskelige muligheder for alvorlige følgevirkninger, hvis man ikke tænker sig grundigt om, inden anvendelsen.

Har du lyst til at udforske kommandoernes rige muligheder, bør du benytte en diskette, som ikke indeholder kritiske data eller programmer af værdi.

#### 8.12.1 Blok kommandoer

Disk-stationen er udstyret med ialt 5 buffere, der hver kan rumme 256 bytes. Normalt styrer disk-stationen selv, hvilke buffere der benyttes, men stiller anvendelsen specielle krav, er det muligt at specificere bufferen, der skal bruges - f.eks. fordi et maskinkodeprogram, skal placeres og udføres i bufferen.

Når en *random file* åbnes, kan det ske på to måder - f.eks.:

```
OPEN 7,8,9,"#" eller OPEN 7,8,9,"#2"
```

I første tilfælde bestemmer diskstationen selv, hvilken buffer, der benyttes, mens andet eksempel fastlægger den benyttede buffer til nr.2 (tilladte buffernumre er 0 til 5). Udvælges en buffer, som allerede er i anvendelse, fås en fejlmeddelelse.

Ligesom med relative filer, kræver også random filer, at kommando kanalen er åben, idet al styring af *buffer-pointeren* sker via denne.

#### 8.12.1.1 Buffer pointer

Buffer-pointeren anvendes til at bestemme, hvor i bufferen lagring eller læsning af data skal foregå. Pointeren kan positioneres frit på alle 256 bytes i en buffer, og placeringen bestemmes med kommandoen:

```
PRINT#If,"B-P";kanal;position
```

Hvor *If* er det logiske filnummer for kommando kanalen, *kanal* er kanal nummeret på den benyttede *random fil*, og *<position>* er en værdi mellem 0 og 255.

#### 8.12.1.2 Block read kommandoer

Kommandoerne benyttes til at indlæse indholdet fra en blok på disketten til en buffer i diskstationen. Der findes to block read kommandoer:

```
PRINT#If,"B-R";kanal,drive,track,sektor
```

eller:

```
PRINT#If,"U1";kanal,drive,track,sektor
```

Førstnævnte kommando indlæser *ikke* det første byte i en blok - dette byte opfattes af disk-stationen, som en pointer for antallet af de bytes, der er skrevet til sektoren. Derfor anvendes normalt den anden udgave (U1), som indlæser alle 256 bytes uden ændringer. F.eks. vil de følgende linjer indlæse første blok i directory:

```
100 OPEN 1,8,15,"I0"
110 OPEN 7,8,9,"#"
120 PRINT#1,"U1",9,0,18,0
130 PRINT#1,"B-P",9,144
140 PRINT#7,"NYT DISK NAVN !!";
```

Buffer pointeren flyttes til det sted, hvor diskettens navn befinder sig, og indholdet overskrives med diskettens nye navn.

#### 8.12.1.3 Block write kommandoen

Når indholdet skal skrives tilbage til disketten igen, må vi benytte en *block write* kommando. Ligesom for *block read* findes også denne i to udgaver, nemlig:

`PRINT#lf,"B-W";kanal,drive,track,sektor`

eller:

`PRINT#lf,"U2";kanal,drive,track,sektor`

sidstnævnte udgave skriver bufferens indhold uforandret tilbage til disketten, mens første udgave skriver *buffer pointerens* værdi efterfulgt af 255 bytes til disketten. Normalt anvendes derfor U2-kommandoen, f.eks. som her:

```
150 PRINT#1,"U2",9,0,18,0
160 CLOSE 7
170 CLOSE 1
```

hvor bufferens indhold, skrives tilbage til disketten i sektor 0 på spor (track) 18. I praksis har vi nu udstyret disketten med et nyt navn, uden at reformatting har været nødvendig.

#### 8.12.1.4 Block allocate kommando

Block Allocate kommandoen anvendes til at markere den specificerede blok i diskettens BAM, som optaget. Blokken vil herefter være beskyttet mod overskrivning. Kommandoen udføres således:

`PRINT#lf,"B-A";drive,track,sektor`

Er den blok, man ønsker at reservere, allerede optaget, fås fejlmeldingen:

```
65 NO BLOCK TT SS
```

hvor TT og SS henviser til næstfølgende frie blok på disketten.

OBS: Vær opmærksom på, at VALIDATE kommandoen sletter *alle* optegnelser, som ikke er opført i diskettens directory, dvs. at block allocate kommandoens virkning ophæves!

#### 8.12.1.5 Block free kommando

Den modsatte kommando af B-A hedder *block free*, og frigiver en blok, der tidligere har været beskyttet mod overskrivning, til almindelig brug. Kommandoen udføres således:

`PRINT#lf,"B-F";drive,track,sektor`



hvorefter den pågældende blok er markeret som fri, i diskettens BAM.

#### 8.12.1.6 Block execute kommando

Kommandoen anvendes til indlæsning af en blok i disk-stationens hukommelse med henblik på at udføre maskinkode programmet i disk-stationens buffer. Kommandoen har følgende syntax:

```
PRINT#lf,"B-E";kanal,drive,track,sektor
```

og anvendelsen kan i praksis se således ud:

```
l00 OPEN 1,8,15
110 OPEN 7,8,9,"#3"
120 PRINT#1,"B-E";9,0,21,12
```

som indlæser en blok fra track 21, sektor 12 til buffer nr. 3, og udfører programmet i bufferen. Disk-stationens fem buffere har følgende placering:

```
Buffer 0 - 0300-03FF
Buffer 1 - 0400-04FF
Buffer 2 - 0500-05FF
Buffer 3 - 0600-06FF
Buffer 4 - 0700-07FF
```

En særlig udgave af U-kommandoerne muliggør en mere målrettet kontrol og styring af maskinkoden, der befinder sig i en af disk-stationens buffere. Kommandoerne anvendes således:

```
PRINT#lf,"Ux:parametre"
```

hvor x står for en værdi mellem 1 og 9, eller A og J. U1 og U2 kommandoerne har vi allerede stiftet bekendtskab med. UJ eller U kommandoen fremtvinger en reset i disk-stationen, svarende til tænd/sluk processen. De øvrige U-kommandoer hopper til en nærmere fastlagt adresse, der er specificeret her:

```
UC - U3 - 0500 hex
UD - U4 - 0503 hex
UE - U5 - 0506 hex
UF - U6 - 0509 hex
UG - U7 - 050C hex
UH - U8 - 050F hex
UI - U9 - via indholdet i 0065/0066
```

Ved at placere kommandoen:

```
JMP adresse
```

på de pågældende pladser i buffer 2, kan man springe til en udvalgt del af en indlæst rutine. De parametre, der overføres sammen med U-kommandoen kan vælges frit, og således udnyttes til at udløse særlige virkninger.

UI eller U9 kommandoen springer til adresse FF01, hvor det afgøres om det efterfølgende parameterområde indledes med + eller -. Er dette tilfældet lagres værdien i adresse 23. Er det ikke tilfældet springes indirekte til adressen lagret i adresserne 0065/0066 (en JMP (\$0065) kommando).

### 8.12.2 Memory kommandoer

Memory kommandoerne er specielt beregnet til læsning eller skrivning af enkeltbytes i computerens hukommelse. Anvendelse af disse kommandoer kræver ikke, at andet end kommando kanalen er åben. Kommandoerne anvendes således:

```
100 OPEN 1,8,15
110 PRINT#1,"M-x";+parametre
120 CLOSE 1
```

Kommandoerne forekommer i tre udgaver:

#### M-E

Memory execute - udfører en maskinkode rutine, der befinder sig på adressen  $lo+256*hi$  i det følgende udtryk:

```
PRINT#1,"M-E";CHR$(lo);CHR$(hi)
```

#### M-W

Memory write - skriver et nærmere specificeret antal bytes (maksimalt 34) til en adressen  $lo+256*hi$ , efter følgende regler:

```
PRINT#1,"M-W";CHR$(lo);CHR$(hi);CHR$(antal);CHR$(data)..CHR$(data)
```

#### M-R

Memory read - læser et nærmere specificeret antal bytes fra disk-stationens hukommelse. Kommandoen har følgende syntax:

```
PRINT#1,"M-W";CHR$(lo);CHR$(hi);CHR$(antal)
```

F.eks. som i følgende eksempel, der indlæser diskettenavnet fra buffer 4 (normalt læses BAM'en ind i denne buffer):

```
100 OPEN 1,8,15,"I0"
110 PRINT#1,"M-R";CHR$(144);CHR$(7);CHR$(16)
120 INPUT#1,A$
130 PRINTA$
```

## 8.12.3 Slettebeskyttelse af programmer

I det følgende gives et lille praktisk og meget nyttigt eksempel på anvendelsen af et par af block kommandoerne.

Normalt kan man kun beskytte sine filer mod sletning ved at klæbe et *WRITE-PROTECT* mærke over udkæringen i siden af disketten. En udokumenteret detalje ved disk-stationens directory format (se appendix Disk format på Commodore 1541) for hver fil-optegnelse, gør det faktisk muligt at slette-beskytte filer individuelt! Denne funktion styres af, om bit 6 i 1 byte i en fil-optegnelse er sat eller ej. Der findes ingen kommando til at sætte dette bit, og dermed beskytte en fil mod utilsigtet sletning (filen mærkes med et efterfølgende "<" i directory-udskrifterne)!

Problemet er dog ikke større, end at vi selv kan fremstille et program, der foretager den ønskede ændring - ved samme lejlighed åbnes der også mulighed for at *un-erase* tidligere slettede filer - forudsat der ikke er skrevet til disketten i mellemtiden.

```

1 DIM TY$(4)
2 TY$(1)="SEQ"
3 TY$(2)="PRG"
4 TY$(3)="USR"
5 TY$(4)="REL"
6 REM
7 REM LOCK IN UPPERCASE MODE
8 REM
9 PRINT CHR$(142);CHR$(8)
10 DIM BL$(255) :REM READ-DATA
11 DIM W$(255) :REM WORK-DATA
12 TR=18:SE=1 :REM FØRSTE SEKTOR
13 OP$="SPULWNE":REM OPTIONER
15 CL$=CHR$(147):REM CLEAR SCREEN
16 REM
17 REM READ (NEXT) BLOCK
18 REM
19 GOSUB 124
20 REM
21 REM FREMSTIL ARBEJDSKOPI
22 REM
23 FOR N=0 TO 255
24 : W$(N)=BL$(N)
25 NEXT N
27 PRINT CL$;" NR LNGD FILNAVN TYPE"
28 PRINT " -----"
29 FOR FL=0 TO 7
30 : FB=2+32*FL
31 : PRINT " ";FL;
32 : PRINT W$(FB+28)+256*W$(FB+29),
33 : FOR N=FB+3 TO FB+18
34 : PRINT CHR$(W$(N));
35 : NEXT N

```

```

36 : PRINT " ";
37 : IF W%(FB) AND 128 THEN PRINT " ";;GOTO 40
38 : IF W%(FB)=0 THEN PRINT"--DEL":GOTO 45
39 : PRINT "*";
40 : IF W%(FB) AND 64 THEN PRINT ">";:GOTO 42
41 : PRINT " ";
42 : X=W%(FB)AND7
43 : IF X>4 THEN PRINT"ERROR!":GOTO 45
44 : PRINT TY$(X)
45 NEXT FL
46 PRINT:PRINTTAB(9);"ENTER N FOR NEXT BLOCK"
47 PRINTTAB(15);"E FOR EXIT"
48 PRINTTAB(12);"OR FILENUMBER"
49 GETA$:IF A$="" THEN 49
50 IF A$="N" THEN GOSUB 102:GOTO 19
51 IF A$="E" THEN GOSUB 110:PRINTCHR$(9);"SLUT":GOTO 152
52 IF A$<"0" OR A$>"7" THEN 49
53 GOSUB 54:GOTO 27
54 PRINT:PRINT"OPTIONER:"
55 PRINT" S=UNDELETE TO SEQ - ";
56 PRINT"P=UNDELETE TO PRG"
57 PRINT" U=UNDELETE TO USR - ";
58 PRINT"L=UNDELETE TO REL"
59 PRINT" W=WRITEPROTECT ON - ";
60 PRINT"N=NO WRITEPROTECT"
61 GET B$:IF B$="" THEN 61
62 K=0
63 FOR N=1 TO LEN(OP$)
64 : IF B$=MID$(OP$,N,1) THEN K=N
65 NEXT N
66 FB=2+32*VAL(A$)
67 ON K GOTO 72,77,82,87,92,96
68 GOTO 61
69 REM
70 REM SEQUENTIAL FILES
71 REM
72 IF W%(FB)<>0 THEN 97
73 W%(FB)=129:RETURN
74 REM
75 REM PROGRAM FILES
76 REM
77 IF W%(FB)<>0 THEN 97
78 W%(FB)=130:RETURN
79 REM
80 REM USER FILES
81 REM
82 IF W%(FB)<>0 THEN 97
83 W%(FB)=131:RETURN
84 REM
85 REM RELATIVE FILES
86 REM
87 IF W%(FB)<>0 THEN 97
88 W%(FB)=132:RETURN

```

```

89 REM
90 REM PROTECT FILE
91 REM
92 W%(FB)=W%(FB) OR 64:RETURN
93 REM
94 REM UNPROTECT FILE
95 REM
96 W%(FB)=W%(FB) AND 191:RETURN
97 PRINT"IKKE-LUKKEDE FILER KAN IKKE UNDELETES!!";
98 RETURN
99 REM
100 REM CHECK OM DER ER FORSKELLE
101 REM
102 ID=1
103 FOR N=0 TO 255
104 : IF BL%(N)<>W%(N) THEN ID=0
105 NEXT N
106 IF ID=1 THEN 118
107 REM
108 REM BEKRAEFT AT OPDATERING ØNSKES
109 REM
110 PRINT"ØNSKES OPDATERING (J/N)?"
111 GET A$:IF A$="" THEN 111
112 IF A$="N" THEN 118
113 IF A$<>"J" THEN 111
114 GOSUB 138
115 REM
116 REM CHECK OM DIRECTORY ER SLUT
117 REM
118 TR=W%(0):IF TR=0 THEN PRINT"IKKE FLERE FILER";CHR$(9):GOTO 152
119 SE=W%(1)
120 RETURN
121 REM
122 REM LAES EN BLOK
123 REM
124 OPEN 15,8,15,"I0:"
125 OPEN 8,8,8,"#"
126 PRINT#15,"U1";8;0;TR;SE
127 PRINT#15,"B-P";8;0
128 FOR N=0 TO 255
129 : GET#8,A$:IF A$="" THEN A$=CHR$(0)
130 : BL%(N)=ASC(A$)
131 NEXT N
132 CLOSE 8
133 CLOSE 15
134 RETURN

```

## Lagring af programmer på diskette

```
135 REM
136 REM SKRIV EN BLOK
137 REM
138 OPEN 15,8,15
139 OPEN 8,8,8,"*"
140 PRINT#15,"B-P";8;0
141 FOR N=0 TO 255
142 : PRINT#8,CHR$(W%(N));
143 : REM HUSK SEMIKOLON !!!!!
144 NEXT N
145 PRINT#15,"U2";8;0;TR;SE
146 CLOSE 8
147 CLOSE 15
148 RETURN
149 REM
150 REM OPDATER DISK-INDHOLD
151 REM
152 IFA$="J"THEN OPEN 15,8,15,"V0:"CLOSE 15
153 END
```

### 8.12.4 Brug af flere disk-stationer

Commadore's særlige serielle IEC-port tillader tilslutning af mere end een disk-station - eneste krav, der skal opfyldes er, at disk-stationerne tildeles forskellige device numre.

Har man lejlighed til at låne en disk-station i perioder, er en permanent løsning næppe gennemførlig i praksis. Device-nummeret er dog yderst enkelt at programmere - ja det er faktisk muligt at give en disk-station forskellige device numre for skrivning og læsning af programmer - f.eks. hvis man vil afprøve et program, der normalt kører på to disk-stationer.

Fremgangsmåden er:

1. Sluk for alle disk-stationer, undtagen den station, der skal programmeres.
2. Programmeringen foretages med kommandoerne:  

```
OPEN 1,8,15
PRINT#1,"M-W";CHR$(119);CHR$(0);CHR$(2);
;CHR$(skriv-dev +32);CHR$(læs-dev +64)
CLOSE 1
```
3. Næste station tændes, og programmeres om nødvendigt.

Programmeringen af device-nummeret forsvinder, når computeren eller disk-stationen reset'es eller slukkes. Ønskes en mere permanent løsning, må der foretages indgreb internt i disk-stationen. Rådfør dig med en erfaren tekniker, *inden* du foretage indgrebet, og husk, at der findes flere forskellige udgaver af printplader til disk-stationen.

### 8.13 DOS 5.1 til Commodore 64

Når DOS 5.1 er læst ind via loader-programmet (*wedge*) kan alle kommandoer kaldes ved først at indtaste ">" eller "@" efterfulgt af et eller flere tegn for den ønskede funktion. Indtastes tegnet "@" alene, fås en udskrift af disk-stationens fejlkanal.

**/programnavn**

LOAD'er programmet *programnavn* ind i BASIC-området.

**%programnavn**

LOAD'er et (*maskinkode*) program med navnet *programnavn* på den absolutte adresse, der er indeholdt i program-header'en. Kommandoen erstatter BASIC-udtrykket:

**LOAD"programnavn",8,1**

men med den lille finesse, at det *ikke* er nødvendigt at indtaste **NEW**, for at undgå en fejlmelding fra BASIC. Kommandoen kan således også benyttes til indlæsning af maskinkode-programmer, *efter* at du har indlæst et BASIC-program!

**^programnavn**

LOAD'er et BASIC-program, som umiddelbart efter indlæsningen RUN'es.

**<-programnavn**

SAVE'er et BASIC-program til diskette under navnet *programnavn*. Kommandoen **<-@@:programnavn** erstatter et program med navnet *programnavn* på disketten med programmet i hukommelsen, men metoden er ikke helt uden risiko. En langt bedre fremgangsmåde er, at benytte **RENAME** eller **SCRATCH** forud for **SAVE**. Så undgår du ubehagelige overraskelser i form af, at også andre programmer på disketten er blevet *erstattet* - navnene behøver ikke engang at have større ligheder!

**@**

Udskriver indholdet af fejlkanalen. Er alt i orden - dvs. nul blink i den røde lampe - fås meddelelsen:

**00 OK 00 00**

**>\$**

Læser diskettens directory (indholdsfortegnelse). Kommandoen kan benyttes med *wildcards* eller *jokere*. Processen virker, dog kun een eller to gange, hvorefter det er nødvendigt at foretage en komplet læsning af directory med **>\$**, inden brugen af *wildcards* igen er mulig.

**>#device**

Fortæller DOS 5.1, at alle efterfølgende kommandoer skal omdirigeres til

disk-station *drivenr.* Har du to disk-station med device-numrene 8 og 9, kan DOS'en med kommandoerne *>#9* og *>#8* omdirigeres til at arbejde på henholdsvis drive 9 og drive 8. Processen kan gentages et vilkårligt antal gange.

>Q

Quit-kommando. Udkobler DOS 5.1 af Commodore 64'erenes operativ-system. Kommandoen bør altid benyttes, inden du starter et maskinkode-program, som ikke udtrykkeligt er specificeret til at leve i fredelig sameksistens med DOS 5.1. Hvis området hex CC00 til hex CFFF ikke er forandret af et andet program, kan DOS 5.1 aktiveres igen med kommandoen SYS 52224.

>NØ:diskettenavn,ID

Formatterer en diskette. Disketten får navnet *diskettenavn* og ID-koden *ID*. Er disketten allerede formatteret, men ønskes en frisk start, indtastes blot >NØ:*diskettenavn*, som kun fører til at directory og sektorbelægningen (BAM) initialiseres. Det tager kun få sekunder - sammenlignet med de godt 80 sekunder, som en re-formattering varer.

ID: ID-koden på en diskette, gør det muligt for operativ-systemet at afgøre, om der er skiftet diskette, og dermed at undgå overskrivning af sektorer, som på den først benyttede diskette var frie, men på den nye diskette er belagte. Sørg derfor for, at ingen af dine disketter har samme ID-nummer. Så er risikoen for ubehagelige overraskelser meget små.

>SØ:programnavn

Sletter filen *programnavn*. Et godt tip: Anvend aldrig wildcards i SCRATCH-kommandoen!. Læses fejl-kanalen (indtast "@") efter SCRATCH fortæles, hvor mange filer, der er slettet - f.eks. 01 FILES SCRATCHED NN 00, hvor NN er antallet af slettede filer. Hvis SCRATCH-kommandoen ikke sletter nogen filer, udskrives "00", men intet fejl-blinkeri opstår. OBS: SCRATCH-kommandoen må ikke benyttes til sletning af filer, der ikke er korrekt CLOSE'de (mærket med \* ud for filtypen - f.eks. \*PRG). Anvend i stedet Verify-kommandoen.

>RØ:nyt navn=gammelt navn

RENAME'r en fil med navnet *gammelt navn*. Det nye filnavn bliver *nyt navn*.

>CØ:kopinavn=originalnavn

COPY-funktion. Laver en kopi af filen *originalnavn* og tildeler kopien navnet *kopinavn*. Kopiering kan ikke foregå mellem to diskettestationer. Kopiering af relative filer er ikke mulig - kun *programfiler* (P eller PRG) *sekventielle filer* (S eller SEQ) og *user filer* (U eller USR kan kopieres).

>I

Initialize dvs. indlæsning af oplysning om diskettens sektorbelægning (BAM). Benyttes udelukkende disketter med forskellige ID-koder er kommandoen ikke strengt nødvendig, men konsekvent anvendelse efter hvert disketteskift sikrer i hvert fald mod alle overraskelser - især kan kommandoen anbefales forud for SAVE med REPLACE (<-Ø:programnavn).



>V

**Validate.** Må ikke forveksles med BASIC-kommandoen **VERIFY**. Kommandoen fører til, at diskstationen gennemlæser *alle* filer på disketten og noterer de sektorer, der er belagt af filer i *directory*'en, i **BAM**. Da *random filer* ikke er opført i *directory*, vil alle referencer til disse (blokkene kan markeres i **BAM** med *Block Allocate* kommandoen **B-A**) blive slettet, og overskrivning af disse blokke kan ske, efter brugen af *Validate disk*-kommandoen! Kommandoen skal anvendes, når du ønsker at fjerne filer, der ikke er korrekt **CLOSE**'de, fra *directory*'en (mærket med \* ud for filtypen - f.eks. \*PRG). Benyttes **SCRATCH**-kommandoen, kan følgerne blive helt uoverskuelige! Render du ind i *stjerne-fænomenet* på en diskette indeholdende *random files*, må du huske at markere de belagte sektorer med *Block-Allocate* kommandoen, straks *Validate* er afsluttet!

## Kapitel 9

### Kommunikation med printere og andet udstyr

Kommunikation med printere er en enkel affære, så længe man indskrænker sit valg til Commodores egne printere. De mere hobby-betonede printere kan normalt tilsluttes computerens serielle udgang i lighed med disk-stationen, mens Commodores større printere kræver anvendelse af et IEEE-interface.

Har man adgang til printere, der ikke følger Commodore's standard, stiger problemerne - ofte opgiver man totalt at prøve, at få computeren til at kommunikere med andet udstyr.

#### 9.1 Commodore 64 user port

Commodore 64 er som hovedparten af Commodores computere udstyret med en *user port* - en række in/output linjer, som kan anvendes til kommunikation med andet udstyr. Normalt benyttes denne port til RS-232 seriel kommunikation (se evt. side 92) - f.eks. printere, modem eller andre computere - men porten kan let konfigureres, så den åbner mulighed for parallel kommunikation (se evt. side 92) med andre computere eller printere med Centronics interface.

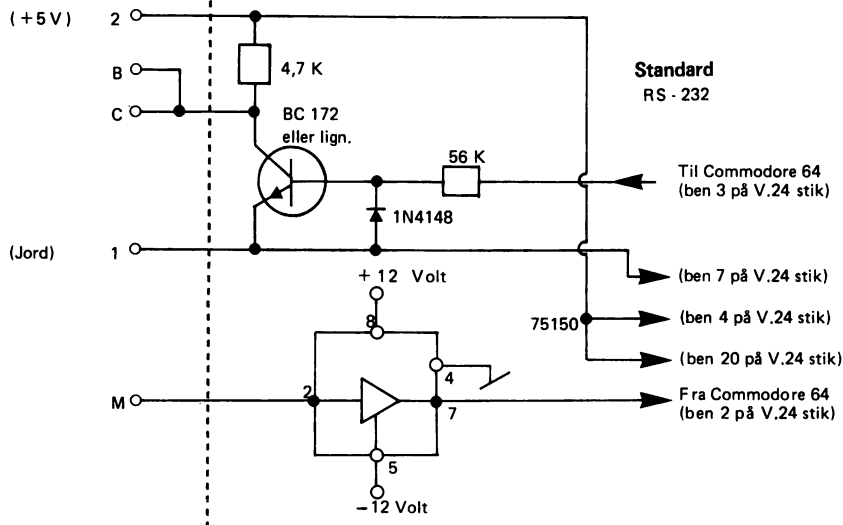
##### 9.1.1 RS-232 kommunikation

Commodore 64 er udstyret med en *ikke-standard* RS-232 udgang. Normalt forventer RS-232 udstyr signalniveauer i størrelsesordenen mellem -15V til -3V (bitværdi 1) og +3V til +15V (bitværdi 0). Computeren selv arbejder med signalniveauer i størrelsesordenen 0V (bitværdi 0) og 5V (bitværdi 1), så allerede på dette grundlæggende område er vi nødt til at lave et par modifikationer, inden vi overhovedet kan kommunikere med andet udstyr - (bortset fra en anden Commodore 64 eller VIC-20). Kredsløbet, der vises herunder, kan løse niveau-problemet ved simpel kommunikation til en tilsluttet printer m. RS-232 eller ved udveksling af data mellem to computere. Kredsløbet er anvendt med held til kommunikation mellem IBM-PC'ere eller dermed kompatible maskiner.

# Commodore 64

## USER PORT

## Kommunikation med printere og andet udstyr



### Diagram for RS-232 niveautilpasning på Commodore 64

Det viste kredsløb er beregnet til 3-line handshake (se under Åbning af kanal til RS-232), og tilsluttes user-porten på følgende måde (anvend et 2x12 polet stik med 0.156 tommers benafstand):

BEN	SIDE	BENÆVNELSE	RETNING	BRUG
B+C	BUND	RECEIVED DATA	IND	3/X
D	BUND	REQUEST TO SEND	UD	X (3)
E	BUND	DATA TERMINAL READY	UD	X (3)
F	BUND	RING INDIKATOR	IND	user
H	BUND	RECEIVED LINE SIGNAL	IND	X
J	BUND	anvendes ikke	-	user
K	BUND	CLEAR TO SEND	IND	X
L	BUND	DATA SET READY	IND	X
M	BUND	TRANSMITTED DATA	UD	3/X
A+N	BUND	SIGNAL-JORD/STEL	-	3/X
2	TOP	+5 Volt DC (100 mA)	-	-
10	TOP	9 Volt AC (+ fase)	50 mA/ca. 400 mA	
11	TOP	9 Volt AC (- fase)	uden kassette	

X = X-line handshake

3 = 3-line handshake

(3)= fast på high (5V) ved 3-line handshake

Tilpasning af niveauerne er dog langt fra løsningen på vore problemer. Inden vi overhovedet kan udnytte RS-232 udgangen, må vi tage stilling til kommunikationsform, hastigheder osv. Først når disse faktorer er i orden, kan vi gøre os håb om at få f.eks. to computere til at tale sammen.

### 9.1.1.1 Baudrate, parity osv.

RS-232 kommunikation kan foretages efter flere forskellige protokoller - "samtale-normer". Med Commodore 64 er vi i praksis begrænset til *asynkron* kommunikation - dvs. at data overføres, når de står til rådighed, og ikke i faste intervaller (*synkront*). Asynkron kommunikation er baseret på de følgende signaler (både for *senderen* og *modtageren*):

Et byte (eller een dataenhed) omformes til en serie impulser, bestående af 1 *startbit*, flere *databits*, evt. et *parity-bit*, og et eller flere *stop-bits*. Impulserne har en nøje defineret varighed - bestemt af *baudrate* - og når der ikke overføres data på kommunikationslinjen, holdes denne i en fast tilstand. Enkeltimpulsernes betydning er:

#### Startbit

Informerer modtageren om, at der nu udsendes een dataenhed fra senderen. Varigheden af start-bit'et er 1/300 sekund ved 300 baud, 1/1200 sekund ved 1200 baud osv. Niveaueet svarer altid til niveaueet for et 0-bit - også kaldet *space*. Når der ikke kommunikeres data på linjen, er niveaueet altid det samme som for et 1-bit - også kaldet *mark*.

#### Databits

De egentlige databits overføres i faste intervaller, umiddelbart efter start-bittet. Modtager og sender skal være indstillet til at forstå samme *dataformat*. Der kan vælges mellem 5, 6, 7 eller 8 bits pr. dataenhed, og denne størrelse *skal* være fastlagt, *inden* kommunikationen begynder. Har man f.eks. valgt et dataformat på 8 bits, vil senderen udsende 8 bitværdier, hver med en varighed på 1/300 sekund ved 300 baud, 1/1200 sekund ved 1200 baud osv. Niveaueet - mark eller space - bestemmes af de enkelt-bit, der skal overføres. F.eks. vil værdien 255 (8 bits, der alle er 1) medføre at kommunikationslinjen holder *mark*-tilstanden i 8/300 sekund ved 300 baud.

#### Parity

Afhængigt af antallet af *databits* kan man vælge mellem flere former for *parity*. Parity-bit'et er en kontrolinformation, som gør det muligt for modtageren at afgøre, om der er opstået fejl i de data, der er modtaget. Metoden er ikke 100% sikker, men ofte tilstrækkelig. Konstateres en parity-fejl, kan modtageren bede senderen om at gentage en større eller mindre del af de overførte data - evt. gentage hele transmissionen. Parity har følgende muligheder:

- **Ingen parity** - der overføres kun databits. Dette er *altid* tilfældet, når kommunikationen består af 8 databits.
- **Mark parity** - Parity bit'et er altid 1 (mark) - kun muligt ved 5, 6 og 7 bit dataformaterne.
- **Space parity** - Parity bit'et er altid 0 (space) - kun muligt ved 5, 6 og 7 bit dataformaterne.
- **Ulige (odd) parity** - Parity bit'et sættes til 1, hvis der er et lige antal 1'ere i databit'ene, og 0, hvis databit'ene indeholder et

ulige antal l'ere. F.eks. vil parity-bit'et blive sat til værdien 1, hvis databit'ene, der overføres, indeholder bitfølgen 1110111 i et 7-bit format! Har man vedtaget, at benytte ulige parity i kommunikationen er modtageren klar over, at der er opstået en fejl, når data-bit'ene indeholder et ulige antal l'ere!

- **Lige (even) parity** - Parity bit'et sættes til 0, hvis der er et lige antal l'ere i databit'ene, og 1, hvis databit'ene indeholder et ulige antal l'ere. F.eks. vil parity-bit'et tildeles værdien 0, hvis databit'ene, der overføres, indeholder bitfølgen 1110111 i et 7-bit format!

Varigheden af et parity-bit (5, 6 eller 7 bit dataformat) er 1/300 sekund ved 300 baud, 1/1200 sekund ved 1200 baud osv.

### Stopbit

Stopbits benyttes til at fortælle modtageren, at transmissionen (af een dataenhed) er afsluttet. Stopbits har altid værdien 1 (mark) og hvert stop-bit har en varighed på 1/300 sekund ved 300 baud, 1/1200 sekund ved 1200 baud osv. Hvis modtageren konstaterer en kortere varighed end det forventede (f.eks. forårsaget af støj på en telefonlinje), er modtageren klar over, at der er noget galt med transmissionen. Commodore 64 åbner mulighed for at vælge mellem 1 eller 2 stopbits.

#### 9.1.1.2 Formatvalg på Commodore 64

Det aktuelle kommunikationsformat fastlægges med OPEN-kommandoen:

OPEN *lf,2,sa,byte1+byte2+byte3.....*

hvor *lf* er det logiske filnummer, *sa* er den sekundære adresse (normalt uden betydning, men skal medtages), og 3 databytes, hvoraf kun *byte1* og *byte2* benyttes til at fastlægge kommunikations formatet (protokollen). Byte 3 m.m. har ingen betydning i praksis, men kan benyttes til programmering af speciel software/hardware, hvis ikke *sa* benyttes til dette formål. Mulighederne er:

**BYTE 1**

bit 7	0	1 stopbit	0
	1	2 stopbits	128
bit 5/6	00	8 bits dataformat (ingen parity)	0
	01	7 bits dataformat	32
	10	6 bits dataformat	64
	11	5 bits dataformat	96
bit 4	-	bruges ikke	
bit 0-3	0000	anvendes ikke	
	0001	50 baud	1
	0010	75 baud	2
	0011	110 baud	3
	0100	134.5 baud	4
	0101	150 baud	5
	0110	300 baud - normalt BASIC	6
	0111	600 baud	7
	1000	1200 baud - maksimalt pålidelige	8
	1001	1800 baud	9
	1010	2400 baud - typisk 3% fejl - BASIC	10
øvrige kombinationer anvendes ikke			

Ved modtagelse er 300 baud den maksimalt anvendelige hastighed i BASIC, hvis data sendes vedvarende. Er der jævnlige pauser på sendesiden (f.eks. for hver skærmlinje etc.) kan op til 1200 baud anvendes i BASIC. Data kan udsendes med hastigheder på op til 1200 baud med en fejlmargen på under 1 pr. 100.000 bytes! Ved højere hastigheder er Commodore 64 meget upålidelig - f.eks. svinger fejlene mellem 25 og 40 pr. 1000 bytes ved 2400 baud, hvilket må betegnes som uacceptabelt. Værdierne er konstaterede ved praktiske forsøg - flere forsøg på op til 700.000 bytes ad gangen ved 8 byte/1 stopbit kommunikation! Testen på 2400 baud er dog altid afbrudt efter ca. 50-100.000 byte, hvor fejlfrekvensen normalt begyndte at svinge ind omkring 2.8 til 3.2 %. Testen er foretaget med udstyr, der kan arbejde sikkert op til 9600 baud, og kommunikationen er foretaget med det illustrerede interface i simpel 3-tråds teknik (se herunder)

**BYTE 2**

bit 7/6/5	xx0	ingen parity (vælges ved 8 bit)	0
	001	ulige parity	32
	011	lige parity	96
	101	fast parity 1	160
	111	fast parity 0	224
bit 4	0	full duplex	0
	1	half duplex	16
bit 3/2/1	-	anvendes ikke	
bit 0	0	3 - line	0
	1	X - line	1

*Full duplex* betyder, at der kan sendes og modtages på samme tid, mens *half duplex* betyder, at der *enten* kan sendes *eller* modtages, men *ikke* begge dele på samme tid. Half duplex anvendes kun sjældent til hobby-brug, idet der kræves at *både* modtager og afsender skal følge en meget streng samtalenorm (protokol).

#### 9.1.1.3 Abning og lukning af RS-232 kanal

Brugen af RS-232 er ikke helt problemfri på Commodore 64. Begrænsningerne er omend ikke umulige, så dog temmelig alvorlige, og der skal tages behørigt hensyn til begrænsningerne:

- OPEN-kommandoen skal anvendes som den allerførste programsætning i et program. Når der åbnes til RS-232 slettes alle variabler, og alle andre kanaler lukkes. I praksis udføres der en CLR kommando, hver gang, der åbnes til RS-232!!!
- Der kan læses fra disk, men *ikke* skrives til disk, mens RS-232 kommunikation foregår. I praksis er det absolut en fordel, først at læse alle data, der skal overføres via RS-232, for derefter at foretage transmissionen, ligesom det er en fordel, at modtage *alle* data, vente et lille stykke tid, hvorefter data kan skrives til diskette.
- CLOSE-kommandoen *skal* anvendes som det sidste i et program. Kommandoen sletter *alle* data i computerens hukommelse. Ligeledes sker det ofte, at maskinen "hænger", når CLOSE-kommandoen udføres, efter at disk-kommandoer har været udført.

Når en RS-232 kanal åbnes, reserveres der plads til to buffere, hver på 256 bytes. Disse buffere placeres *over top of BASIC* pointeren, der justeres nedefter med ialt 512 bytes. Dette er den direkte årsag til at CLR-kommandoen er en del af OPEN og CLOSE processerne. OPEN kommandoen fastlægger det benyttede dataformat - f.eks. vil:

OPEN 222,2,2,CHR\$(8)+CHR\$(Ø)

fastlægge parametrene til:

1200 baud  
8 bits  
1 stopbit  
no parity  
full duplex  
3 tråds handshake  
automatisk line-feed efter RETURN

Der findes to former for *handshake* - kommunikationsstyring. 3-line handshake betyder, at ingen kontrol-ledninger benyttes. Kun tre signal-linjer anvendes, nemlig *sende*, *modtage* og *stel/jord*. X-line handshake benytter yderligere linjerne RTS (request to send), DTR (data terminal ready), DCD (received line signal), CTS (clear to send) og DSR (data set ready), men da signalerne ikke følger standarden, skal der benyttes en større omgang elektronik, før metoden kan anvendes i praksis!

Bemærk anvendelsen af det logiske filnummer 222. Ethvert logisk filnummer med en værdi over 127 tvinger Commodore 64 til at tilføje en *line-feed* karakter (ASCII 10) efter hver *carriage return* (ASCII 13). Mange computere og printere kræver, at hver *carriage return* følges af en *line-feed*, før der foretages et linjeskift. Modtager f.eks. en printer (af denne type) kun en *carriage-return* karakter, flyttes valseen kun frem til begyndelsen af linjen, men der foretages intet linjeskift. Åbnes med et logisk filnummer under 128, vil en sådan printer skrive alle tekstlinjer oveni hinanden!

Bemærk, at når CLOSE-kommandoen udføres, slettes indholdet i RS-232 bufferne straks. Der ventes ikke til alle karakterer er afsendt! Benyttes følgende lille rutine, er man altid sikker på, at alle karakterer er sendt, når CLOSE-kommandoen udføres:

```
900 IF (PEEK(670)-PEEK(669)) THEN 900  
910 CLOSE 2
```

De to adresser er henholdsvis slutningen og begyndelsen af RS-232 output-bufferen, og differensen er antallet af karakterer i bufferen. Først når differensen bliver nul, udføres linje 910.

#### 9.1.1.4 Datakommunikation via RS-232

Der kan både skrives til og læses fra en RS-232 port. Skrivning foregår helt problemløst med kommandoen:

PRINT#*lf*,*data*.....

og er der tale om at sende en listning via RS-232 til et modem eller evt. en anden computer, benyttes kommandoen:



## CMD *lf* :LIST

Alle programmer, der er benyttet i denne bog, er overført via det viste interface til en den computer, forfatteren benytter i sit daglige arbejde. Alle programmer er først fremstillet på et professionelt tekstbehandlingsprogram, konverteret til Commodore's specielle interne format, og sendt til Commodore 64 via RS-232. Programmerne er derefter blevet prøvekørt og evt. modificerede, hvorefter en listning igen er sendt til tekstbehandlingsanlægget. Det er denne sidste listning, der er trykt. Kommunikationshastighederne, der har været anvendt, er 300 baud, når der primært skulle sendes til Commodore 64, og 1200 baud, når data primært skulle overføres fra Commodore 64. Med disse transmissionshastigheder optræder der praktisk taget ingen fejl!

Mens det er forholdsvis problemfrit at sende data til andet udstyr, er modtageprocessen knap så enkel - for at sige det mildt. For det første er INPUT# kommandoen ikke særlig anvendelig i praksis. Årsagerne er:

- Kommaer og RETURN karakterer ignoreres af INPUT#-kommandoen.
- INPUT# kommandoen skal møde en RETURN karakter senest efter 88 indlæste tegn - det maksimale antal tegn, som BASIC input-buffere (se side 112) kan rumme (må ikke forveksles med RS-232 buffernes størrelse).
- Hvis INPUT# kommandoen *ikke* møder en RETURN karakter "hænger" systemet - dette kan f.eks. forårsages af fejl på linjen m.m.

Selvom GET# kommandoen ikke er den mest effektive, kan det dog varmt anbefales, at benytte denne i stedet. Den største fordel er, at kommandoen aldrig får maskineriet til at gå i "baglås". Ulempen er, at man *aldrig* kan være helt sikker på, om data har indeholdt værdien CHR\$(0) eller om RS-232 buffere bare har været tom. Ganske vist er det muligt at teste status-variablen for en tom buffer, men metoden er *ikke* pålidelig (læs mere om status i det følgende afsnit)!

Problemet kan omgås på to måder. Den enkleste er at udelukke ASCII-værdien 0 fra det gyldige karaktersæt. Er der tale om rene tekstinformationer, skulle problemet være let at løse. Er der derimod tale om programmer (maskinkode eller BASIC) må en anden fremgangsmåde benyttes. I disse tilfælde kan det anbefales at benytte de følgende rutiner, som omformer eet byte til en to-byte hex-kode for sending, og konverterer denne kode tilbage til eet byte igen på modtagesiden. Rutinerne kan udformes således:

```
100 REM FORUDSÆTNINGER
110 OPEN 2,2,CHR$(8)+CHR$(0):REM VALGFRIT
120 TABEL$="0123456789ABCDEF"
130 A=ASC(karakter)
```

Alle informationer, der skal afsendes, leveres til rutinen i form af en ASCII-værdi i variablen A:

```
590 REM SENDING
600 B=INT(A/16):A=A AND 16
610 PRINT#2,MID$(TABEL$,A+1,1);MID$(TABEL$,B+1,1);
```

620 RETURN

Denne subrutine kaldes, hver gang et byte skal læses fra RS-232 porten. Rutinen kalder i sig selv en subrutine, der indlæser 2 byte-værdier fra RS-232, og rutinen konverterer disse til et tegn i IND\$ og en ASCII værdi fra 0 til 255 i B.

```
680 REM 1 BYTE
690 B=0
700 FOR N=0 TO 1
710 : GOSUB 810:B=B+A*16~N
720 NEXT N
730 IND$=CHR$(B)
740 RETURN
```

Denne rutine kontrollerer alle modtagne bytes, og giver en fejlmelding, hvis formatet af en eller anden grund ikke er overholdt.

```
800 REM MODTAGELSE AF DELBYTES
810 GET#2,A$:IF A$="" THEN 810
820 IF A$<"0" OR A$>"F" THEN 850
830 IF A$>"0" THEN A=ASC(A$)-55:RETURN
840 IF A$<="9" THEN A=ASC(A$)-48:RETURN
850 PRINT "DATAFEJL":GOTO 810
```

Rutinen kan med fordel ændres, så den automatisk bryder af efter et forud fastlagt stykke tid (mindst mellem 1 og 2 sekunder). Til markering af et evt. time-out anvendes variablen TX, som indeholder det antal sekunder, man ønsker at vente, inden transmissionen afbrydes. Rutinen kaldes hver gang med TX sat til den ønskede værdi - f.eks. 2 sekunder (indsæt linjen 705 TX=2 i det foregående). Subrutinen kan ændres til:

```
800 REM MODTAGELSE AF DELBYTES
805 T$=TI$:IF TX<0 THEN PRINT "TIMEOUT":RETURN
806 IF T$<>TI$ THEN TX=TX-1:GOTO 805
810 GET#2,A$:IF A$="" THEN 806
820 IF A$<"0" OR A$>"F" THEN 850
830 IF A$>"0" THEN A=ASC(A$)-55:RETURN
840 IF A$<="9" THEN A=ASC(A$)-48:RETURN
850 PRINT "DATAFEJL":GOTO 806
```

TI\$ er ligesom TI en reserveret variabel, i dette tilfælde udnyttet til måling af tidsforskelle; urfunktionen som sådan udnyttes ikke. Når rutinen kaldes, sættes variablen T\$ lig med TI\$. Værdien er underordnet. I linje 806 vil uligheden være opfyldt mindst een gang i sekundet - første gang kan være lige efter; derfor testes på mindre end nul og ikke lig med nul. Det betyder i praksis, at time-out testen har en usikkerhed på 1 sekund - fra lige over 2 sekunder til eksakt 3 sekunder, inden TIMEOUT meldingen udløses. I praksis anvender man ofte timeout-værdier på 1 sekund til indikation af, om der er noget galt under selve modtagelsen, og hvis kommunikationen består af både sending og modtagelse, benyttes ofte et timeout på 10 sekunder fra sidste sending til modtagelse af første karakter. Er der ikke modtaget nogen karakter inden da, kan man enten vælge at afbryde forbindelsen automatisk, eller tilbyde brugeren et valg mellem afslutning eller fortsættelse af transmissionen. Denne valgmulighed gentages så hvert tiende sekund, indtil der modtages en karakter, hvorefter man går tilbage til et timeout på f.eks.

et sekund imellem hver overført karakter. Er der gået mere end et sekund, hvor man ikke har modtaget en karakter, indikerer det, at senderen ikke har mere materiale p.t., og man kan kvittere for meddelelsen, lagre de modtagne data (indenfor 10 sekunder), sende egne meddelelser etc., og igen vende tilbage til 10 sekunders timeout for modtagelse af flere karakterer osv. Processen kan forekomme langsommelig, men anvendes hyppigt i praksis under en eller anden form - især hvis man benytter *half-duplex* kommunikation.

### 9.1.2 STATUS og RS-232

Som allerede antydnet har STATUS-variablen en særlig betydning under RS-232 transmission. De enkelte bits har følgende betydning:

BIT	BETYDNING
0	PARITY ERROR
1	FRAMING ERROR
2	RECEIVER BUFFER OVERRUN
3	RECEIVER BUFFER EMPTY
4	CTS SIGNAL MISSING
5	anvendes ikke
6	DSR SIGNAL MISSING
7	BREAK DETECTED

CTS og DSR SIGNAL MISSING samt BREAK DETECTED har kun betydning, hvis X-LINE kommunikations-formatet benyttes.

RECEIVER BUFFER EMPTY indikerer, om der befinder sig data i modtage bufferen. Har man forsøgt et GET#, vil dette bit være sat, hvis der ikke befinder sig flere tegn i bufferen. I langt de fleste tilfælde virker indikatoren efter hensigten, men i praksis kan funktionen svigte. En fornuftig forklaring herpå kan ikke gives, blot er dette konstateret i praksis i enkeltstående situationer. Et programs virkemåde bør derfor ikke baseres udelukkende på anvendelsen af denne indikator.

RECEIVER BUFFER OVERRUN indikerer, at det program, man benytter, ikke er i stand til at læse de karakterer, der indløber, hurtigt nok. Problemet kan optræde, hvis man forsøger at modtage signaler med en baudrate på 1200 baud eller mere i et BASIC-program. Især hvis man også udskriver til skærmen under kommunikationen. I disse tilfælde er man normalt henvist til maskinkode. Ved transmissionshastigheder på 300 baud eller derunder optræder der normalt ikke de store problemer i BASIC, medmindre der foretages mange operationer mellem læsning af hver enkelt karakter. I de tilfælde kan det anbefales at koncentrere rutinerne om indlæsning, og foretage behandlingen af data, når transmissionen er slut, eller når senderen afventer data - hvis det kan gøres inden en evt. time-out træder i kraft!

PARITY ERROR bit'et sættes, hvis computeren konstaterer en afvigelse i den forventede parity-tilstand. Undersøg om den korrekte parity er valgt, eller specificer 8 bit, uden parity, og ignorer bit 7 i de modtagne signaler (kun muligt, hvis der er tale om 7 bit plus parity (odd, even, mark eller space)).

Der er tale om fejl i data transmissionen, men i mange tilfælde - f.eks. hvis der er tale om tekst - kan man senere rekonstruere de korrekte data.

FRAMING ERROR indikerer, at der er sket et spring i modtagelsen, således at de enkelte bits i transmissionen af een dataenhed ikke ankommer indenfor den afsatte tid, eller er forskudt tidsmæssigt - f.eks. 1 bit. Fejlen kan skyldes støj på linjen, forkert baudrate eller en afvigelse i transmissionshastigheden, som ligger udenfor det tilladelige - f.eks. at en computer sender med 1000 eller 1400 baud hastighed i praksis, selvom hastigheden specificeres til 1200 baud. Dette sidste tilfælde optræder kun sjældent, idet en sådan fejl normalt skyldes så alvorlige fejl i en computer, at brugeren straks er klar over, at der er noget galt!

### 9.2 Parallel kommunikation via user porten

Selvom user porten normalt er forbeholdt RS-232 er der intet i vejen for at udnytte den til andre formål. Ialt findes 11 signal-linjer, som kan udnyttes på enkel måde - alle linjerne befinder sig på samme CIA 6526 kreds, og er ført ud på de følgende placeringer på user-porten:

BEN	BIT	
8	PC2	special
B	FLAG2	special
C	0	port B
D	1	port B
E	2	port B
F	3	port B
H	4	port B
J	5	port B
K	6	port B
L	7	port B
M	2	port A

PC2 linjen er normalt 1 (high eller +5 Volt), men hver gang der skrives til eller læses fra Port B, går signal-linjen på nul i 1 mikrosekund (1/1.000.000 sekund). Denne signal-linje kan iøvrigt ikke kontrolleres.

Flag2 linjen har den modsvarende funktion til PC2. Hver gang et signal til denne linje går fra 1 til nul, sættes bit 4 på adressen 56589 til værdien 1. Når man læser adresse 56589, sættes dette bit tilbage til 0. Flag 2 kan anvendes til at udløse en NON-MASKABLE (NMI) interrupt, som tvinger processoren til at afbryde alle løbende processer, og springe til den adresse, der er lagret på adresserne FFFA/FFFB hex. Denne funktion udnyttes af RS-232 porten til modtagelse af karakterer, og det er også denne egenskab, der skaber alle problemerne med RS-232. NMI-muligheden kan kun udnyttes, hvis man virkelig behersker maskinkode, og kender Commodore 64' interne hardware opbygning til sidste detalje. Derfor vil NMI-muligheden ikke blive udnyttet i det følgende. Bittet kan dog udnyttes til at indikere en eller

anden tilstand i tilsluttet udstyr - f.eks. at der er data parat til indlæsning.

Port A har adressen 56576 og Port B har adressen 56577. Begge porte kan både benyttes til læsning (input) og skrivning (output). Funktionen bestemmes af retningsregistrene, som har adresserne 56578 (port A) og 56579 (port B). Retningsregistrene bestemmer dataretningen for hvert portbit. Sættes et retningsbit til værdien 0, vil det tilsvarende portbit indstille sig til læsning (input) fra ydre udstyr, og sættes et retningsbit til værdien 1, vil det tilsvarende portbit virke som output til ydre udstyr. Vær meget omhyggelig, når du tilslutter ydre udstyr på Commodores user port, idet skaderne kan blive meget omfattende, hvis der sker fejl! Kredse accepterer kun standard TTL-niveauer, og er i stand til at drive en TTL belastning. Hvis du ikke ved, hvad der menes hermed, så lad være med at eksperimentere!

Lad os se på et praktisk eksempel på anvendelsen. Overførsel af data mellem to Commodore 64 computere. Den ene udnævnes til "HERRE" eller "MASTER" og den anden til "SLAVE".

Masteren kontrollerer al kommunikation og bestemmer, om der skal sendes til slaven eller modtages fra slaven. Dette indikeres ved at sætte bit 2 på port A til henholdsvis værdien 1 eller 0. Når data er klar, skrives de på port B, og signalledningen PC2 går på nul et kort øjeblik. Forbindes PC2 på masteren til FLAG2 på slaven, kan slaven kontrollere, hvornår data er gyldige. Når slaven læser port B, går PC2 på slaven på nul et kort øjeblik, og er denne linje forbundet til FLAG2 på masteren, kan denne kontrollere hvornår data er læst. De otte datalinjer, er PB0 til PB7.

Forbindelsen mellem de to Commodore 64 maskiner, skal se således ud:

BEN	BIT		BEN	BIT
B	PC2	special	B	FLAG2
B	FLAG2	special	B	PC2
C	0	port B	C	0
D	1	port B	D	1
E	2	port B	E	2
F	3	port B	F	3
H	4	port B	H	4
J	5	port B	J	5
K	6	port B	K	6
L	7	port B	L	7
M	2	port A	M	2

Lad os først se på MASTER siden. Klargøring til sending ser således ud:

```
200 POKE 56578,PEEK(56578)OR 4:REM BIT 2 PORT A TIL OUTPUT
210 POKE 56579,255:REM PORT B TIL OUTPUT
220 X=PEEK(56589):REM SIKRER AT FLAG2 ER NUL
230 POKE 56576,PEEK(56576)OR 4:REM RETNING MEDDELES TIL SLAVE
```

Slaven klargøres med:

```
200 POKE 56578,PEEK(56578)AND 251:REM BIT 2 PORT A TIL INPUT
210 X=PEEK(56589):REM SIKRER AT FLAG2 ER NUL
220 REM SLAVEN KONTROLLERER DATARETNING
230 A=PEEK(56576) AND 4:REM FIND RETNING
240 IF A THEN POKE 56579,0:GOSUB modtagelse:GOTO 230
250 POKE 56579,255: GOSUB sending:GOTO 230
```

For hvert byte, der skal overføres til slaven, udføres denne rutine:

```
290 find data
300 POKE 56577,byte
310 IF NOT(PEEK(56589)AND 16) THEN 310
320 GOTO 290
```

I linje 310 venter masteren, indtil data er læst - bit 4 sat. Slaven udfører følgende rutine for hvert byte:

```
300 IF NOT(PEEK(56589)AND16)THEN 300
310 B=PEEK(56577)
```

Herefter udføres lagring eller tolkning af det overførte byte, hvorefter der returneres til linje 240, der springer til linje 230 for fornyet kontrol af dataretningen.

Ønsker masteren at modtage data, indikeres dette med:

```
400 POKE 56576,PEEK(56576)AND251
410 POKE 56579,0:REM PORT B TIL INPUT
420 X=PEEK(56589):REM SIKRER AT FLAG2 ER NUL
```

Hvorefter masteren går i ventestilling

```
430 IF NOT(PEEK(56589)AND16)THEN GOTO 430
440 B=PEEK(56577):GOSUB databehandling
```

Efter modtagelsen, skal masteren afgøre om modtagelsen skal fortsætte eller ej. Slaven udfører følgende operation, så længe masteren indikerer, at den er i modtagestilling:

```
390 GOSUB find data
400 POKE 56576,byte
410 IF NOT(PEEK(56589)AND16)THEN GOTO 410
420 RETURN
```

Udover denne grund-kommunikation, kan man udvikle særlige kommunikationsprotokoller - samtaleformer - som f.eks. kan oplyse om hvilke informationer, der skal overføres, hvor mange bytes osv. Denne del af kommunikationen må skræddersys til læserens behov og ønsker. Kommunikationshastigheden er meget høj - selv i BASIC. Benyttes maskinkode kan man - afhængigt af kommunikationens form - nå hastigheder på over 10.000 bytes pr. sekund. Eller mere 25 gange så hurtigt, som dataoverførslen mellem disk-station og computer, og mere end 300 gange så hurtigt som mellem computer og kassettebåndoptager!!

### 9.3 Centronics interface

User porten kan også med virkeligt godt udbytte anvendes, hvis man ønsker at benytte en af de mange printere, der er udstyret med et Centronics interface. Løsningen er faktisk så enkel, at man ofte undres over, at Commodore ikke har inkluderet denne mulighed fra starten. Begrundelsen kan i forfatterens øjne kun være, at Commodore ønsker at mindske konkurrencen fra andre printer leverandører til et absolut minimum. Det bekymrer åbenbart ikke Commodore, at mange brugere er mere end almindeligt irriteret over denne beslutning.

Men vil Commodore ikke, kan vi da heldigvis selv. Herunder er en komplet listning på en Centronics-driver til Commodore 64. Driveren kan placeres i området C000 uden at kolliderer med DOS 5.1, og flere karaktersæt kan anvendes/indlæses efter behov. Den viste driver er kun egnet til brug for matrix-printere, og kræver foruden den egentlige kode 2k til alfabetet. Valget af denne fremgangsmåde skyldes udelukkende at de færreste printere er i stand til at gengive Commodores karaktersæt, og skal vi have rådighed over hele karaktersættet - med mulighed for selv at fremstille karakterer til specielle formål - er den valgte løsning, den mest fleksible. Koderne fra 0 til 127 er det normale karaktersæt, og koderne fra 128 til 255 er reverse, men om ønsket kan disse udskiftes til et alternativt alfabet. Ønsker man at anvende flere alfabetter - hvad siges om russisk eller græsk - behøver man blot at fremstille de 2k data (256x8 bytes), og SAVE dem til disk (evt. kassette), hvorfra de indlæses til adresse C1C6, hvis rutinen anvendes uforandret. Kommandoen:

LOAD "ALFABET 3",8,1

klarar resten - evt. kan man kalde en lille maskinkode-rutine, der indlæser alfabetet (se kapitlet om maskinkode).

Forbindelsen til printeren skabes således:

COMMODORE	FUNKTION	CENTRONICS
A	JORD/STEL	17/14
B	BUSY	11
C	BIT 0	2
D	BIT 1	3
E	BIT 2	4
F	BIT 3	5
H	BIT 4	6
J	BIT 5	7
K	BIT 6	8
L	BIT 7	9
M	STROBE	1

Normalt er STROBE high (1), men når der er en karakter parat til printerens, trækkes STROBE lav (0) et kort øjeblik. Dette indlæser karakteren i printerens. I samme øjeblik sætter printerens BUSY linjen på high (1), og først når printerens er parat til at modtage en ny karakter, går BUSY igen på lav (0). Dette skift fra 1 til 0, får FLAG-bit'et til at skifte fra 0 til 1, og dette benyttes som en test af, om printerens er klar til modtagelse af en karakter eller ej. På de fleste printere vil BUSY være high, når der optræder fejl, når der mangler papir osv.

Programlistningen ser således ud:

```

155:  C000          *=  $C000
160:  C000          .OPT 08,P7
;
;PRINTERDRIVER FOR COMMODORE 64
;
200:  C000          KERNAL   =  $031A
210:  C000          DRRA    =  $DD02
220:  C000          DRRB    =  $DD03
230:  C000          PORTA   =  $DD00
240:  C000          PORTB   =  $DD01
250:  C000          FLAG2   =  $DD0D
260:  C000          FILANTAL =  $98
270:  C000          LF      =  $B8
280:  C000          SA      =  $B9
290:  C000          FA      =  $BA
300:  C000          LFTAB   =  $0259
310:  C000          FATAB   =  $0263
320:  C000          SATAB   =  $026D
330:  C000          LIST    =  $0F
340:  C000          XREG    =  $97
;
;INITIALISERING AF RUTINEN
;
380:  C000 A0 00      INITIAL LDY  #$00
390:  C002 B9 3D C0   LOOP1   LDA  TABEL,Y
400:  C005 99 1A 03      STA   KERNAL,Y
410:  C008 C8          INY
420:  C009 C0 0E          CPY   #$0E
430:  C00B D0 F5          BNE   LOOP1
;
;KLARGØRING AF PORTEN
;
470:  C00D AD 02 DD      LDA   DRRA
480:  C010 09 04          ORA   #$04
490:  C012 8D 02 DD      STA   DRRA          ;SELECT OUTPUT
500:  C015 A9 FF          LDA   #$FF
510:  C017 8D 03 DD      STA   DRRB          ;SELECT OUTPUT
520:  C01A A9 00          LDA   #$00          ;NULBYTE
530:  C01C 8D 01 DD      STA   PORTB        ;RESET AF PORT
540:  C01F 08          PHP
550:  C020 F0 0C          BEQ   CENTRON2     ;KLARGØR STROBE
;
;SKRIV ET BYTE TIL CENTRONICS PORT
;
590:  C022 08          CENTRON PHP          ;DISABLE INTERRUPTS

```



```

600: C023 78          SKI          ;FORHINDRER AT ENKELTE PRINTERE
610: C024 8D 01 DD    STA  PORTB   ;SKRIVER DOBBELT PERIODISK
620: C027 A9 10      LDA  $$10     ;MASKE FOR TEST AF BUSY-SIGNAL
630: C029 2C 0D DD    TESTBUSY BIT  FLAG2
640: C02C F0 FB      BEQ  TESTBUSY ;VENT INDTIL PRINTER ER PARAT
650: C02E AD 00 DD    CENTRON2 LDA  PORTA
660: C031 29 FB      AND  $$FB     ;AKTIVER STROBE
670: C033 8D 00 DD    STA  PORTA
680: C036 09 04      ORA  $$04
690: C038 8D 00 DD    STA  PORTA
700: C03B 28        PLP           ;GENSKAB TIDLIGERE INTERRUPT
710: C03C 60        RTS

;
;DATA FOR VEKTORERNE, DER STYRER OPERATIVSYSTEMET
;VEKTORERNE FOR OPEN, CLOSE, CHKIN, CHKOUT OG
;CHROUT FLYTTES TIL VORE RUTINER
;
770: C03D          TABEL  =  *
;
790: C03D 4B C0      .WORD OPEN
800: C03F 83 C0      .WORD CLOSE
810: C041 9B C0      .WORD CHKIN
820: C043 B2 C0      .WORD CHKOUT
830: C045 33 F3      .WORD $F333
840: C047 57 F1      .WORD $F157
850: C049 C9 C0      .WORD CHROUT
;
;OPEN-RUTINEN - TILLADER ÅBNING TIL DEVICE 4
;FORSØGES ÅBNING AF DEVICE 2 FÅS FEJLMELDING
;
900: C04B A6 B8      OPEN  LDX  LF          ;LOGISK FILNUMMER
910: C04D F0 05      BEQ  OPENFEJL
920: C04F 20 0F F3    JSR  $F30F          ;ER FILEN ALLEREDE AABEN
930: C052 D0 03      BNE  ALTOK
940: C054 4C FE F6    OPENFEJL JMP  $F6FE          ;FILE OPEN ERROR
950: C057 A6 98      ALTOK  LDX  FILANTAL
960: C059 E0 0A      CPX  $$0A          ;MAX 10 AABNE FILER
970: C05B 90 03      BCC  NOTOFLOW
980: C05D 4C FB F6    JMP  $F6FB          ;TOO MANY FILES ERROR
990: C060 E6 98      NOTOFLOW INC  FILANTAL
1000: C062 A5 B8      LDA  LF
1010: C064 9D 59 02    STA  LFTAB,X
1020: C067 A5 B9      LDA  SA
1030: C069 09 60      ORA  $$60
1040: C06B 9D 6D 02    STA  SATAB,X
1050: C06E A5 BA      LDA  FA
1060: C070 9D 63 02    STA  FATAB,X
1070: C073 C9 02      CMP  $$02          ;CHECK FOR RS-232
1080: C075 F0 09      BEQ  NOTALLOW
1090: C077 C9 04      CMP  $$04
1100: C079 D0 02      BNE  EJCENTRN
1110: C07B 18        CLC
1120: C07C 60        RTS
1130: C07D 4C 77 F3    EJCENTRN JMP  $F377          ;NORMALT VIDERE
1140: C080 4C 13 F7    NOTALLOW JMP  $F713          ;ILLEGAL DEVICE
;

```

```

;CLOSE-RUTINEN
;
1180: C083 20 14 F3 CLOSE JSR $F314 ;FIND OPTEGNELSEN
1190: C086 F0 02 BEQ CLOSING
1200: C088 18 CLC
1210: C089 60 RTS ;FILEN ER ALLEREDE LUKKET
1220: C08A 20 1F F3 CLOSING JSR $F31F ;FIND FILPARAMETRENE
1230: C08D 8A TXA
1240: C08E 48 PHA
1250: C08F A5 BA LDA FA ;HENT DEVICE-NUMMER
1260: C091 C9 04 CMP #$04 ;ER DET VORES
1270: C093 F0 03 BEQ CLOSEOUR
1280: C095 4C 9D F2 JMP $F29D ;CLOSE ANDRE DEVICES
1290: C098 4C F1 F2 CLOSEOUR JMP $F2F1 ;CLOSE VORT DEVICE
;
;CHECK FORSØG PAA LÆSNING (ULOVLTGT)
;
1330: C09B 20 0F F3 CHKIN JSR $F30F
1340: C09E F0 03 BEQ FILEOPEN
1350: C0A0 4C 01 F7 JMP $F701 ;FILE NOT OPEN ERROR
1360: C0A3 20 1F F3 FILEOPEN JSR $F31F
1370: C0A6 A5 BA LDA FA ;FIND DEVICE
1380: C0A8 C9 04 CMP #$04 ;ER DET VORES
1390: C0AA F0 03 BEQ OURDEV
1400: C0AC 4C 19 F2 JMP $F219 ;ANDET DEVICE
1410: C0AF 4C 0A F7 OURDEV JMP $F70A ;NOT INPUT FILE ERROR
;
;KLARGØR FOR SKRIVNING AF KARAKTERER
;
1450: C0B2 20 1F F3 CHKOUT JSR $F31F
1460: C0B5 F0 03 BEQ CHKOPEN
1470: C0B7 4C 01 F7 JMP $F701 ;FILE NOT OPEN ERROR
1480: C0BA 20 1F F3 CHKOPEN JSR $F31F
1490: C0BD A5 BA LDA FA ;FIND DEVICE
1500: C0BF C9 04 CMP #$04 ;ER DET VORES
1510: C0C1 D0 03 BNE NOTOURS
1520: C0C3 20 75 F2 JSR $F275 ;OUR DEVICE CLEAR
1530: C0C6 20 5B F2 NOTOURS JSR $F25B ;ANDET DEVICE
;
;DEN EGENTLIGE UDLÆSNINGSROUTINE
;
1570: C0C9 48 CHROUT PHA
1580: C0CA A5 9A LDA $9A ;CHECK DEVICE-NR.
1590: C0CC C9 04 CMP #$04
1600: C0CE F0 03 BEQ CENTRNUD
1610: C0D0 4C CD F1 JMP $F1CD ;ANDET DEVICE
1620: C0D3 68 CENTRNUD PLA
1630: C0D4 48 PHA
1640: C0D5 86 97 STX XREG
1650: C0D7 29 7F AND #$7F ;CHECK FOR KONTROLKARAKTERER
1660: C0D9 C9 20 CMP #$20
1670: C0DB B0 34 BCS ALMTEKST
;
;DET DREJER SIG OM EN KONTROLKARAKTER
;
1710: C0DD 68 PLA

```

```

1720: C0DE 48          PHA
1730: C0DF 10 03      BPL UNDER128
1740: C0E1 38          SEC
1750: C0E2 E9 60      SBC $$60          ;KONVERTER TIL ASCII 32-63
1760: C0E4 AA          UNDER128 TAX
1770: C0E5 BD 86 C1   LDA KONTROL,X ;OPSLAG I TABEL
1780: C0E8 10 0E      BPL ALTIDUD ;KARAKTERER MED BIT 7 SAT SKAL
1790: C0EA 24 0F      BIT LIST ;CHECKES NÆRMERE
1800: C0EC 30 1D      BMI LISTING ;ER DER TALE OM LISTPRINT
1810: C0EE E0 12      CPX $$12
1820: C0F0 F0 0E      BEQ REVERSON
1830: C0F2 E0 32      CPX $$32
1840: C0F4 F0 0D      BEQ REVERSOFF
1850: C0F6 29 7F      AND $$7F ;FJERN BIT 7
1860: C0F8 20 22 C0 ALTIDUD JSR CENTRON ;SEND KONTROLKARAKTER TIL PRINTER
1870: C0FB 68          EXIT PLA
1880: C0FC A6 97      LDX XREG
1890: C0FE 18          CLC
1900: C0FF 60          RTS ;FINISHED

;
;SET/RESET REVERSE
;
1940: C100 A9 80      REVERSON LDA $$80
1950: C102 2C          .BYTE $2C
1960: C103 A9 00      REVERSOFF LDA $$00
1970: C105 8D 83 C1   STA REVERSE
1980: C108 4C FB C0   JMP EXIT

;
;KONVERTERING AF KONTROLKODE TIL REVERSE KARAKTERKODE
;
2020: C10B 18          LISTING CLC
2030: C10C 8A          TXA
2040: C10D 69 C0      ADC $$C0
2050: C10F D0 24      BNE CONVERT
2060: C111 38          ALMTEKST SEC
2070: C112 E9 20      SBC $$20
2080: C114 10 1C      BPL ALLDONE
2090: C116 C9 DF      CMP $$DF
2100: C118 D0 04      BNE CHCKMORE
2110: C11A A9 5E      LDA $$5E ;PI
2120: C11C D0 14      BNE ALLDONE
2130: C11E C9 A0      CHCKMORE CMP $$A0
2140: C120 90 0D      BCC SUB20
2150: C122 C9 C0      CMP $$C0
2160: C124 90 04      BCC SUB60
2170: C126 E9 40      SBC $$40
2180: C128 D0 08      BNE ALLDONE
2190: C12A 38          SUB60 SEC
2200: C12B E9 60      SBC $$60
2210: C12D D0 03      BNE ALLDONE
2220: C12F 38          SUB20 SEC
2230: C130 E9 20      SBC $$20
2240: C132 0D 83 C1 ALTIDUD ORA REVERSE

;
;ASCII-KODERNE ER NU KONVERTEREDE SÅLEDES AT
;

```

# Kommunikation med printere og andet udstyr

```

; 32-127 = 0-95
;160-191 = 96-127
;192-223 = 64-95
;224-254 = 96-126
;
;
; REVERSE ON FLYTTER KODERNE TIL 128-255
;-----
; FØRST UDSENDES PRINTERKOMMANDOEN FOR GRAFIK
;
2370: C135 48      CONVERT PHA
2380: C136 A2 00    LDX  #00
2390: C138 BD 7B C1 FORTSAET LDA KOMMANDO,X
2400: C13B 20 22 C0 JSR  CENTRON
2410: C13E E8      INX
2420: C13F EC 7A C1 CPX  KOMBYTES
2430: C142 D0 F4    BNE  FORTSAET
;
; STARTADRESSEN PAA KARAKTERBYTES BEREKNES
;
2470: C144 AD 84 C1 LDA  STARTADR
2480: C147 85 FB    STA  $FB
2490: C149 AD 85 C1 LDA  STARTADR+1
2500: C14C 85 FC    STA  $FC
2510: C14E 68      PLA
2520: C14F 2A      ROL
2530: C150 2A      ROL
2540: C151 2A      ROL
2550: C152 48      PHA
2560: C153 2A      ROL
2570: C154 29 07    AND  #07
2580: C156 18      CLC
2590: C157 65 FC    ADC  $FC
2600: C159 85 FC    STA  $FC
2610: C15B 68      PLA
2620: C15C 29 F8    AND  #$F8
2630: C15E 18      CLC
2640: C15F 65 FB    ADC  $FB
2650: C161 85 FB    STA  $FB
2660: C163 90 02    BCC  OUTPUT
2670: C165 E6 FC    INC  $FC
2680: C167 98      OUTPUT TYA
2690: C168 48      PHA
2700: C169 A0 00    LDY  #00
2710: C16B B1 FB    NAESTE LDA  ($FB),Y
2720: C16D 20 22 C0 JSR  CENTRON
2730: C170 C8      INY
2740: C171 C0 08    CPY  #08
2750: C173 D0 F6    BNE  NAESTE
2760: C175 68      PLA
2770: C176 A8      TAY
2780: C177 4C FB C0 JMP  EXIT
;
; FORSKELLIGE TABELVAERDIER
;
2820: C17A 00      KOMBYTES .BYTE 0 ;ANTAL BYTES I GRAFIKKOMMANDO
2830: C17B 00 00 00 KOMMANDO .BYTE 0,0,0,0,0,0,0,0 ;MAX 8 BYTES

```

```

2840: C183 00      REVERSE .BYTE 00      ;PLADS TIL REVERSE-FLAG
2850: C184 C6 C1   STARTADR .WORD KARAKTER
;
;KODER MED BIT 7 SAT SVARER TIL KONTROLKARAKTER
;KODER MED EN VÆRDI (BORSET FRA BIT7) UDLØSER EN VIRKNING
;HOS PRINTEREN - CLR OG HOME = FORM FEED
;CURSOR RIGHT = SPACE, DOWN = LINE-FEED, LEFT=BACK-SPACE
;
2915: C186          KONTROL =      *
;
2920: C186 00 00 00 .BYTE $00,$00,$00 ;0-2
2930: C189 00 00 80 .BYTE $00,$00,$80 ;3-5
2940: C18C 00 00 80 .BYTE $00,$00,$80 ;6-8
2950: C18F 80 00 00 .BYTE $80,$00,$00 ;9-11
2960: C192 00 0D 80 .BYTE $00,$0D,$80 ;12-14
2970: C195 00 00 8A .BYTE $00,$00,$8A ;CURSOR DOWN UDLØSER LINE-FEED
2980: C198 80 8C 80 .BYTE $80,$8C,$80 ;HOME UDLØSER FORM-FEED
2990: C19B 00 00 00 .BYTE $00,$00,$00 ;21-23
3000: C19E 00 00 00 .BYTE $00,$00,$00 ;24-26
3010: C1A1 00 00 A0 .BYTE $00,$00,$A0 ;27-29 - CURSOR RIGHT=SPACE
3020: C1A4 80 80 80 .BYTE $80,$80 ;30-31
3030: C1A6 00 80 00 .BYTE $00,$80,$00 ;128-130
3040: C1A9 00 00 80 .BYTE $00,$00,$80 ;131-133
3050: C1AC 80 80 80 .BYTE $80,$80,$80 ;134-136
3060: C1AF 80 80 80 .BYTE $80,$80,$80 ;137-139
3070: C1B2 80 8D 80 .BYTE $80,$8D,$80 ;140-142
3080: C1B5 00 80 80 .BYTE $00,$80,$80 ;143-145
3090: C1B8 80 8C 80 .BYTE $80,$8C,$80 ;CLR=FORM-FEED
3100: C1BB 80 80 80 .BYTE $80,$80,$80 ;149-151
3110: C1BE 80 80 80 .BYTE $80,$80,$80 ;152-154
3120: C1C1 80 80 88 .BYTE $80,$80,$88 ;CURSOR LEFT=BACK-SPACE
3130: C1C4 80 80 80 .BYTE $80,$80 ;158-159
;
;START PAA KARAKTERTABELLEN - HVER KARAKTER HAR 8 BYTES
;
3170: C1C6          KARAKTER =      *
;
3190: C1C6 00 00 00 .BYTE 0,0,0,0,0,0,0,0 ;1. KARAKTER
;
;OSV. 255 GANGE
;

```

Uanset om printeren skal have mindre end 8 bytes eller 8 bytes for at "tegne" en karakter, skal hver karakter optræde i tabellen med 8 bytes. Antallet af bytes, der sendes til printeren bestemmes i linje 2740, som her har værdien 8. De aktuelle værdier afhænger af den benyttede printer, og må derfor fremstilles "i hånden".

Den relokerbare udgave af programmet er listet herunder. Husk blot, at alfabetet skal begynde ved startadressen+453 bytes, hvis fremgangsmåden med LOAD til en absolut adresse anvendes. Den kode, der kobler printeren til at skrive grafiktegn, skal starte på startadressen+378 bytes, og det antal tegn, der er indeholdt i denne kommando (maksimalt 8) skal POKE's til startadresse +377.

```

10000 DATA 49152:REM STARTADRESSE
49152 DATA 160,000,185,-49213,153,026,003
49160 DATA 200,192,014,208,245,173,002,221
49168 DATA 009,004,141,002,221,169,255,141
49176 DATA 003,221,169,000,141,001,221,008
49184 DATA 240,012,008,120,141,001,221,169
49192 DATA 016,044,013,221,240,251,173,000
49200 DATA 221,041,251,141,000,221,009,004
49208 DATA 141,000,221,040,096,-49227,-49283
49216 DATA -49307,-49330,051,243,087
49224 DATA 241,-49353,166,184,240,005,032
49232 DATA 015,243,208,003,076,254,246,166
49240 DATA 152,224,010,144,003,076,251,246
49248 DATA 230,152,165,184,157,089,002,165
49256 DATA 185,009,096,157,109,002,165,186
49264 DATA 157,099,002,201,002,240,009,201
49272 DATA 004,208,002,024,096,076,119,243
49280 DATA 076,019,247,032,020,243,240,002
49288 DATA 024,096,032,031,243,138,072,165
49296 DATA 186,201,004,240,003,076,157,242
49304 DATA 076,241,242,032,015,243,240,003
49312 DATA 076,001,247,032,031,243,165,186
49320 DATA 201,004,240,003,076,025,242,076
49328 DATA 010,247,032,031,243,240,003,076
49336 DATA 001,247,032,031,243,165,186,201
49344 DATA 004,208,003,032,117,242,032,091
49352 DATA 242,072,165,154,201,004,240,003
49360 DATA 076,205,241,104,072,134,151,041
49368 DATA 127,201,032,176,052,104,072,016
49376 DATA 003,056,233,096,170,189,-49542
49384 DATA 016,014,036,015,048,029,224,018
49392 DATA 240,014,224,050,240,013,041,127
49400 DATA 032,-49186,104,166,151,024,096
49408 DATA 169,128,044,169,000,141,-49539
49416 DATA 076,251,192,024,138,105,192,208
49424 DATA 036,056,233,032,016,028,201,223
49432 DATA 208,004,169,094,208,020,201,160
49440 DATA 144,013,201,192,144,004,233,064
49448 DATA 208,008,056,233,096,208,003,056
49456 DATA 233,032,013,-49539,072,162,000
49464 DATA 189,-49531,032,-49186,232,236
49472 DATA -49530,208,244,173,-49540,133
49480 DATA 251,173,-49541,133,252,104,042
49488 DATA 042,042,072,042,041,007,024,101
49496 DATA 252,133,252,104,041,248,024,101
49504 DATA 251,133,251,144,002,230,252,152
49512 DATA 072,160,000,177,251,032,-49186
49520 DATA 200,192,008,208,246,104,168,076
49528 DATA -49403,000,000,000,000,000,000,000
49536 DATA 000,000,000,000,-49606,000,000
49544 DATA 000,000,000,128,000,000,128,128
49552 DATA 000,000,000,013,128,000,000,138
49560 DATA 128,140,128,000,000,000,000,000
49568 DATA 000,000,000,160,128,128,000,128
49576 DATA 000,000,000,128,128,128,128,128

```

```
49584 DATA 128,128,128,141,128,000,128,128
49592 DATA 128,140,128,128,128,128,128,128
49600 DATA 128,128,128,136,128,128
49608 REM printerkarakterer (256x8 bytes)
49608 DATA 999
```

## Kapitel 10

### Commodore 64 grafik

Commodore 64 er benådet med virkeligt fornemme grafikmuligheder - et område, som BASIC'en overhovedet ikke kender noget til. Og det gør jo ikke ligefrem livet lettere for de brugere, der ikke er yndere af maskinkode. Software industrien jubler naturligvis over denne grove forsømmelse fra Commodores side, men det er jo en ringe trøst for dem, der skal punge ud med nogle flere halvstore sedler, blot for at tegne en linje på skærmen.

#### 10.1 Billedskærmens muligheder

Computeren har to hovedtilstande - nemlig tekst og grafik. Tekst kan optræde i tre hovedformer - normal, *multicolor* og *extended color*. Grafikken findes i to hovedformer - normal *højopløsningsgrafik* og *multicolor højopløsningsgrafik*. Begge former kan udnytte *spritegrafik*, der består af op til 8 forskellige *movable object blocks* - også kaldet *sprites*.

I kapitlet Commodore TASTATUR og SKÆRM har vi allerede behandlet, hvordan vi kan flytte skærmen og hvordan vi definerer vort eget karaktersæt. De øvrige tekstmuligheder, samt styring af farven, vil behandle først i dette kapitel.

##### 10.1.1 Farvehukommelse for tekst

Skærmen kan opdeles i tre hovedområder - rammen, baggrunden og forgrunden. Rammen er det område af skærmen, der ligger udenfor det egentlige tekstområde, og rammens farve bestemmes af de fire nederste bits på adresse 53281. De fire øverste bits har ingen betydning. Rammens farve bestemmes med kommandoen:

POKE 53280,farve

På tilsvarende måde fastlægges skærmens baggrundsfarve:

POKE 53281,farve



Også her kan enhver værdi mellem 0 og 255 benyttes, men kun de nederste 4 bits har nogen virkning.

Forgrundsfarven - farven på hvert bit, der har værdien 1 i de enkelte tegn - fastlægges af et særligt hukommelsesområde, der har samme størrelse som skærmens karakterområde. Mens skærmens karakterhukommelse kan flyttes frit i computerens hukommelse, er farvehukommelsen placeret fast i området 55296 til og med 56319 (D800-DBFF), hvorefter kun de 1000 nederste bytes udnyttes. Hver enkelt position i farvehukommelsen svarer således til en karakterposition på skærmen, og farven på den pågældende karakter bestemmes af den farvekode, der befinder sig på tilsvarende position i farvehukommelsen. Ligesom for rammen og baggrunden har kun de nederste 4 bits i hvert byte i farvehukommelsen betydning.

I alt er det muligt at benytte 16 farver samtidigt på skærmen, svarende til farbekoderne 0 til 15. De korresponderende farver er:

0	SORT	4	PURPUR	8	ORANGE	12	MELLEMGRA
1	HVID	5	GRØN	9	BRUN	13	LYSEGRØN
2	RØD	6	BLÅ	10	LYSERØD	14	LYSEBLÅ
3	CYAN	7	GUL	11	MØRKGRA	15	LYSEGRA

### 10.1.2 Multicolor tekst

Indkobles multicolor tekst ændres betydningen af de enkelte punkter på skærmen. I stedet for at hver bitværdi afgør, om for- eller baggrundsfarven vises, er det nu to bits, der fastlægger farverne. Det medfører, at hvert bogstav i praksis består af 4x8 punkter (4 vandret), i stedet for 8x8 punkter. Hvert af de 4 punkters farve fastlægges af en to-bit-kombination:

BITS	FARVEREGISTER	ADRESSE
00	BAGGRUNDSFARVE	53281
01	BAGGRUND 1	53282
10	BAGGRUND 2	53283
11	FORGRUNDSFARVE	FARVEHUK.

Bemærk, at farvehukommelsen har en ganske særlig betydning i multicolor. De nederste 3 bits bestemmer nu farven (farvekode 0 til 7), mens bit 4 afgør, om det tilsvarende tegn skal vises normalt eller i multicolor. Er det fjerde bit lig med en (f.eks. værdien 15) vil det korresponderende tegn blive tolket som et multicolor tegn, og er værdien nul (f.eks. værdien 7), opfattes tegnet normalt - i begge tilfælde er forgrundsfarven gul! Dvs. at det er muligt at blande normale tegn og multicolor tegn frit på skærmen. Multicolor muligheden aktiveres med kommandoen:

POKE 53270,PEEK(53270)OR 16

Multicolor tekst tillader brugen af op til 4 forskellige farver i hvert bogstavfelt.

### 10.1.3 Extended color tekst

Extended color tekst tillader kun anvendelsen af 64 forskellige tegn. Når ekstended color indkobles med kommandoen:

POKE 53265,PEEK(53265)OR 64

benyttes kun de nederste 64 tegn i karakter ROM'en. De øverste to bits i karakterkoden (POKE-værdien - se tabellen bag i bogen) bestemmer hvilken af fire baggrundsfarver, der skal benyttes. Forgrundsfarven bestemmes stadig af farvehukommelsen. Baggrundsfarverne bestemmes således:

SKÆRMKODE	BAGGRUNDSFARVE	REGISTER
0- 63	BAGGRUND 0	53281
64-127	BAGGRUND 1	53282
128-191	BAGGRUND 2	53283
192-255	BAGGRUND 3	53284

Extended color tekst er især velegnet, når man ønsker at kunne skifte baggrundsfarve for dele af skærmen i tekstmeddelelser, uden at resten af baggrunden forandres.

Skrives der til skærmen med normale PRINT sætninger, kan følgende ASCII karaktertabel anvendes til styring af baggrundsfarven.

FARVE 1	FARVE 2	FARVE 3	FARVE 4
32-63	96-127	32-63	96-127
64-95	160-191	64-95	160-191
REVERSE OFF		REVERSE ON	

Ekstended color tekst tillader brugen af to farver i hvert karakterfelt, og tegnene defineres på akkurat samme måde, som ved normal tekst - blot omfatter alfabetet kun 64 tegn, eller 512 bytes (64\*8).

### 10.2 Programmering af sprites

Sprites er særlige, brugerdefinerede blokke, som kan bevæges frit på skærmen, uden iøvrigt at påvirke skærmens indhold. Der kan eksistere op til 8 sprites samtidigt på skærmen, og disse kan individuelt fastlægges i to

bevægelsesplaner - nemlig bag "bogstaverne" eller foran. Sprites nummereres fra 0 til 7, og når to eller flere sprites skal bevæges forbi hinanden, vil sprite 0 altid være forrest, mens sprite 7 vil dækkes af de syv andre. Sprite 1 kan dække sprite 2 til 7 osv.

### 10.2.1 Fremstilling af sprites

En sprite består af 24 punkter vandret og 21 punkter lodret. Ialt 63 (24/8\*21) bytes. Hver sprite skal skabes af brugeren og kan placeres overalt i RAM, blot kræves at de skal befinde sig i samme grundområde, som tekst-skærmen (se side 19), og at startadressen for hver enkelt sprite skal være delelig med 64. Sprite formatet er:

BYTE	0	1	2
0	11111111	11111111	11111111
+3	11111111	11111111	11111111
+6	11111111	11111111	11111111
+9	11111111	11111111	11111111
osv. for byte 12 til 53			
+54	11111111	11111111	11111111
+57	11111111	11111111	11111111
+60	11111111	11111111	11111111

Hvert bit med værdien 0 udgør den gennemsligtige del af en sprite - er bitværdien 1, vises punktet i sprites farve.

Når vi har fremstillet en sprite og placeret denne i hukommelsen, skal computeren have at vide, hvor den kan finde "tegningen". Til det formål, er det reserveret et særligt område i de øverste 8 bytes af den 1024 byte blok, der anvendes som tegn-hukommelse. Er skærmen placeret, som den plejer at være det, er det adresserne 2040 til 2047, som fortæller computeren, hvor i hukommelsen den finder sprite 0 til 7. Adressen meddeles således til computeren:

**POKE skærmstart+1018+spitenr,startadresse/64**

Nu ved computeren, hvor den kan finde de(n) de sprites, vi har fremstillet. Men der skal mere til, før computeren bekymrer sig om at vise spriten på skærmen. Først skal de(n) ønskede sprite(s) tændes - det sker med kommandoen:

**POKE 53269,2^spitenr**

som sætter et bit, svarende til den sprite, vi ønsker at tænde. Ønsker vi at afbryde en sprite, benyttes følgende formel:

**POKE 53269,PEEK(53269) AND (255-2^spritennr)**

F.eks. vil værdien 255 tænde alle sprites, værdien 1 tænder kun sprite 0, og værdien 0 slukker for alle sprites.

Inden vi tænder for en sprite, kan det være meget praktisk at bestemme farven:

**POKE 53287+spritennr,farvekode**

og ligeledes om vi ønsker, at spriten skal "dække" den tekst den møder eller ej. Skal spriten dække teksten, benyttes kommandoen:

**POKE 53275,2^spritennr**

og skal teksten dominere over en sprite, der passerer forbi, benyttes kommandoen:

**POKE 53275,PEEK(53275) AND (255-2^spritennr)**

Nu mangler vi kun at placere de(n) ønskede sprite(s) på skærmen. Vi skal bestemme både X-koordinaterne og Y-koordinaterne. X-koordinatet kan have en værdi fra 0 til 511, og Y-koordinatet en værdi fra 0-255. Sammenligner vi med skærmens format på 320x200 punkter er der tale om ikke så lidt af en afvigelse. Forståelsen lettes dog betydeligt, hvis vi forestiller os, at sprites optræder på en særlig skærm, hvoraf tekstskærmen kun udgør en mindre del. Tekstskærmens øverste, venstre hjørne har da koordinaterne 50,24 i spriteskærmen, og det nederste, højre hjørne har koordinaterne 344,250.

Placeringen af en sprite bestemmes således (øverste, venstre hjørne):

```
100 X1=X-akse/256
110 X2=X-akse AND 255
120 POKE 53248+2*spritennr,X1
130 POKE 53249+2*spritennr,Y
140 POKE 53264,X1*2^spritennr
```

Det eneste vi skal gøre for at flytte rundt på en eller flere sprite, er at POKE'e nye værdier for X og Y akser - mere skal der ikke til. Det er altså en meget enkel sag at få fart på, selv når man programmerer i BASIC!

Mulighederne er dog langt fra udtømte. Vi kan f.eks. udvide størrelsen på en sprite - både vandret, lodret eller begge dele - uden at vi skal ændre på indholdet i RAM'en! Ønsker vi dobbelt størrelse i vandret retning, klares det således:

**POKE 53277,2^spritennr**

og normalstørrelsen indkobles igen med:

**POKE 53277,PEEK(53277) AND (255-2^spritennr)**

Gælder sagen at udvide spriten i lodret retning, udføres kommandoen:

POKE 53271,2^*sprit*enr

og tilbagekobling til normal størrelse sker med:

POKE 53271,PEEK(53271) AND (255-2^*sprit*enr)

Fremstiller vi spil - f.eks. et stock-car race - kan det være en god ting at vide, om der er køreere, der er bræaaaget sammen, eller ej. Vi kan naturligvis undersøge hver enkelt kørers position, hver gang vi flytter "bilen" (spriten), men det kan være en omstændelig og dermed tidskrævende proces. Skal der være fart over feltet - i BASIC - er der ganske enkelt ikke tid til den slags luksus! Der findes dog en enkel og elegant udvej for check af, om to sprites er kollideret eller ej. VIC-chip'en rummer nemlig to registre, som viser, om der er sket en kollision eller ej. Registrerne kan ikke vise, hvad der er kollideret med hvem (hvis mere end to sprites benyttes), eller hvor på skærmen kollisionen er sket, men er der ikke sket nogen kollision, behøver vi heller ikke lave nogen beregninger - altså sparer vi en masse tid! En kollision mellem to sprites, kan vi konstatere således:

SK=PEEK(53278)

IF SK THEN GOSUB *hvem kolliderede*

Hver gang vi læser denne adresse, nulstilles kollisions-bit'et, derfor er vi nødt til at læse værdien ind i en variabel. Ved en kollision sættes de bits, der svarer til de sprites, der har været indvolverede i en kollision. Har vi afmærket banen på skærmen, kan det også være interessant at undersøge, hvem der ikke kan kontrollere sit køretøj ordentligt - dvs. en kollision mellem *tekst* og *sprite*. Sker dette vil bit'et for den eller de berørte sprites blive sat, og vi behøver kun at udføre sætningen:

SS=PEEK(53279)

IF SS THEN GOSUB *hvem skred i svinget*

for at finde ud af, om der er noget, vort program skal kigge lidt nærmere på.

Kollisionsbit'ene tændes når følgende betingelser er opfyldt:

SKÆRM TYPE - SKÆRMBIT KOLLIDERER MED SPRITEBIT		SPRITETYPE	
NORMAL	1	1	NORMAL
EXTENDED	1	1	NORMAL
MULTICOLOR	10	1	NORMAL
MULTICOLOR	11	1	NORMAL
NORMAL	1	01/10/11	MULTICOLOR
EXTENDED	1	01/10/11	MULTICOLOR
MULTICOLOR	10	01/10/11	MULTICOLOR
MULTICOLOR	11	01/10/11	MULTICOLOR

Uanset *sprite*-type tændes *sprite* til *sprite* kollisionsbit'et, når de

ikke-gennemsigtige dele af to sprites "overlapper" - nærkontakt med "lakken" imellem er ikke nok!

### 10.2.2 Multicolor spritegrafik

Ligesom tekstskærmen kan også sprites skiftes til multicolor. Denne funktion kan indkobles uafhængigt for hver sprite, der da kan bestå af op til tre forskellige farver - plus en gennemsigtig del. I multicolor tilstanden vil de to ekstra farver dog være identiske for alle multicolor sprites. Multicolor indkobles for hver enkelt sprite med kommandoen:

POKE 53276,2^*spritennr*

og tilbagekobling til normal sprite sker således:

POKE 53276,PEEK(53276) AND (255-2^*spritennr*)

De to ekstra farver bestemmes med kommandoerne:

POKE 53285,*farve 1*

POKE 53286,*farve 2*

Ligesom i multicolor tekst ændres betydningen af de enkelte bits for multicolor sprites. I normale sprites afgør et bit, om punktet har en farve eller er "gennemsigtigt". I multicolor sprites anvendes nu 2 bits til farveinformationerne, således at en multicolor sprite består af 12x21 synlige punkter (hvert punkt i vandret retning er nu dobbelt så bredt). Betydningen af de fire bitkombinationer er:

BITS	FARVE	ADRESSE
00	gennemsigtig	----
01	multifarve 1	53285
10	spritefarve	53287-53294
11	multifarve 2	53286

Sprite multicolor indkobling påvirker ikke tekst eller grafikskærmens funktion på nogen måde.

### 10.2.3 Normal højopløsningsgrafik

Tekstskærmen på Commodore 64 rummer mange interessante muligheder, men mange opgaver kan kun løses i højopløsningsgrafik. Når denne skærmfunktion aktiveres, ændres skærmens opbygning fra 25 linjer, hver med 40 tegn, til 25 linjer med 40 blokke, der hver består af 8 bytes. Hver blok i højopløsning har samme format, som et tegn i den normale tekstskærm. Dvs. at højopløsningskærmen indeholder 8x25 punkter i lodret retning og 40x8

punkter (40x8 bits) i det vandrette plan - eller sagt på en anden måde: skærmen har en opløsning på 320x200 punkter. Hvert enkelt af disse punkter kan tændes eller slukkes efter ønske, og det åbner jo helt nye muligheder.

Grafik skærmen optager ialt 8000 (25x40x8) bytes af hukommelsen, og da også denne skal placeres indenfor det 16K grundområde (se side 19), som VIC-chip'en kan arbejde i. Indenfor dette område fastlægges skærmens placering således:

VÆRDI	STARTADRESSE
0	0
8	8196

Denne værdi indlæses i stedet for alfabetets placering i VIC-registeret på adresse 53272. Der er intet i vejen for at placere højopløsnings-skærmen under ROM, blot skal man huske, at skærmen ikke må placeres i RAM i området 0-7999, og at skærmen vil kollidere med karakter ROM'en, hvis den placeres i området 32768 til 40767. Højopløsnings-skærmens placering i grundområdet fastlægges således:

POKE 53272,(PEEK(53272)AND 240)OR værdi

Og højopløsnings-skærmen aktiveres således:

POKE 53265,PEEK(53265)OR 32

Det er ikke muligt at skrive tekster til en højopløsnings-skærm, så det vil altid være en god ide, at udkoble højopløsning, inden man afslutter et program. Det sker således:

POKE 53265,PEEK(53265)AND 223

Højopløsning udkobles også, når man trykker på RUN/RESTORE.

En speciel detalje ved højopløsnings-skærmen er farvevalget. Den normale farvehukommelse benyttes overhovedet ikke. I stedet anvendes tekst-skærmen til lagring af farveinformationerne. Hvert byte i tekstskærmen indeholder farvekoderne svarende til en 8x8 blok på højopløsnings-skærmen. Farvekoderne for hver blok bestemmes således:

POKE skærmstart+INT(grafikadresse/8),16\*(1-bit farve)+(0-bit farve)

Der er intet i vejen for at benytte 16 farver på grafik-skærmen, blot skal man huske, at der kun kan eksistere 2 farve indefor hver 8x8 blok. Følgende lille eksempel viser, hvordan det kan se ud:

```
100 POKE 56,31:CLR:REM SET TOP OF BASIC
110 POKE 53265,PEEK(53265)OR 32:REM HIRES ON
120 POKE 53272,(PEEK(53272)AND 240)OR 8
130 FOR N=1024 TO 2023
140 : POKE N,N AND 255:REM SET FARVE
150 NEXT N
160 FOR N=8192 TO 16191
170 : POKE N,N AND 255
180 NEXT N
190 POKE 53265,PEEK(53265)AND 223:REM TEKST ON
200 END
```

Eksemplet er udformet, så du tydeligt kan se, hvad der sker.

Ønskes udskrift af "flotte" kurver m.m. i højopløsningsgrafik, kan følgende rutine benyttes (se også næste afsnit):

```
100 REM F0 = 0-BIT FARVE
110 REM F1 = 1-BIT FARVE
120 REM SK = TEKSTSKAERM
130 REM GR = GRAFIKSKAERM
140 REM X = 0-319
150 REM Y = 0-199
160 REM P = 0 ELLER 1 (OFF/ON)
170 POKE SK+40*INT(Y/8)+INT(X/8),F1*16+F0
180 AD=GR+320*INT(Y/8)+8*INT(X/8)+Y AND 7
190 BI=P*2^(X AND 7)
200 MS=255-2^(X AND 7)
210 POKE AD,(PEEK(AD)AND MS)OR BI
220 RETURN
```

#### 10.2.4 Multicolor højopløsningsgrafik

Også i højopløsning kan multicolor indkobles. Formatet på skærmen er eksakt det samme, men nu svarer 2 bits til eet punkt på skærmen. Dvs. at X-aksen kun indeholder 160 punkter, der hver er dobbelt så brede, som i normal grafik. Hver blok på 4x8 synlige punkter (8 bits x 8 bytes) kan nu rumme fire forskellige farver på en gang. Multicolor grafik indkobles ved at tænde for multicolor bit'et, når højopløsning er indkoblet:

```
POKE 53270,PEEK(53270)OR 16
```

og den følgende kommando kobler tilbage til normal højopløsning igen:

```
POKE 53270,PEEK(53270)AND 239
```

Vær opmærksom på, at det ikke er muligt at "blande" multicolor og normal grafik, som det er ved tekst.

Også i multicolor grafik udnyttes tekstskaermen til lagring af



farveinformationer, men yderligere benyttes den normale farvehukommelse, samt det normale baggrundsfarve-register i VIC-chip'en. Bit kombinationerne har følgende betydning:

BITS	FARVEINFORMATIONER	ADRESSER	
00	baggrundsfarve	53281	0
01	tekstskærm - bit 4-7		1
10	tekstskærm - bit 0-3		2
11	farvehukommelse		3

I det følgende præsenteres en maskinkoderutine, der kan sætte en af de fire farvemuligheder i enhver blok på skærmen. Dette styres af kommandoen SYS, som i dette tilfælde har et meget specielt format:

**SYS programadresse,X,Y,M**

hvor X er i området 0-159, Y i området 0-199 og M er en af de 4 farvemuligheder, der eksisterer indenfor hver blok. X og Y koordinaterne skal altid specificeres, ellers fås fejlmeldingen SYNTAX ERROR. Ligger X og Y udenfor det legale område, fås fejlmeldingen ILLEGAL QUANTITY. Hvis M-værdien ikke specificeres, anvendes sidst benyttede M-kode - første gang er koden 0. M skal altid være mindre end 255, men programmet finder iøvrigt selv en værdi i området 0 til 3 (AND 3). Derudover er eneste forudsætning for brugen er, at kommandoen:

**POKE 2,startadresse grafikskærm/256**

før SYS kommandoen udføres første gang. Udover adresse 2 benytter denne "PLOT"-kommando også page nul adresserne 253 og 254. Rutinen kan placeres overalt i hukommelsen - udenfor ROM-områderne. Grafik-skærmen kan uden problemer placeres under ROM, som det efterfølgende eksempel viser.

```

510:  C000          ;
          ;          *=  $C000
          ;
530:  C000      HIRESPAGE=  $02
540:  C000      LOBYTE  =  $FD
550:  C000      HIBYTE  =  $FE
          ;
570:  C000      VALUETOX =  $B79E
580:  C000      ERROROUT =  $A437
590:  C000      YTOFPA  =  $B3A2
600:  C000      AYTOFPA =  $B391
610:  C000      FPATOARG =  $BC0C
620:  C000      FPATOAY =  $B7F7
630:  C000      FPAMULARG=  $BA2B
          ;
          ;INDLÆSNING AF PARAMETRENE
          ;
670:  C000 4C 0B C0 ADRESSE JMP DOTSET
          ;
          ;

```

```

700: C003 A2 0B SYNTAXER LDX #$0B
710: C005 2C .BYTE $2C
720: C006 A2 0E OVERFLOW LDX #$0E
730: C008 4C 37 A4 JMP ERROROUT
;
750: C00B 20 B5 C0 DOTSET JSR CHCKOMMA ;UNDESØG OM KOMMA FINDES
760: C00E D0 F3 BNE SYNTAXER ;NEJ=SYNTAX ERROR
770: C010 20 73 00 JSR $0073 ;PEG MOD NÆSTE TEKN
780: C013 20 9E B7 JSR VALUETOX
790: C016 E0 A0 CPX #$A0 ;160 ELLER DEROVER
800: C018 B0 EC BCS OVERFLOW
810: C01A 8E BC C0 STX XPARAM
820: C01D 20 B5 C0 JSR CHCKOMMA
830: C020 D0 E1 BNE SYNTAXER
840: C022 20 73 00 JSR $0073
850: C025 20 9E B7 JSR VALUETOX
860: C028 C9 C8 CMP #$C8 ;200 ELLER DEROVER
870: C02A B0 DA BCS OVERFLOW
880: C02C 8E BD C0 STX YPARAM
890: C02F 20 B5 C0 JSR CHCKOMMA
900: C032 D0 0C BNE NOCOLOR
910: C034 20 73 00 JSR $0073
920: C037 20 9E B7 JSR VALUETOX
930: C03A 8A TXA
940: C03B 29 03 AND #$03
950: C03D 8D BE C0 STA COLOR
960: C040 A5 02 NOCOLOR LDA HIRESPAGE
970: C042 85 FE STA HIBYTE
;
; KONVERTER X,Y TIL EN ADRESSE
;
1010: C044 AD BC C0 LDA XPARAM
1020: C047 48 PHA
1030: C048 29 03 AND #$03
1040: C04A 8D BF C0 STA DOTPOS
1050: C04D 68 PLA
1060: C04E 29 FC AND #$FC
1070: C050 0A ASL
1080: C051 85 FD STA LOBYTE
1090: C053 90 02 BCC NOHICOMP
1100: C055 E6 FE INC HIBYTE
1110: C057 AD BD C0 NOHICOMP LDA YPARAM
1120: C05A 48 PHA
1130: C05B 29 07 AND #$07
1140: C05D 05 FD ORA LOBYTE
1150: C05F 85 FD STA LOBYTE
1160: C061 68 PLA
1170: C062 4A LSR
1180: C063 4A LSR
1190: C064 4A LSR ;Y=INT(Y/8)
1200: C065 A8 TAY
1210: C066 20 A2 B3 JSR YTOFPA
1220: C069 20 0C BC JSR FPATOARG
1230: C06C A9 01 LDA #>320
1240: C06E A0 40 LDY #<320
1250: C070 20 91 B3 JSR AYTOFPA

```

# Commodore 64 grafik

```

1260: C073 20 2B BA      JSR  FPAMULARG      ;320*Y
1270: C076 20 F7 B7      JSR  FPATOAY
1280: C079 18             CLC
1290: C07A 65 FE          ADC  HIBYTE
1300: C07C 85 FE          STA  HIBYTE
1310: C07E 98             TYA
1320: C07F 65 FD          ADC  LOBYTE
1330: C081 85 FD          STA  LOBYTE
1340: C083 90 02          BCC  NOCARRY
1350: C085 E6 FE          INC  HIBYTE
1360: C087 AC BE C0       NOCARRY LDY  COLOR
1370: C08A AE BF C0       LDX  DOTPOS
1380: C08D BD C1 C0       LDA  MASKUD,X
1390: C090 49 FF          EOR  #$FF
1400: C092 39 C5 C0       AND  COLORBIT,Y
1410: C095 8D C0 C0       STA  MASKREG
1420: C098 A0 00          LDY  #$00
1430: C09A 08             PHP
1440: C09B 78             SEI
1450: C09C 20 AE C0       JSR  ROMSKIFT
1460: C09F B1 FD          LDA  (LOBYTE),Y
1470: C0A1 3D C1 C0       AND  MASKUD,X
1480: C0A4 0D C0 C0       ORA  MASKREG
1490: C0A7 91 FD          STA  (LOBYTE),Y
1500: C0A9 20 AE C0       JSR  ROMSKIFT
1510: C0AC 28             PLP
1520: C0AD 60             RTS

;
1540: C0AE A5 01         ROMSKIFT LDA  $01
1550: C0B0 49 02         EOR  #$02
1560: C0B2 85 01         STA  $01
1570: C0B4 60             RTS

;
;OBS KODEN MÅ IKKE PLACERES UNDER ROM
;
1610: C0B5 A0 00         CHCKOMMA LDY  #$00
1620: C0B7 A9 2C         LDA  #$2C      ;KOMMA-KARAKTER
1630: C0B9 D1 7A         CMP  ($7A),Y
1640: C0BB 60             RTS

;
1660: C0BC 00           XPARAM  .BYTE $00
1670: C0BD 00           YPARAM  .BYTE $00
1680: C0BE 00           COLOR   .BYTE $00
1690: C0BF 00           DOTPOS  .BYTE $00
1700: C0C0 00           MASKREG .BYTE $00

;
1720: C0C1           MASKUD   =    *

;
1740: C0C1 3F           .BYTE %00111111
1750: C0C2 CF           .BYTE %11001111
1760: C0C3 F3           .BYTE %11110011
1770: C0C4 FC           .BYTE %11111100

;
1790: C0C5           COLORBIT =    *

;
1810: C0C5 00           .BYTE %00000000

```

## Commodore 64 grafik

```
1820: C0C6 55          .BYTE %01010101
1830: C0C7 AA          .BYTE %10101010
1840: C0C8 FF          .BYTE %11111111
```

;

Den relokerbare udgave af programmet ser således ud:

```
49151 DATA 49152:REM STARTADRESSE
49152 DATA 076,-49163,162,011,044,162,014
49160 DATA 076,055,164,032,-49333,208,243
49168 DATA 032,115,000,032,158,183,224,160
49176 DATA 176,236,142,-49340,032,-49333
49184 DATA 208,225,032,115,000,032,158,183
49192 DATA 201,200,176,218,142,-49341,032
49200 DATA -49333,208,012,032,115,000,032
49208 DATA 158,183,138,041,003,141,-49342
49216 DATA 165,002,133,254,173,-49340,072
49224 DATA 041,003,141,-49343,104,041,252
49232 DATA 010,133,253,144,002,230,254,173
49240 DATA -49341,072,041,007,005,253,133
49248 DATA 253,104,074,074,074,168,032,162
49256 DATA 179,032,012,188,169,001,160,064
49264 DATA 032,145,179,032,043,186,032,247
49272 DATA 183,024,101,254,133,254,152,101
49280 DATA 253,133,253,144,002,230,254,172
49288 DATA -49342,174,-49343,189,-49345
49296 DATA 073,255,057,-49349,141,-49342
49304 DATA 160,000,008,120,032,-49326,177
49312 DATA 253,061,-49345,013,-49344,145
49320 DATA 253,032,-49326,040,096,165,001
49328 DATA 073,002,133,001,096,160,000,169
49336 DATA 044,209,122,096,000,000,000,000
49344 DATA 000,063,207,243,252,000,085,170
49352 DATA 255,999
```

Det følgende program illustrer anvendelsen af "PLOT" kommandoen:

```
100 IF A=0 THEN A=9:LOAD"DOTS-HEX",8,1
110 POKE 53280,0:REM SORT RAMME
120 POKE 53281,1:REM HVID BAGGRUND
130 POKE 56578,PEEK(56578)OR3:REM SIKRING AF OUTPUT
140 POKE 56576,PEEK(56576)AND252:REM BANK 3
150 POKE 53265,PEEK(53265)OR32:REM HIRES
160 POKE 53270,PEEK(53270)OR16:REM MULTICOLOR
170 POKE 53272,2*16+8
180 HI=57344:REM GRAFIK START
190 POKE 2,14*16:REM FORTÆL HVOR SKÆRMEN STARTER
200 FA=12*4096+2*1024:REM TEKSTSKÆRM
210 F3=2:F1=6:F2=0:REM FARVEVALG
```

Farveinformationerne skrives til skærm- og farvehukommelse:

```
220 FOR N=0 TO 999
230 POKE FA+N,F2*16+F3
240 POKE 55296+N,F1
250 NEXT N
```

Grafikskærmen slettes:

```
260 FOR N=HI TO HI+7999
270 POKE N,255
280 NEXT N
```

Og nu tegnes der på skærmen:

```
290 X=80:GR=12*4096
300 FOR Y=0 TO 199
310 SYS GR,X,Y,0
320 NEXT Y
330 Y=100
340 FOR X=0 TO 159
350 SYS GR,X,Y,0
360 NEXT X
370 REM
380 REM TEGN SINUS
390 REM
400 X=0
410 Y=100+90*SIN((X+A)/10)
420 SYS GR,X,Y,0
430 X=X+1:IF X>159 THEN A=A+160:X=0
440 GOTO 410
```

### 10.3 Specielle skærmfunktioner

Udover de allerede beskrevne og meget omfattende styringsmuligheder for skærmens tekst, grafik og sprites eksisterer der en række mere specielle funktioner, som primært vil være en hjælp for maskinkode programmører.

En af de mere interessante funktioner er muligheden for en "blød" *scrolling* af skærmen, også når tekst anvendes. VIC-chip'en indeholder to registre, som muliggør punktvis justering af skærmen i lodret og vandret retning. Virkningen kan kontrolleres således:

```
100 FOR N=0 TO 7
110 : POKE 53266,(PEEK(53266)AND 248)OR N
120 NEX N
140 FOR N=7 TO 0 STEP -1
150 : POKE 53266,(PEEK(53266)AND 248)OR N
160 NEX N
170 GOTO 100
```

Rutinen scroller i lodret retning. Vandret scroll kan testes med:

```
100 FOR N=0 TO 7
110 : POKE 53270,(PEEK(53270)AND 248)OR N
120 NEX N
```

```
140 FOR N=7 TO 0 STEP -1
150 : POKE 53270,(PEEK(53270)AND 248)OR N
160 NEX N
170 GOTO 100
```

Som det kan ses, opstår der "tomme" skærmfelter på skærmen, når scroll funktionerne udnyttes. For at skjule disse, kan skærmens format ændres, så disse områder skjules for brugeren. F.eks. kan skærmformatet ændres fra 40 til 38 tegn i vandret retning (kolonne 1 til 38) med denne kommando:

```
POKE 53270,PEEK(53270)OR 8
```

Tilbagekobling til de normale 40 tegn i bredden (kolonne 0 til 39) foretages således:

```
POKE 53270,PEEK(53270)AND 247
```

I Y-aksens retning sker omkobling til 24 linjer således:

```
POKE 53265,PEEK(53265)OR 8
```

Tilbagekobling til 25 linjer sker med kommandoen:

```
POKE 53265,PEEK(53265)AND 247
```

Mulighederne anvendes især i spil o.l., hvor det er af betydning at bevægelsen virker jævn. Når skærmen er scrollet punktvis helt ud i en retning, anvendes maskinkode til at flytte alle skærmens informationer et tegn til højre eller venstre, op eller ned, hvorefter det tilsvarende scroll-register stilles tilbage i udgangspositionen. I BASIC er anvendelsesmulighederne begrænsede, men man kan da benytte muligheden til at simulere en eksplosion, rystelser i en rally-bil osv.

I enkelte situationer kan man have behov for at "slukke" skærmen - f.eks. mens større dele af skærmen opdateres, eller hvis man vil opnå særlige blink-effekter i forbindelse med større billeder i højopløsnings-grafik. I disse tilfælde kan:

```
POKE 53265,PEEK(53265)AND 239
```

koble hele skærmen ud - dvs. at forgrund og baggrund m.m. får samme farve, som rammen. Indkobling sker således:

```
POKE 53265,PEEK(53265)OR 16
```

## 10.4 VIC chip'en og interrupts

Computerens grafik-kreds rummer flere muligheder for interrupts - signaler, der udløses, når en speciel situation opstår, og afbryder processorens normale arbejde. Udnyttelse af disse funktioner kræver et indgående kendskab til maskinkode, computerens hardware og KERNAL ROM'ens indhold. I BASIC programmer er værdien af disse muligheder yderst begrænsede.

Interrupts styres af et register, placeret på adresse 53274. Er et interrupt indtruffet, kan typen læses i registeret på adressen 53273. Registrerne har følgende opbygning:

### BIT BETYDNING (53273)

---

7	interrupt indtruffet
6-4	ingen betydning
3	lyspen interrupt
2	sprite-sprite kollisions interrupt
1	sprite-baggrund kollisions interrupt
0	raster compare interrupt

---

Aktiveringen af de interrupt-typer, der skal have lov at afbryde processorens normale arbejde, sker ved at sætte det pågældende bit i register 53274. Ønsker man f.eks., at der skal udløses et interrupt hver gang to sprites kolliderer, men ellers ikke andre former for interrupts, skal bit 2 i register 53274. Register 53273 indikerer, hvilke interrupts, der er indtruffet - også selvom de ikke udløser nogen form for processor interrupts. Uanset hvor mange interrupts, der indtræffer, vil bit 7 antage værdien 1, når det sker (let at checke med maskinkode instruktionen BIT).

Lyspen interrupt bit'et sættes, hver gang en gyldig værdi optræder i lyspen-registrerne. Lyspenens placering på skærmen indikeres som to værdier:

Adresse 53267	X-aksen
Adresse 53268	Y-aksen

Værdierne kan siden anvendes i et program, som udløser en eller anden funktion - f.eks. at tænde et punkt på skærmen, at springe til en udvalgt del af et programmet (valget sker ved at udpege den ønskede funktion med lyspenen).

Raster compare interrupts udløses, hver gang VIC-kredsen skal til at tegne en *raster linje* på skærmen. En *raster linje* er en vandret linje af punkter, som optræder på skærmen. I alt kender VIC-chip'en 512 (0-511) raster-linjer, hvoraf kun 200 optræder synligt på skærmen! Den synlige del af skærmen går fra raster 51 til 251. Foretages evt. ændringer udenfor dette område, kan "sne" og andre forstyrrelser i billedet helt undgås. Ligeledes er det muligt at kombinere raster interrupts med ændringer i skærmens opbygning - f.eks. kunne øverste del af skærmen være normal tekst, mens nederste del kunne skiftes til multicolor grafik. Teknikken kunne også udnyttes til at opnå mere

end 8 sprites på skærmen, forskellige tegnsæt for den øverste og nederste del af skærmen osv.

Den værdi, der skal udløse en evt. interrupt (aktiveres i interrupt registeret) fastlægges således:

```
100 RL=450:REM RASTER LINJE
110 RU=RL AND 255
120 RN=128*INT(RL/256)
130 POKE 53266,RU
140 POKE 53265,PEEK(53265)OR RN
```

I dette tilfælde placeres en evt. aktiveret interrupt på skærmlinje 450. Ønsker man at kontrollere om VIC chip'en befinder sig et bestemt sted på skærmen i "billeddannelsen" kan det gøres ved at læse de samme to registre. Adresse 53266 returnerer de nederste 8 bits og bit 7 på adressen 53265 indeholder det ottende bit for raster linje tælleren.

scroll, 24x38, interrupts etc.



## Appendix A

### Oversigt over Commodore BASIC

De følgende sider indeholder en komplet af alle BASIC-kommandoer i Commodore 64 (BASIC 2.0) og i Serie 264 BASIC (BASIC 3.5), som findes i Commodore PLUS/4 og Commodore C16. Monitoren er behandlet særskilt i kapitel 5 Commodore PLUS/4 og maskinkode, der starter side 78.

En stor del af kommandoerne i det følgende findes ikke i Commodore 64 BASIC. Dette er markeret med mærket (+4) efter TYPE-betegnelsen i oversigten - typebetegnelsen kunne f.eks. være *kommando* eller *funktion* etc.

For at lette opslags-processen for Commodore 64 ejere, følges BASIC-gennemgangen af en særlig Oversigt over Commodore 64 BASIC.

Hvor det har været muligt, er beskrivelsen af de kommandoer, som ikke findes i Commodore 64 BASIC, udvidet med et eksempel, som simulerer samme funktion eller kommando - evt. i form af en henvisning til programmer, beskrevet tidligere i bogen.

Beskrivelsen af hver BASIC-kommando omfatter følgende punkter:

#### TYPE

Beskriver, om der er tale om en funktion (numerisk eller streng), kontrolvariabel eller kommando etc.

#### FORMAT

Forklarer de(t) indtastningsformat(er) - den syntax - som funktionen eller kommandoen kræver eller tillader. F.eks. kunne formatet beskrives som: ATN(*numerisk udtryk*), hvor teksten, der er fremhævet *således*, kan erstattes af et vilkårligt valgt numerisk udtryk. Følgende betegnelser er anvendt til beskrivelse af funktions-argumenter m.m.:

- *udtryk* eller *udsagn* betyder, at ethvert udtryk indenfor definitionsområdet er tilladt. Hvor intet andet er nævnt, gælder de generelle grænser for Commodore BASIC.
- *numeriske udtryk* betyder, at alle numeriske konstanter og variable eller streng-konstanter og -variable, som indgår i funktioner, der returnerer en numerisk værdi, er tilladt.
- *streng-udtryk* betyder, at alle streng-konstanter og -variable, som indgår i funktioner, der returnerer en streng, er tilladt.

- *afstand, konstant* eller lignende udtryk betyder, at variabler, funktioner m.m. ikke må anvendes. Kun konstanter, som f.eks. tallet 100.
- *dummy-variabel, streng-variabel* eller *numerisk variabel* antyder, at en vilkårlig variabel af den nævnte type kan anvendes. Ordet *dummy* betyder, at variabelen ikke påvirker resultatet og ikke i sig selv undergår nogen forandring.

**FORMAT**-beskrivelsen kan indeholde et særskilt felt, der uddyber de benyttede parametre i kommandoer af mere omfattende natur - f.eks. grafik-kommandoen **CIRCLE**. De følgende udtryk vil finde anvendelse i denne parameterbeskrivelse:

- *farvekilde* - den virkning, grafik-kommandoen skal have på skærbilledet. Parameteren kan antage værdier fra 0 til 3. Værdierne referer til baggrundsfarve (0), og en (normal højopløsningsgrafik) eller flere forgrundsfarver (multicolor højopløsningsgrafik).
- *pixel-cursor* - også kaldet grafik-cursor. Betegnelse for det punkt, som den sidst benyttede grafik-kommando sluttede på (uanset om punktets farve blev ændret eller ej). Pixel-cursoren arbejder analogt til tekst-cursoren, blot med reference til det specificerede koordinatsystem. Pixel-cursoren er ikke synlig, som tekstcursoren er det. Pixel-cursoren kan placeres absolut eller relativt (i forhold til et allerede defineret punkt). Relativ placering sker, når koordinaterne i en grafik-kommando udstyres med fortegn. F.eks.:

**LOCATE** a,b        absolut placering i punkt a,b  
**LOCATE** +a,+b      relativ placering i punkt x+a,y+b,  
                      hvor x,y er pixel-cursorens position,  
                      inden **LOCATE**-kommandoen udføres.

**LOCATE** -a,-b      relativ placering i punkt x-a,y-b,  
                      hvor x,y er pixel-cursorens position,  
                      inden **LOCATE**-kommandoen udføres.

- *rotation* - Beskriver den vinkel, som en figur skal roteres i (med uret), når den tegnes. Rotationen sker altid omkring figurens centrum eller midtpunkt.
- *paint* - Afgør om en flade (f.eks. i kommandoen **BOX**) skal udfyldes eller ej - dvs. om der skal tegnes en udfyldt figur eller kun et omrids.
- *inc* - Når Commodore Plus 4 tegner en cirkel på skærmen, er der i virkeligheden tale om en polygon (en mangelkant). Cirkelns omkreds dannes af en række punkter, forbundet med rette linjer. Er afstanden mellem disse punkter lille, fremkommer en cirkelvirkning, og er afstanden stor, fremkommer en polygon. Parameteren *inc* repræsenterer denne punktafstand, og den

udtrykkes i det antal grader, der er mellem punkterne, set fra cirkelens centrum.

- *koordinater* - det benyttede koordinatsystem fastlægges af SCALE-kommandoen.

#### **VIRKNING**

Forklarer virkningen og anvendelsen af den pågældende funktion eller kommando mere detaljeret.

#### **EKSEMPEL**

Giver et praktisk og kort eksempel på kommandoens eller funktionens anvendelse - evt. suppleret med en sidehenvisning til en mere uddybende forklaring af brugen.

#### **COMMODORE 64**

Giver eet eksempel på, hvordan funktionen kan simuleres i BASIC 2.0. Enkelte kommandoer, som AUTO og RENUMBER m.fl., der er inkluderet i langt de fleste TOOLKITS, vil dog ikke blive skildret. Det samme gælder BASIC 3.5's monitor. Den vil heller ikke blive simuleret i Commodore 64 BASIC, idet et egentligt monitorprogram i længden vil vise sig at have større værdi.

**Bemærk:** I alle programlistninger benyttes CHR\$-udtryk, i stedet for den grafiske repræsentation af Commodore's kontrolkarakterer - f.eks. benyttes CHR\$(14) i stedet for den specielle grafik-karakter, der betyder "skift til små bogstaver". Ligeledes erstatter betegnelsen PI den særlige Commodore karakter.

**ABS****TYPE** : Funktion - numerisk**FORMAT**: ABS(*numerisk udtryk*)**VIRKNING**: Returnerer den absolutte - dvs. positive værdi - af et tal. Den absolutte værdi af et negativt tal, er lig med tallet ganget med værdien -1.**EKSEMPEL** på anvendelse af ABS-funktionen:

```

ABS(-10)  giver værdien 10
ABS(0)    giver værdien 0
ABS(10)   giver værdien 10

```

Sammenlign funktionerne SGN og INT.

**AND****TYPE** : Logisk eller binær operator**FORMAT**: *udtryk* AND *udtryk*

**VIRKNING**: (logisk operator) AND anvendes til at skabe en logisk "OG"-forbindelse mellem to udsagn, som kan være "sande" eller "falske". Afhængigt af de enkelte udsagns sandhedsværdi, vil den resulterende OG-forbindelse antage værdien "sand" eller "falsk". Er et udsagn "sandt" udtrykkes det i Commodore BASIC med værdien "-1", og er et udsagn "falsk" tildeles det værdien "0 (nul)". For programsætningen:

```
IF udsagn1 AND udsagn2 THEN aktion...
```

gælder følgende "sandheds"-tabel:

udsagn1	falsk	falsk	sand	sand
udsagn2	falsk	sand	falsk	sand
-----				
AND-forbindelse	falsk	falsk	falsk	sand

Kun hvis både udsagn1 OG udsagn2 er "sande" (forskellige fra nul), vil aktionen (udtrykket) efter THEN blive udført.

**EKSEMPEL på anvendelse af AND som logisk operator:**

```

100 FOR N=-4 TO 4
110 : FOR M=-4 TO 4
120 : IF N=0 AND M=0 THEN PRINT N,M,1:GOTO 150
130 : IF N=0 THEN PRINT N,M,"'UENDELIG':GOTO 150
140 : PRINT N,M,(1/N)^M
150 : NEXT M
160 NEXT N

```

I linje 120 tages højde for det specialtilfælde, hvor  $1/N$  vil føre til en fejlmelding, selvom resultatet pr. definition er lig med værdien "1" (enhver værdi  $X$  i potensen 0 er lig med 1).

Sammenlign de logiske operatoren OR og NOT.

**VIRKNING: (binær operator)** Operatoren AND kan også anvendes til binære operationer. Commodore BASIC foretager alle binære operationer indenfor talområdet -32768 til 32767 - dvs. at der er tale om 16 bit operationer. Talværdier udenfor dette område resulterer i fejlmeldingen ?ILLEGAL QUANTITY. Den binære AND-funktion kan beskrives således (1-bit operation):

udtryk	resultat
0 AND 0	0
0 AND 1	0
1 AND 0	0
1 AND 1	1

**EKSEMPEL på anvendelse af AND som binær operator:**

```

100 REM DECIMAL TIL HEXADECIMAL
110 HEX$="0123456789ABCDEF"
120 INPUT"INDTAST ET TAL FRA 0 TIL 255";TAL
130 IF TAL<0 OR TAL>255 THEN 120
140 LO=TAL AND 15
150 HI=TAL AND 240
160 PRINT TAL;" = ";MID$(HEX$,HI/16,1);MID$(HEX$,LO,1)
170 GOTO 120

```

Indtastes f.eks. værdien 167 sker følgende:

```

I linje 140: (167 AND 15 = 7)
0000000010100111 AND 0000000000001111 = 0000000000000111 (07)
I linje 150: (167 AND 240 = 160)
0000000010100111 AND 0000000011110000 = 0000000010100000 (A0)
Linje 160 giver udskriften:
167 = A7

```

Sammenlign de binære operatoren OR og NOT.

## ASC

**TYPE** : Funktion - numerisk

**FORMAT**: ASC(*streng-udtryk*)

**VIRKNING**: Returnerer ASCII-værdien af den *første* karakter i strengudtrykket (se afsnittet ASCII-koder i appendix E). Funktionen returnerer en værdi fra 0 til og med 255, svarende til tegnkoden. Er den anvendte streng tom - dvs. at den ikke indeholder nogen karakter - fås fejlmeldingen ?ILLEGAL QUANTITY.  $X=ASC(X\$)$  er den omvendte funktion af  $Y\$=CHR\$(X)$ , men vær dog opmærksom på, at hvis  $X\$$  består af mere end een karakter, vil  $X\$$  og  $Y\$$  være forskellige. I det tilfælde vil følgende lighed eksistere  $Y\$=LEFT\$(X\$,1)$ .

**EKSEMPEL på anvendelse af ASC-funktionen:**

```
10 X$="A":Y$="DET VAR DET":Z$=" WOW!"
20 PRINT ASC(X$),ASC(Y$),ASC(Z$),ASC("AARHUS")
```

Linje 120 giver udskriften:

```
65      68      32      65
```

på skærmen. Bemærk, at mellemrum også tæller som karakterer. Det er især værd at have i tankerne, når der foretages sammenligninger. Der er intet i vejen for at kombinere funktionen med andre funktioner. Følgende udtryk er således lovligt, *forudsat* resultatet af MID\$-funtkionen indeholder mindst een karakter, og at alle funktioner og udtryk resulterer i gyldige værdier.

```
1230 PL=ASC(MID$(QL$,ABS(VX),LEN(Y$)-32))
```

## ATN

**TYPE** : Funktion - numerisk

**FORMAT**: ATN(*numerisk udtryk*)

**VIRKNING**: Funktionen returnerer en vinkel (udtrykt i radianer), som giver tangensværdien i det numeriske udtryk. Funktionen  $X=ATN(Y)$  er den omvendte af  $Y=TAN(X)$ . Resultatet vil altid befinde sig i intervallet  $-PI/2$  til  $+PI/2$  (-180 til +180 grader).

**EKSEMPEL på anvendelse af ATN-funktionen:**

$$10 \text{ GRADER} = \text{ATN}(X) * 180 / \text{PI}$$

**AUTO****TYPE** : Kommando (+4)**FORMAT**: AUTOAUTO *afstand*

**VIRKNING**: AUTO-kommandoen kan kun anvendes som direkte indtastning - ikke i programmer. Når AUTO-kommandoen indtastes fulgt af en talværdi - f.eks. 739 - indkobles den automatiske linjenummererings-funktion. Indtastes herefter et linjenummer fulgt af en kommando - f.eks. 100 REM - vil Commodore Plus/4 efter hvert tryk på RETURN, automatisk levere næste linjenummer, og placere cursoren et felt fra linjenummeret. I vort tilfælde vil AUTO-funktionen give linjenumre med trin på 739, startende fra linje 100 - dvs. 839, 1578, 2317 osv.

Indtastes intet efter et linjenummer - f.eks. hvis vi trykker på RETURN umiddelbart efter at linjenummeret 2317 er udskrevet, kan vi hoppe midlertidigt ud af AUTO-funktionen - f.eks. for at lave en kontrol-listning af en anden del af programmet. Så snart vi igen indtaster et linjenummer efterfulgt af en kommando - f.eks. 200 REM - genoptages den automatiske linjenummerering. Bemærk, at nummereringen fortsætter fra 200 og ikke fra linje 2317, og at de følgende linjenumre vil blive 939, 1678, 2417 etc. Da det normalt ikke er hensigten, at "blande" linjenumrene på denne måde, bør anvendelsen ske med den fornødne omtanke. Især da AUTO-funktionen *ikke* giver nogen advarsel, hvis der allerede eksisterer en programlinje med samme nummer, som det linjenummer AUTO-funktionen leverer. Trykkes på RETURN i den situation, forsvinder det "gamle" nummer! Hver gang! Og det uanset, om du indtaster noget efter linjenummeret eller ej.

Den sikreste måde at forlade AUTO-funktionen på, er at trykke på SHIFT og CLEAR/HOME knapperne samtidigt (sletter skærmen), og derefter indtaste AUTO uden linjenummer, AUTO Ø eller RUN med eller uden linjenummer. Indtastning af GOTO- eller GOSUB-kommandoer afbryder også AUTO-funktionen. Optræder AUTO-kommandoen i en programlinje, fås fejlmeldingen ?DIRECT MODE ONLY.

**EKSEMPEL på anvendelse af AUTO-kommandoen:**

```

AUTO 10  Udskriver linjenumre i trin på 10
AUTO 50  Udskriver linjenumre i trin på 50
AUTO Ø   Afbryder automatisk udskrift af linjenumre
AUTO     Afbryder automatisk udskrift af linjenumre

```

## BACKUP

**TYPE :** Disk-kommando (+4)

**FORMAT:** BACKUP Ddrevnr TO Ddrevnr

BACKUP Ddrevnr TO Ddrevnr ON Udevice-nr

**VIRKNING:** Kommandoen kopierer en diskette fra et diskdrev (drevnr 0 eller 1) uændret til det andet drev (drevnr. 1 eller 0) i en dobbelt disk-station. Formattering af måldisketten er unødvendig. Er man i besiddelse af flere disk-stationer, eller har man indstillet disk-stationens device-nummer til en anden værdi end 8, skal device-nummer medtages i kommandoen. Bemærk, at den diskette, der kopieres til, slettes, inden kopieringen begynder! Som en sikkerhed mod almindelig menneskelig svaghed (dvs. fejl), stilles spørgsmålet **ARE YOU SHURE?** altid, før kommandoen udføres. Indtastes et andet svar end Y (for Yes), udføres kommandoen ikke. Dette er dog ikke til nogen større hjælp, hvis man har specificeret en forkert *kopierings-retning*! Ønskes en *ikke-destruktiv* kopiering, skal kommandoen COPY benyttes. Hvor det er muligt, bør COPY-kommandoen foretrækkes, idet denne også "rydder op i" filerne, hvis de er blevet splittet op på en uhensigtsmæssig måde på disketten. Se også side 135.

**OBS:** Denne kommando kan kun anvendes sammen med en dobbelt disk-station - eller særlig software, som simulerer en dobbelt disk-station ved brug af to Commodore 1541 disk-stationer.

### EKSEMPEL på anvendelse af BACKUP-kommandoen:

BACKUP D0 TO D1 Kopierer disketten i drev 0 til drev 1.

BACKUP D1 TO D0 U9 Kopierer disketten i drev 1 til drev 0 på disk-stationen med device-nummeret 9.

**COMMODORE 64:** Den tilsvarende virkning opnås på Commodore 64 med BACKUP-programmet på side 125.

## BOX

**TYPE :** Grafik-kommando (+4)

**FORMAT:** BOX cs,X1,Y1,X2,Y2,rotation,paint



PARAMETER	BETYDNING	NORMALVÆRDI
cs .....	farvekilde	1 (forgrund)
X1,Y1 .....	hjørne 1 koordinat	skal specificeres
X2,Y2 .....	hjørne 2 koordinat	pixel-cursor
rotation ....	med uret i grader	0 grader
paint .....	udfyldning (0 eller 1)	0 (ingen udfyldning)

Kommandoen tegner et rektangel af vilkårlig størrelse, overalt på skærmen. Rotationen sker omkring centret i rektanglet (det punkt, hvor de to diagonaler krydses). Pixel cursoren "afleveres" i punktet X2,Y2, efter at BOX kommandoen er udført. Koordinaterne X1,Y1 og X2,Y2 rettes ind i overensstemmelse med SCALE kommandoen! Paint bestemmes om rektanglet skal udfyldes (1) eller ej (0). Ikke specificerede parametre overtages fra foregående grafik kommando.

**EKSEMPEL** på anvendelse af BOX-kommandoen:

```
BOX 1,10,10,60,60   Omrids af et rektangel i forgrundsfarven.
BOX ,10,10,60,60,45,1 Udfuldt rektangel i forgrundsfarve,
                      roteret 45 grader
BOX ,30,90,,,45,1   Udfuldt polygon, roteret 45 grader
```

**COMMODORE 64:** Kommandoen kan simuleres med programmet, der er vist på side 195. Simuleringen ser således ud:

```
100 X1=øverste højre hjørne
110 Y1=øverste højre hjørne
120 X2=øverste venstre hjørne
130 Y2=øverste venstre hjørne
140 FOR X=X1 TO X2-1
150 : SYS PLOT,X,Y
160 NEXT X
170 FOR Y=Y1 TO Y2-1
180 : SYS PLOT,X,Y
190 NEXT Y
200 FOR X=X2 TO X1+1 STEP-1
210 : SYS PLOT,X,Y
220 NEXT X
230 FOR Y=Y2 TO Y1+1 STEP-1
240 : SYS PLOT,X,Y
250 NEXT Y
```

**CHAR**

**TYPE :** Grafik-kommando (+4)

**FORMAT:** CHAR farvekilde, x, y, streng, reverse-flag

**VIRKNING:** Placerer teksten *streng* begyndende fra tekstlinje y og

tekstkolonne x på højopløsningskærmen. Kommandoen tillader blanding af tekst og grafik. Karaktererne læses fra karakter ROM'en i Commodore Plus/4. CHAR kommandoen virker eksakt som PRINT kommandoen, hvis skærmen er i TEXT mode. I GRAFIK mode kan kontrolkarakterer placeret i tekst-strengen ikke udnyttes! Ønskes udskrift i forgrundsfarven i multicolor 1 mode skal reverse flaget og farvekilden have værdien 0. I multicolor 2 mode skal reverse flaget have værdien 1.

**EKSEMPEL på anvendelse af CHAR-kommandoen:**

```
100 PRINT 0,23,12,"DENNE TEKST SKRIVES",0
110 A$="DENNE TEKST SKRIVES"
120 PRINT 0,24,12,A$,0
```

**COMMODORE 64:** En lignende kommando findes i bl.a. SIMONS BASIC til Commodore.

## CHR\$

**TYPE :** Streng-funktion

**FORMAT:** CHR\$(*numerisk udtryk*)

**VIRKNING:** Omdanner en numerisk værdi i området fra 1 til og med 255 (ASCII-værdi) til den tilsvarende Commodore-karakter. Anvendes en værdi udenfor det specificerede område, fås fejlmeldingen ?ILLEGAL QUANTITY. En oversigt over Commodores karactersæt med tilhørende ASCII-koder findes i Appendix E.

**EKSEMPEL på anvendelse af CHR\$-funktionen:**

```
100 FOR N=0 TO 128 STEP 128
110 : FOR C=32 TO 127
120 : PRINT CHR$(C+N);
130 : NEXT C
140 NEXT N
```

Dette eksempel vil udskrive alle karakterer i Commodores alfabet (excl. kontrolkarakterer).

## CIRCLE

**TYPE :** Grafik-kommando (+4)

**FORMAT:** CIRCLE *cs,ac,bc,xr,yr,sa,ea,vinkel,inc*

PARAMETER	BETYDNING	NOEMALVÆRDI
cs .....	farvekode	1 (forgrund)
ac,bc .....	centrum - koordinat	pixel-cursor
xr .....	radius i x-aksens retning	skal specificeres
yr .....	radius i y-aksens retning	xr
sa .....	start på cirkelbue	0 grader
ea .....	slut på cirkelbue	360 grader
vinkel .....	rotation omkring centrum	0 grader
inc .....	step mellem forbundne punkter	2 grader

**VIRKNING:** Kommandoen tegner omkredsen af en cirkel, ellipse eller polygon - helt eller delvist. Parameteren *inc* bestemmer "antallet" af kanter i figuren. Er værdien lille - f.eks. den værdi, der benyttes (2 grader), hvis ikke andet er specificeret, fremtræder figuren som en cirkel. Har figuren en lille radius, kan en større værdi for *inc* vælges, uden at cirkel-virkningen går tabt. Ved stor radius og store værdier for *inc* fås en polygon - en mangekant, der kan roteres omkring centrum, og "trykkes flad" ved at vælge forskellige værdier for *xr* og *yr*. På denne måde kan en 3D virkning simuleres. Ikke specificerede værdier, overtages fra tidligere grafik kommandoer (f.eks. farvekode) eller tildeles en default værdi (f.eks. 2 grader for *inc*).

**EKSEMPEL** på anvendelse af CIRCLE-kommandoen:

CIRCLE ,160,100,65,10	tegner en ellipse
CIRCLE ,160,100,65,50	tegner en cirkel
CIRCLE ,60,40,20,18,,,45	tegner en ottekant
CIRCLE ,260,40,20,,,,,90	tegner en rhombe
CIRCLE ,60,140,20,18,,,120	tegner en trekant

**COMMODORE 64:** Virkningen kan simuleres "PLOT"-kommandoen, der findes på side 195.

## CLOSE

**TYPE :** Kommando

**FORMAT:** CLOSE *logisk filnummer*

**VIRKNING:** Lukker en fil eller en kommunikationskanal. Det logiske filnummer skal være det samme, som blev benyttet i OPEN-kommandoen (se denne). Vær opmærksom på, at anvendelse af et forkert logisk filnummer ikke giver nogen fejlmelding. Især er det vigtigt, at sikre sig at åbne disk-filer eller kassettefiler lukkes korrekt, inden disketten fjernes fra disk-stationen, og inden strømmen til computer eller disk-station afbrydes.

Når filer skrives til kassettebåndoptager eller disk-station, skrives karaktererne i første omgang til en buffer. Først når bufferen er fyldt, eller når CLOSE-kommandoen udføres, skrives bufferen til filen. Selvom man med et

par krumspring, kan reparere på skaden (se M - filkommandoen på side 121), vil sidste del af filen gå tabt (op til 256 karakterer på en disk-station eller 192 karakterer med kasettebåndoptager).

**EKSEMPEL** på anvendelse af CLOSE-kommandoen:

```
100 OPEN 1,8,15
110 INPUT#1,A$,B$,C$,D$
120 CLOSE 1
130 PRINT A$;" ";B$;" ";C$;" ";D$
```

Eksemplet læser og udskriver indholdet i diskettens fejlkanal.

## CLR

**TYPE** : Kommando

**FORMAT**: CLR

**VIRKNING**: Sletter variabelhukommelsen (nulstiller alle variabler, og ophæver alle dimensionerede variabler). RUN kommandoen har samme virkning.

Ønsker man at redimensionere et numerisk array eller et streng array, skal CLR-kommandoen altid benyttes. Sker det ikke, fås fejlmeldingen ?REDIM'D ARRAY. I compilere - f.eks. PET-SPEED - er denne fremgangsmåde ikke tilladt. I compilere er forbudet mod redimensionering af arrays absolut.

Har man flyttet top of BASIC pointeren (se side 67) vil CLR-kommandoen (eller NEW) justere alle andre pointere i systemet. Først herefter vil det reserverede område være beskyttet mod overskrivning!

**EKSEMPEL** på anvendelse af CLR-kommandoen:

```
100 POKE 56,PEEK(56)-4
110 CLR
```

Programmet reserverer 1024 bytes (4x256) udenfor BASIC'ens arbejdsområde på Commodore 64 - f.eks. til maskinkode.

## CMD

**TYPE** : Kommando

**FORMAT**: CMD *logisk filnummer*

**VIRENING:** Kommandoen omdirigerer skærmudskrifter til den fil, der er specificeret i det logiske filnummer. Kommandoen benyttes f.eks., når en listning af et program skal sendes til en printer, men listningen kan sendes til ethvert aktivt device - også disk eller RS-232 (f.eks. et modem).

Før kommandoen anvendes, skal det logiske filnummer være fastlagt i en OPEN-kommando, ellers fås fejlmeldingen ?FILE NOT OPEN.

Alle PRINT og LIST kommandoer, der udføres efter CMD, vil blive omdirigeret til det udvalgte device. Optræder der en fejlmelding undervejs, kobles automatisk tilbage til skærmudskrift igen. Kommunikationskanalen er dog stadig aktiv. Deaktivering (*un-listen*) skal derfor altid foretages efter f.eks. en ?SYNTAX ERROR. Fremgangsmåden, der er vist herunder, bør også anvendes ved lukning (CLOSE) af den logiske fil - især når listning til en printer skal afsluttes. Det sikrer, at printerens interne buffer bliver tømt inden operationen afsluttes. Korrekt afslutning efter brugen af CMD ser således ud:

PRINT#*logisk fil*:CLOSE *logisk fil*

**EKSEMPEL** på anvendelse af CMD-kommandoen:

```
100 REM LISTNING AF ET PROGRAM TIL DISK
110 OPEN 2,8,3,"PROGRAMLISTNING,S,W"
120 CMD 2:LIST
130 PRINT#2:REM UN-LISTEN
140 CLOSE 2
150 REM LISTNING TIL PRINTER
160 OPEN 2,4,7:REM SMAA BOGSTAVER
170 CMD 2:LIST
180 PRINT#2:REM PRINTER-BUFFEREN TØMMES
190 CLOSE 2
```

## COLLECT

**TYPE :** Disk-kommando (+4)

**FORMAT:** COLLECT Ddrevnr

COLLECT Ddrevnr,Udevice-nr

**VIRKNING:** Genskaber BAM på en diskette ud fra de filer, der findes i directory'en. Alle ikke-lukkede filer og evt. markeringer for *random files* fjernes fra BAM! Hvis ingen parametre specificeres, udføres kommandoen på device 8, drev 0.

**EKSEMPEL** på anvendelse af COLLECT-kommandoen:

```
COLLECT D0,U8    foretager collect på en 1541 disk-station
COLLECT          foretager collect på en 1541 disk-station
```

**COMMODORE 64:** Kommandoen COLLECT er identisk med programlinjen:

```
OPEN 15,device,15,"Vdrev:":CLOSE 15
```

## COLOR

**TYPE :** Grafik-kommando (+4)

**FORMAT:** COLOR *farvekilde, farvenr, lysstyrke*

**VIRKNING:** Tildeler en farve til en af de fem mulige farvekilder:

- 0 baggrund
- 1 forgrund
- 2 multicolor 1
- 3 multicolor 2
- 4 ramme

Farverne kan specificeres fra 1 til 16, og en luminansværdi (lyestyrke) kan vælges mellem 0 (svagest) og 7 (kraftigst). Udelades luminansværdien, vælger computeren værdien 7.

### FARVETABEL:

1 SORT	5 PURPUR	9 ORANGE	13 BLÅ-GRØN
2 HVID	6 GRØN	10 BRUN	14 LYS BLÅ
3 RØD	7 BLÅ	11 GUL-GRØN	15 MØRK BLÅ
4 CYAN	8 GUL	12 LYSERØD	16 LYS GRØN

**COMMODORE 64:** Tilsvarende muligheder findes ikke på Commodore 64 (Se dog side 195).

## CONT

**TYPE :** Kommando

**FORMAT:** CONT

**VIRKNING:** Genoptager kørslen af et program, der er blevet stoppet med kommandoen STOP eller af et tryk på RUN/STOP knappen. Programmet genoptages fra den programlinje, der står umiddelbart foran udførelse -

normalt den efterfølgende programlinje.

Stoppes et program med RUN/STOP-knappen, udføres den kommando, som computeren er i gang med, først. Stop aktiveres umiddelbart før næste kommando skal udføres. Er der flere kommandoer i en programlinje, vil CONT-kommandoen starte med den "følgende" kommando, som godt kan befinde sig midt i en programlinje. Her er et par eksempler:

```
100 GOTO 320 eller 100 GOSUB 320
```

Programmet fortsættes i linje 320, og *ikke* i linje 110.

```
120 GOSUB 190
130 .....
.....
190 PRINT X:GOSUB 320:RETURN
```

Sker stoppet i linje 120, vil CONT starte på linje 190. Sker stoppet i linje 190 (efter PRINT X) vil CONT fortsætte i samme linje (GOSUB 320 udføres som det første). Sker stoppet efter GOSUB 320 vil CONT fortsætte i linje 320, og sker stoppet efter RETURN, vil CONT (i dette eksempel) fortsætte fra linje 130!

Meddelelsen **BREAK IN LINE** *linjenummer* fortæller kun, *hvilken* linje programstoppet er sket i, men intet om det linjenummer, der vil blive udført, når kommandoen CONT aktiveres.

Stopper et program med en fejlmelding, kan kørslen *ikke* genoptages med CONT. Det samme er tilfældet, hvis der rettes i programmet, inden CONT-kommandoen udføres. Trykkes på RETURN knappen, mens cursoren befinder sig over en gyldig programlinje, vil virkningen være den samme, som hvis du havde ændret i programmet.

En ændring af variablernes indhold påvirker dog ikke CONT-kommandoen. Det gør STOP- og CONT-kommandoerne til et meget anvendeligt makkerpar, mens man undersøger et program for fejl (debugging).

Kommandoer, som PRINT, LIST m.fl., der ikke ændrer selve programmet, kan benyttes uden problemer efter et stop.

## COPY

**TYPE** : Disk-kommando (+4)

**FORMAT**: COPY (Ddrev,)"original" TO (Ddrev,)"kopi" (,Udevice)

**VIKNING**: Fremstiller en kopi af en fil - enten på samme diskette, eller på en diskette i det andet drev i en dobbelt disk-station. COPY-kommandoen kan ikke anvendes til kopiering mellem to disk-stationer (forskellige devicenumre). Hvis device og drev ikke specificeres, udgår computeren fra device nr. 8,

drev nr. 0.

**EKSEMPEL på anvendelse af COPY-kommandoen:**

COPY "MOMS TIL DATO" TO "MOMS APRIL 84"

**COMMODORE 64:** Kommandoen er identisk med:

OPEN 15,device,15,"Cdrev:KOPI=drev:ORIGINAL":CLOSE 15

## COS

**TYPE :** Funktion - numerisk

**FORMAT:** COS(*numerisk udtryk*)

**VIRKNING:** Funktionen finder cosinus til argumentet (udtrykket i parantesen), hvor vinklen skal opgives i radianer. Er beregningerne baseret på grader, kan følgende omregningsformel benyttes:

$$\text{RADIANER} = \text{PI} * (\text{GRADER} / 180)$$

Commodore BASIC indeholder ikke den inverse funktion, men denne simuleres således:

$$X = -\text{ATN}(Y / \text{SQR}(1 - Y * Y)) + \text{PI} / 2$$

hvor Y er cosinusværdien i funktionen  $Y = \text{COS}(X)$  og X er vinklen i radianer. X kan antage en værdi mellem 0 og PI. Omregning til grader kan ske med formlen:

$$\text{GRADER} = 180 * \text{RADIANER} / \text{PI}$$

**EKSEMPEL på anvendelse af COS-funktionen:**

10 X=COS(Y)

## DATA

**TYPE :** Kommando



**FORMAT:** DATA *data,data,data...*

**VIRKNING:** Muliggør lagring af data som en integreret del af et program. Disse data indlæses i variabler med READ-kommandoen. DATA sætningernes elementer læses sekventielt (dvs. fra en ende af), men DATA sætningerne kan placeres frit i programmet. Det følgende er således fuldt tilladt, omend ikke særligt overskueligt:

```
100 DATA data1
110 GOTO 130
120 DATA data2
130 FOR N=1 TO 2
140 : READ X:PRINT X
150 NEXT N
170 FOR N=1 TO 2
180 : READ X:PRINT X
190 NEXT N
```

Det viste eksempel vil stoppe i linje 170 med fejlmeldingen OUT OF DATA, idet vi allerede har indlæst alle data med programlinjerne 130 til 150. Skal vi have programmet til at virke, må vi først flytte en tænkt *data-pegepind* frem til begyndelsen af DATA-sætningerne igen. Det sker med RESTORE kommandoen. Indsætter vi programlinjen:

```
160 RESTORE
```

i eksemplet herover, vil alt være i orden.

Når et program starter, placeres vor tænkte *data-pegepind*, så den peger på et punkt lige før programmet. Når en READ-kommando udføres, flyttes vor tænkte *data-pegepind* frem til det følgende dataelement, uanset hvor i programmet, dette måtte befinde sig. Findes der ikke flere data-elementer i programmet, ved computeren ikke, hvor den skal placere *pegepinden*, og den protesterer i form af en fejlmelding. Kommandoerne RUN, CLR og RESTORE har alle den egenskab, at de flytter *pegepinden* frem til starten af programmet igen. Den samme egenskab har kommandoen NEW, men da brugen af denne kommando også "sletter" programmet i computerens hukommelse, er anvendelsesmulighederne begrænsede.

DATA-sætninger kan indeholde konstanter, men ikke formler eller andre udtryk, der først skal behandles. Konstanterne kan være talværdier (f.eks. -237.54) eller tekststrengene (f.eks. OTTO). Det er kun nødvendigt at benytte anførselstegn, hvis tekststrengene indeholder tegn som:

- karaktererne ",", ":" eller mellemrum.
- grafik-karakterer eller bogstaver, som kræver anvendelse af SHIFT-knappen.
- kontrol-karakterer, f.eks. karakteren HOME.

**EKSEMPEL på anvendelse af DATA-kommandoen:**

```

100 REM TEST AF GYLDIG DATO 1901 TIL 1999
110 DATA JANUAR,FEBRUAR,MARTS,APRIL,MAJ
120 DATA JUNI,JULI,AUGUST,SEPTEMBER
130 DATA OKTOBER,NOVEMBER,DECEMBER
140 PRINT CHR$(147):REM SLET SKAERM
150 INPUT"INDTAST DAG,MAANED,AAR";D$;M$;A$
160 REM CHECK AAR
170 IF LEN(AA$)<2 OR LEN(AA$)>4 THEN RUN
180 AA=VAL(AA$):IF AA>1999 OR AA<1901 THEN RUN
190 AA=AA-1900:SK=1
200 REM CHECK SKUDAAR
210 IF AA/4 <> INT(AA/4) THEN SK=0
220 REM CHECK MAANED
230 MA=INT(VAL(M$)):IF MA<0 OR MA>12 THEN RUN
240 IF MA<>0 THEN 310
250 REM TEST FOR MAANEDSNAVNE (3 FØRSTE TEGN)
260 FOR N=1 TO 12
270 : READ X$
280 : IF LEFT$(M$,3)=LEFT$(X$,3) THEN MA=N
290 NEXT N
300 REM TEST FOR DAG
310 D=VAL(D$):IF D<0 OR D>31 THEN RUN
320 IF MA=2 AND D>(28+SK) THEN RUN
330 IF (MA=4 OR MA=6 OR MA=9 OR MA=11) AND D>30 THEN RUN
340 REM UDSKRIV MED KORREKT STAVET MAANEDSNAVN
350 RESTORE
360 FOR N=1 TO MA
370 : READ X$
380 NEXT N
390 PRINT "DATOEN: ";D;". ";X$;1900+AA;" ER GYLDIG"
400 GET A$:IF A$="" THEN 400
410 RUN

```

**DEC**

**TYPE** : Funktion - numerisk (+4)

**FORMAT**: DEC(*streng-udtryk*)

**VIRKNING**: Konverterer en hexadecimal strengværdi i området fra 0000 til FFFF til en decimalværdi (0 til 65535). Se også funktionen HEX\$.

**EKSEMPEL på anvendelse af DEC funktionen:**

```

100 INPUT"HEXADECIMAL VÆRDI";X$
110 PRINT"DECIMALVÆRDIEN ER";DEC(X$)

```

**COMMODORE 64:** Den viste rutine kan udvides til større hexadecimale værdier, om ønsket:

```

100 HEX$="0123456789ABCDEF":DEC=0
110 INPUT "HEXADECIMAL VÆRDI";X$
120 FOR N=LEN(X$) TO 1 STEP -1
130 : FOR H=1 TO 16
140 : IF MID$(HEX$,H,1)=MID$(X$,N,1) THEN H=99:
      DEC=DEC+(H-1)*16^(N-1):GOTO 150
150 : NEXT H
160 : IF H<99 THEN PRINT "ERROR":N=99
170 NEXT N
180 IF N>99 THEN RUN
190 PRINT "DECIMALVÆRDI ER";DEC
200 RUN

```

## DEF FN

**TYPE :** Kommando

**FORMAT:** DEF FN *navn( numerisk variabel ) = numerisk udtryk*

**VIRKNING:** Definerer en brugerspecificeret funktion. Funktionen kan indeholde et vilkårligt matematisk udtryk, der resulterer i en numerisk værdi, og kan rummes på een programlinje.

Brugerdefinerede funktioner anvendes med fordel, hvor flere dele af et program benytter samme formel. Formlen skal kun specificeres een gang, og kaldes med funktionen:

FN *navn( numerisk udtryk )*

Navnet kan bestå af et eller to tegn, hvor første tegn skal være i området A til Z, og andet tegn yderligere kan indeholde cifrene 0 til 9.

Programmet *skal* møde formeldefinitionen *før* funktionen kaldes, men kun compilere kræver, at formeldefinitionen også skal befinde sig *før* det første formelkald rent fysisk - dvs. at et formelkald skal have højere linjenummer end formel-definitionen, hvis man benytter en compiler. Computerens indbyggede BASIC er ikke så kræsen, og i modsætning til en compiler, fås ingen fejlmelding, hvis programmet møder samme formeldefinition flere gange i programmet. Formel-definitionen ignoreres blot efter "det første møde".

**EKSEMPEL på anvendelse af DEF FN kommandoen:**

```

100 REM KONVERTERING FRA BINER TIL DECIMAL
110 DEF FN BN(N)=VAL(MID$(IP$,N,1))*2^(LEN(IP$)-N)
120 INPUT "INDTAST BINAER-TAL";IP$
130 B=0
140 FOR N=1 TO LEN(IP$)
150 : B=B+FN BN(N)
160 NEXT N
170 PRINT:PRINT IP$;" = ";B

```

Bemærk, at der ikke foretages nogen kontrol af indtastningens rigtighed - kun 0 og 1 er gyldige cifre!

**DIM**

**TYPE :** Kommando

**FORMAT:** DIM *variablenavn*(*elementer*,*elementer*...)

**VIRKNING:** Dimensionerer en variabel - dvs. reserverer plads til et streng eller numerisk array (Se side 36). Et array kan maksimalt indeholde 32768 elementer (element 0 til 32767) og maksimalt have 255 dimensioner - i praksis vil det sige, at det er hukommelsen, der sætter grænsen. F.eks. har nedenstående arrays hver tre dimensioner:

```

DIM AN$(3,2,1)  strengarray
DIM AN%(3,2,1)  integer array
DIM AN(3,2,1)   numerisk array

```

Alle arrays rummer 4x3x2=24 elementer (husk, at elementerne tælles fra 0). Hvert element i strengarray'et kan indeholde fra 0 til 255 karakterer, og hvert element i de to andre arrays kan rumme et tal indenfor definitionsområdet - for integer-variable går området fra -32767 til og med 32767.

Møder et program en arrayvariabel, der ikke tidligere er dimensioneret, reserveres indtil 11 elementer i hver dimension. Kræves flere elementer i en dimension, fås en fejlmedaling. Her er et par eksempler:

```

100 A$(10,10,10)=323      tilladt
110 A$(10,10,10)="PETER"  tilladt
120 A(10,10,10)=75689.234 tilladt
130 A(1,1,11)=75689.234   ikke tilladt

```

Linje 130 giver fejlmedalingen ?BAD SUBSCRIPT ERROR, men da hvert element i et array kræver plads i hukommelsen, kan det ske, at computeren løber ind i pladsproblemer (læs mere om pladskrav på side 37). Det signaleres med fejlmedalingen OUT OF MEMORY ERROR. F.eks. vil hver af de følgende programlinjer give denne fejlmedaling, når linjen udføres:

```
140 A(10,10,10,10)=75689.234
150 DIM B(10,10,10,10)
```

selvom linjen i sig selv er tilladt. Det er bl.a. een af årsagerne til, at man normalt dimensionerer arrays som det første i et program. Så er tingene lettere at overskue. Man kan udmærket definere flere arrays på en linje - f.eks.:

```
100 DIM A$(3,3),A%(2,7),ALFA(3,4)
```

eller således:

```
100 DIM A$(3,3):DIM A%(2,7):DIM ALFA(3,4)
```

Virningen er den samme, og valg af metode er et spørgsmål om personlig indstilling.

#### EKSEMPEL på anvendelse af DIM-kommandoen:

```
100 REM BLANDING AF ET SPIL KORT
110 DIM KORT$(52)
120 FOR N=1 TO 52
130 : KORT$(N)=N-1
140 NEXT N
150 KORT$(0)=52:REM KORT I BUNKEN
160 REM BLANDING
170 FOR N=1 TO 52
180 : X=INT(1+52*RND(0)):Y=INT(1+52*RND(0))
190 : A=KORT$(X):KORT$(X)=KORT$(Y):KORT$(Y)=A
200 NEXT N
210 REM KORTGIVNING
220 IF KORT$(0)=0 THEN KORT$(0)=52:GOTO 170
230 X=KORT$(KORT$(0))
240 FARVE=INT(X/13):REM 0 TIL 3
250 VAERDI=1+X-FARVE*13:REM 1 TIL 13
260 KORT$(0)=KORT$(0)-1
270 RETURN
```

Eksemplet er udformet som en subrutine. Rutinen skal kaldes med GOTO 110 en og kun een gang. Et nyt kort (FARVE og VAERDI) fås med GOSUB 220. Er der ikke flere kort i bunken, udføres en blanding (afgøres i linje 220).

#### DIRECTORY

TYPE : Disk-kommando (+4)  
 FORMAT: DIRECTORY (Ddrev),Udevice,("mønster")

**VIRKNING:** Lister en diskettes indhold til skærmen, uden at påvirke indholdet i computerens hukommelse. Hvis parametrene *drev* og *device* ikke specificeres, læses automatisk *device* nr. 8, *drev* nr. 0

**EKSEMPEL på anvendelse af DIRECTORY kommandoen:**

DIRECTORY "?ST\*" lister filnavne, hvor 2. og 3. bogstav er S og T.

**COMMODORE 64:** Sammenlign >\$ kommandoen i DOS 5.1, der kommer nærmest DIRECTORY kommandoen. Device udvælges med kommandoen >#*device*.

## DLOAD

**TYPE :** Disk-kommando (+4)

**FORMAT:** DLOAD "FILNAVN"(*Ddrev,Udevice*)

**VIRKNING:** Indlæser et BASIC-program fra diskette. Hvis ingen parametre medtages, læses programmet automatisk fra *device* nr. 8, *drev* nr. 0. Kommandoen kan ikke anvendes til indlæsning af maskinkode rutiner (benyt LOAD i stedet).

**EKSEMPEL på anvendelse af DLOAD disk-kommandoen:**

DLOAD "DETTE HER GAAR"  
DLOAD (A\$) - *husk paranteserne!*

**COMMODORE 64:** Kommandoen er identisk med:

LOAD "*drev*:FILNAVN",*device*

## DO

**TYPE :** Kommando (+4)

**FORMAT:** DO UNTIL *udtryk*: *program*. LOOP  
DO WHILE *udtryk*: *program*. LOOP

**VIRKNING:** DO kommandoen informerer Plus/4 computeren om, at den skal udføre en programløkke. Typen fastlægges af den næstefølgende kommando - enten WHILE eller UNTIL. DO kommandoen kan anvendes uden nærmere specifikation. De tre løkkestrukturer virker således:

DO UNTIL <i>betingelse</i>	DO WHILE <i>betingelse</i>	DO
.....	.....	.....
<i>program</i>	<i>program</i>	<i>program</i>
.....	.....	.....
LOOP	LOOP	LOOP

Alle udgaverne kan kombineres med en EXIT-kommando, hvis man ønsker at forlade løkken "før tid". Den tredje udgave af DO kommandoen vil skabe en såkaldt uendelig løkke, som ikke forlades af programmet, medmindre en EXIT kommando anvendes - f.eks. som her:

```
100 DO
120 A=A+1
130 IF A=23 THEN EXIT
140 PRINT A
150 LOOP
```

EXIT kommandoen afbryder løkken, og programmet fortsætter efter linje 150. Sammen med FOR/NEXT løkkerne råder Plus/4 brugere over tre *forskellige* løkkestrukturer, som hver har deres specielle anvendelsesområde baseret på de forskelle, der er i deres egenskaber:

LØKKETYPE	GENNEMLØB	MINIMUM	FASTLÆGGES	TESTES
FOR/NEXT	fast	1 gang	i starten	til slut
DO WHILE	variabel	0/1 gange	start eller slut	
DO UNTIL	variabel	0/1 gang	start eller slut	
DO/EXIT	variabel	1 gang	undervejs	undervejs

DO/EXIT kan betragtes som et specialtilfælde af DO UNTIL. UNTIL løkker udføres indtil en eller anden betingelse er opfyldt, og WHILE løkker udføres, mens en eller anden betingelse er opfyldt. Afhængigt af, hvordan løkken er udformet, testes betingelsen i starten eller i slutningen af løkken. Testes i starten kan løkken overspringes, hvis betingelsen ikke er opfyldt. Læs evt. også side 43 om FOR/NEXT løkker.

**EKSEMPEL på anvendelse af DO kommandoen:**

```
100 DO
120 B=B+A
130 PRINT A
140 WHILE B<C
```

I dette eksempel testes løkken først i slutningen.

**COMMODORE 64:** De viste løkketyper kan alle simuleres af kombinationer af IF/THEN og FOR/NEXT kommandoer eller af IF/THEN kommandoer alene. F. eks. en DO WHILE og DO UNTIL simulering:

100 IF X<100 THEN 130	100 IF X>=100 THEN 130
110 X=X+P	110 X=X+P
120 GOTO 100	120 GOTO 100

svarende til:

100 DO WHILE X<100	100 DO UNTIL X>=100
110 X=X+P	110 X=X+P
120 LOOP	120 LOOP

**DRAW****TYPE** : Grafik-kommando (+4)**FORMAT**: DRAW *fk,x,y,(TO a,b)*

**VIRKNING**: Kommandoen anvendes til at tegne punkter, linjer eller figurer på grafik-skærmen i den specificerede farvekilde (se COLOR kommandoen). Ikke specificerede parametre tager udgangspunkt i sidst definerede værdier. Udelades TO, tegnes et punkt.

**EKSEMPEL** på anvendelse af DRAW grafik-kommandoen:

```
100 DRAW 1,10,30 TO 90,90 TO 40,80 TO 10,30
110 DRAW ,, TO 34,77
```

Første eksempel tegner en trekant, og andet eksempel en linje fra sidste definerede punkt eller midten af skærmen, hvis ingen grafik-kommandoer har været benyttet forud.

**COMMODORE 64**: Se "PLOT"-kommandoen side 195.**DS****TYPE** : kontrolvariabel (+4)**VIRKNING**: Returnerer fejlnummeret efter udførelsen af en diskkommando.**EKSEMPEL** på anvendelse af DS kontrolvariablen:

```
100 OPEN 2,8,7,"FILNAVN,S,R"
110 IF DS>19 THEN CLOSE 8:GOTO fejlrutine
```

**COMMODORE 64**: Se afsnittet om DOS 5.1.



DS\$

**TYPE** : Kontrolvariabel (+4)**VIRKNING**: Returnerer *fejlmeldingen* efter udførelsen af en diskkommando.**EKSEMPEL på anvendelse af DS\$ kontrolvariablen:**

```
100 OPEN 2,8,7,"FILNAVN,S,R"  
110 IF DS>19 THEN CLOSE 8:PRINT DS$
```

**COMMODORE 64**: Se afsnittet om DOS 5.1.

DSAVE

**TYPE** : Disk-kommando (+4)**FORMAT**: DLOAD "*programnavn*"DLOAD "*programnavn*",D*drive*DLOAD "*programnavn*",D*drive*,U*device*

DLOAD (A\$)....

**VIRKNING**: Indlæser et program fra det specificerede *drive* på det specificerede *device* (disk-station). Udeladte parametre erstattes med henholdsvis *device* nr. 8 og *drive* nr. 0. Bemærk parantesen omkring strengvariablen, som indeholder programnavnet.

**EKSEMPEL på anvendelse af DLOAD kommandoen:**

```
100 DLOAD "PROGRAMNAVN",D0,U8
```

**COMMODORE 64**: Kommandoen er identisk med:

```
100 LOAD "0:PROGRAMNAVN",8
```

**EL**

**TYPE** : Kontrolvariabel (+4)

**VIRKNING:** Indeholder linjenummeret, hvor en fejl er optrådt. Se kommandoen TRAP.

**COMMODORE 64:** Kræver BASIC udvidelse.

**END**

**TYPE** : Kommando

**FORMAT:** END

**VIRKNING:** Afslutter kørslen af et program. Kommandoen er kun nødvendig, når et program skal slutte på en linje, der ikke befinder sig sidst i programmet. STOP kommandoen, der har samme virkning, er normalt forbeholdt til midlertidig brug under fejlfinding i et program. På andre computere kan der optræde forskelle mellem de to kommandoer.

**ER**

**TYPE** : Kontrolvariabel (+4)

**VIRKNING:** Indeholder fejlkoden for en fejl, der er indtrådt i et program. Se kommandoen TRAP.

**COMMODORE 64:** Kræver BASIC udvidelse.

**ERR\$**

**TYPE** : Kontrolvariabel (+4)

**VIRKNING:** Indeholder fejlmeldingsteksten på en fejl, der er opstået i et program. Se kommandoen TRAP.

**COMMODORE 64:** Kræver BASIC udvidelse.

**EXIT**

**TYPE** : Kommando (+4)

**FORMAT:** EXIT

**VIRKNING:** Gør det muligt at forlade et DO, DO UNTIL eller DO WHILE loop "før tiden". Se DO kommandoen.

**EKSEMPEL på anvendelse af EXIT kommandoen:**

```
100 DO
110 A=A*1
120 IF A>999 THEN EXIT
130 PRINT A
140 LOOP
```

**COMMODORE 64:** Se DO kommandoen.

**EXP**

**TYPE** : Funktion - numerisk

**FORMAT:** EXP(*numerisk udtryk*)

**VIRKNING:** Returnerer værdien  $e^x$  - det naturlige tal opløftet i potensen X. Den inverse funktion af LOG(X).

**EKSEMPEL på anvendelse af EXP-funktionen:**

LØ X=EXP(Y)

**FN**

**TYPE :** Funktion

**FORMAT:** FN *funktionsnavn*(*numerisk udtryk*)

**VIRKNING:** Kaldet en bruger-defineret funktion. Se DEF FN side 221.

**FOR..TO..(STEP..)**

**TYPE :** Kommando

**FORMAT:** FOR *variabel*=*udtryk1* TO *udtryk2* (STEP *udtryk3*)

....*program*

NEXT (*variabel*)

**VIRKNING:** Løkke-kommando, som specificerer en kontrolvariabel for en programløkke, startværdien for variabelen (*udtryk1*) og slutværdien (*udtryk2*), der bestemmer, hvornår løkken skal forlades. Hvis STEP kommandoen udelades, opskrives variabelen med værdien 1, hver gang NEXT kommandoen udføres. STEP kommandoen kan fastlægge en vilkårlig positiv eller negativ værdi for denne "opskrivning" (*udtryk3*). Der kan kun anvendes normale numeriske variable som FOR/NEXT kontrolvariable (se side 34). Det er ikke nødvendigt at specificere variabelnavnet i NEXT kommandoen, men det letter overskueligheden i et program. Læs mere om FOR/NEXT løkker på side 43.

**FRE**

**TYPE :** Funktion

**FORMAT:** FRE(*dummy variabel*)

**VIRKNING:** Funktionen returnerer størrelsen af den ledige plads i hukommelsen og fremtvinger en *garbage collection* - se side 46. Mens Plus/4

altid giver et korrekt resultat, vises alle tal over 32767 som negative tal på Commodore 64. I disse situationer kan den korrekte værdi bestemmes således:

```
PRINT 2^16+FRE(X)
```

## GET

**TYPE :** Kommando

**FORMAT:** GET *variabelnavn*

**VIRKNING:** Returnerer en værdi fra computerens tastaturbuffer. Findes ingen karakterer i tastaturbufferen, returneres en tom streng - dvs. ingen karakterer! GET kommandoen kan benyttes sammen med numeriske eller integer variabler og arrays, men trykkes der på en knap, der er forskellig fra cifrene 0 til 9, fås fejlmeldingen **SYNTAX ERROR**. Derfor anvendes strengkommandoen kun i kombination med en eller flere strengvariabler i praksis (se også kapitel 2).

**EKSEMPEL på anvendelse af GET kommandoen:**

```
100 GET A$:IF A$="" THEN 100
```

## GET#

**TYPE :** Kommando

**FORMAT:** GET#*lf, variabel, ...*

**VIRKNING:** Indlæser en værdi fra en kanal, der er åbnet med det *logiske filnummer lf*. Kommandoen virker analogt med GET kommandoen. Bemærk, at ASCII værdien 0 (nul) returneres som en tom streng!

**EKSEMPEL på anvendelse af GET# kommandoen:**

```
100 OPEN 3,8,3,"FILNAVN,U,R"  
110 GET#3,A$:IF A$="" THEN A$=CHR$(0)
```

**GETKEY**

**TYPE** : Kommando (+4)

**FORMAT**: GETKEY *variabel*,...

**VIRKNING**: Identisk med GET kommandoen, bortset fra at kommandoen afventer indtastning af en karakter, hvis tastaturbufferen er tom.

**COMMODORE 64**: Se programlinjen i GET-kommandoen.

**GOSUB**

**TYPE** : Kommandoen

**FORMAT**: GOSUB *linjenummer*

**VIRKNING**: Kaldet en subrutine, som udføres fra det specificerede linjenummer. Eksisterer den pågældende programlinje ikke, fås en fejlmelding, og programmet stopper. Programmet returnerer til den kommando eller den programlinje, som følger umiddelbart efter GOSUB kommandoen, når programmet møder en RETURN kommando i subrutinen. Der er intet i vejen for, at subrutiner kan indeholde GOSUB kommandoer, der kalder andre subrutiner.

**GOTO**

**TYPE** : Kommando

**FORMAT**: GOTO *linjenummer*

**VIRKNING**: Tvinger programmet til at fortsætte kørslen fra det specificerede linjenummer. Eksisterer den pågældende programlinje ikke, fås en fejlmelding, og programmet stopper. GOTO kommandoen skal omgås med varsomhed, idet overdreven brug kan føre til, at programmer bliver helt uoverskuelige i deres struktur, og faren for fremstilling af "uendelige løkker" som i det følgende eksempel er altid til stede.

**SKRÆKKEKSEMPEL på anvendelse af GOTO kommandoen:**

```
100 PRINT "DETTE ";:GOTO 150
120 PRINT "TYDELIGT ":GOTO 140
130 PRINT "ILLUSTRERER ":GOTO 120
140 PRINT "PROBLEMET.":GOTO 100
150 PRINT "EKSEMPEL ";:GOTO 130
```

**GRAPHIC**

**TYPE** : Grafik kommando (+4)

**FORMAT**: GRAPHIC *skærmtype*(,*clearparameter*)  
GRAPHIC CLR

**VIRKNING**: Vælger mellem flere forskellige skærmtyper. Typen bestemmes af værdien *skærmtype*:

- 0 normal tekst
- 1 grafik
- 2 grafik, split screen
- 3 multicolor grafik
- 4 multicolor grafik, split screen

Benyttes en af værdierne 1 til 4, reserverer computeren automatisk 10 K af hukommelsen til grafik-skærmen. Denne hukommelsesplads frigøres *ikke*, før kommandoen GRAPHIC CLR udføres. Specificeres ingen *clearparameter*, er det muligt at skifte mellem grafik og tekst, *uden* at skærmenes indhold slettes! Benyttes en værdi på 1, slettes den skærm, man skifter til.

**COMMODORE 64**: Se kapitlet Commodore 64 grafik på side 186.

**GSHAPE**

**TYPE** : Grafik kommando (+4)

**FORMAT**: GSHAPE *strengvariabel*,*x*,*y*,*virkning*

**VIRKNING**: Se kommandoen SSHAPE.

**HEADER**

**TYPE** : Disk-kommando (+4)

**FORMAT**: HEADER "*disknavn*"(, *ID*, *Ddrive*, *Udevice*)

**VIRKNING**: Formatterer en diskette i drev nr. *drev* på device nr. *device*. Parametrene er ikke nødvendige, hvis der kun benyttes een disk-station med eet drev - f.eks. en 1541 disk-station med devicenummer 8. I alle andre tilfælde skal alle parametrene medtages, hvis man vil undgå faren for formattering og dermed sletning af hele indholdet på en forkert diskette. Disketten tildeles navnet *disknavn* og ID-koden *ID*. ID-koden kan udelades, hvis disketten har været brugt tidligere, og man kun ønsker at slette indholdet i *directory*.

**COMMODORE 64**: Kommandoen er identisk med:

OPEN 8,*device*,8,"Ndrev:*disknavn*,*ID* "

**HELP**

**TYPE** : Kommando (+4)

**FORMAT**: HELP

**VIRKNING**: Når kommandoen indtastes efter en fejlmelding i et program, vises den fejlbehæftede linje på skærmen. Linjens indhold blinker, startende fra den første kommando, der er involveret i fejlen. Funktionen kan udløses ved et tryk på HELP knappen eller ved indtastning. Det er også muligt at anvende kommandoen i et program - f.eks. som led i en TRAP rutine.

**COMMODORE 64**: Kræver BASIC udvidelse.

**HEX\$**

**TYPE** : Funktion (+4)

**FORMAT**: HEX\$(*numerisk variabel*)



**VIRKNING:** Konverterer en numerisk værdi til en hexadecimal streng bestående af fire cifre. Den numeriske værdi *skal* være i området 0 til 65535, ellers fås fejlmeldingen ILLEGAL QUANTITY.

**EKSEMPEL** på anvendelse af HEX\$ funktionen:

```
PRINT HEX$(15)
```

Vil udskrive værdien "000F" på skærmen.

**COMMODORE 64:** Se eksemplet i AND kommandoen.

**IF..THEN..**

**TYPE :** Kommando

**FORMAT:** IF *betingelse* THEN *aktion*

**VIRKNING:** Betingelsen mellem IF og THEN undersøges. Er betingelsen opfyldt tildeles den værdien -1, og er betingelsen ikke opfyldt tildeles udtrykket værdien 0. Kun hvis værdien er ulig nul, udføres THEN kommandoen. Normalt skal THEN følges af en anden kommando - f.eks. PRINT - men er der tale om et GOTO, kan der vælges frit mellem de følgende tre udgaver:

```
100 IF A THEN GOTO 900 100 IF A THEN 900 100 IF A GOTO 900
```

**INPUT**

**TYPE :** Kommando

**FORMAT:** INPUT *variabel*,...

INPUT "*meddelelse*"; *variabel*

**VIRKNING:** Indlæser indtastninger fra et tastatur til en eller flere variabler (adskilt af kommaer). Indtastningerne kan ligeledes adskilles af kommaer, eller indtastes enkeltvis. Et tryk på RETURN kommandoen afslutter en indtastning. Er der flere variabler i samme INPUT-kommando, vil aktiveres spørgsmålstegnet for næste indtastning på den følgende linje. Først når alle variabler er indtastede, fortsætter programmet. Der kan maksimalt indtastes 80 karakterer, svarende til 2 skærmlinjer. Alle tegn kan anvendes i meddelelsen - også farvekoder, cursorkoder m.m. (se side 29).

**EKSEMPEL på anvendelse af INPUT kommandoen:**

```
100 INPUT "FØDSELS DAG, MAANED OG AAR";FD,MA$,AA(3)
```

**INPUT#**

**TYPE :** Kommando

**FORMAT:** INPUT#*lf, variabel, ...*

**VIRKNING:** Kommandoen indlæser værdier til variabler fra det *logiske* filnummer - *lf* -, der er specificeret tidligere i en OPEN kommando (se side 112).

**INSTR**

**TYPE :** Funktion (+4)

**FORMAT:** INSTR(*streng1, streng 2, (startposition)*)

**VIRKNING:** Funktionen søger efter forekomsten af en streng (streng 2) i en anden streng (streng 1). Er streng 2 en del af eller lig med streng 1, returneres karakterpositionen for første sammenfald. Indeholder den *afsøgte* streng(variabel) ikke den *søgte* streng(variabel) returneres værdien nul. Normalt *afsøges* en streng fra første karakter, men en anden startposition kan specificeres, om ønsket. Da strenge kun kan indeholde op til 255 værdier, vil en større værdi for startpositionen resultere i værdien ILLEGAL QUANTITY.

**EKSEMPEL på anvendelse af INSTR funktionen:**

```
100 A=0: A$="PETERSEN ER - NAA JA!"
110 DO
120 A=INSTR(A$,"ER",A+1)
130 IF A THEN PRINT A
140 LOOP WHILE A
```

Programmet vil udskrive værdierne 4 og 10 på skærmen.

**COMMODORE 64:** Programeksemplet herunder udfører en identisk funktion:

```
100 A$="PETERSEN ER - NAA JA!":B$="ER"
110 FOR N=1 TO LEN(A$)+1-LEN(B$)
120 : IF MID$(A$,N,LEN(B$))=B$ THEN PRINT N
130 NEXT N
```

## INT

**verbatim**<TYPE : Funktion - numerisk FORMAT: INT(*numerisk udtryk*)>

**VIRENING:** Returnerer en integer værdi af det numeriske udtryk.

**EKSEMPEL på anvendelse af INT-funktionen:**

```
10 X=INT(Y)
```

## JOY

**TYPE :** Funktion (+4)

**FORMAT:** JOY(*joystick nr*)

**VIRENING:** Læser joystick 1 eller 2. Der returneres en værdi mellem 1 (op) og 8 (opad, til venstre) Værdien øges med en for hver 45 grader i solens retning. Trykkes på FIRE knappen adderes 128 til resultatet.

**EKSEMPEL på anvendelse af JOY funktionen:**

```
100 FOR N=1 TO 2
120 Y=JOY(N):PRINT "JOYSTICK";N,
130 IF Y AND 128 THEN PRINT "FIRE ";
140 IF Y =3 THEN PRINT "RIGHT ";
150 IF Y =7 THEN PRINT "LEFT ";
160 IF Y =5 THEN PRINT "DOWN ";
170 IF Y =1 THEN PRINT "UP ";
180 PRINT
190 NEXT N
200 GOTO 100
```

**COMMODORE 64:** Se side 76.

**KEY**

**TYPE :** Kommando (+4)

**FORMAT:** KEY *F-knap, strengværdi*  
KEY

**VIRKNING:** Programmerer funktionsknapperne F1 (knap nr 1) til F8/HELP (knap nr 8). Summen af alle tegnene i de enkelte funktionsknapper kan maksimalt være 128 karakterer, eller 16 tegn i gennemsnit - incl. alle kontrolkarakterer. Anvendes KEY uden efterfølgende parametre, udskrives en liste over funktionsknappernes indhold.

**EKSEMPEL på anvendelse af KEY kommandoen:**

```
100 KEY
110 KEY 1, "LOAD"+CHR$(34)+"$*=PRG"+CHR$(34)+",8"+CHR$(13)
120 KEY
```

Bemærk, at anførselstegn skal programmeres som CHR\$(34).

**COMMODORE 64:** Kræver BASIC udvidelse.

**LEFT\$**

**TYPE :** Funktion

**FORMAT:** LEFT\$(*strengudtryk, antal tegn*)

**VIRKNING:** Udlæser et specificeret antal tegn fra et strengudtryk, optalt fra venstre side (1. tegn). Specificeres et negativt antal, fås fejlmeldingen ILLEGAL QUANTITY. Benyttes antallet nul, udlæses ingen tegn. Benyttes et antal, der er større end eller lig med antallet af tegn i strengudtrykket, udlæses hele strengen.

**EKSEMPEL på anvendelse af LEFT\$ funktionen:**

```
100 PRINT LEFT$(CHR$(34)+A$+STR$(N), LEN(A$)+3)
```

**LEN****TYPE** : Funktion**FORMAT**: LEN(*strengudtryk*)**VIRKNING**: Udskriver antal tegn, der er indeholdt i det specificerede strengudtryk.**EKSEMPEL på anvendelse af LEN funktionen:**

```
100 A$="TRE":PRINT A$+" ";LEN(A$)
```

**LET****TYPE** : Kommando**FORMAT**: LET *variabel*=*udtryk***VIRKNING**: Tildeler en værdi til en variabel. LET kommandoen er dog ikke nødvendig. Begge eksempler herunder har samme virkning:

```
LET A=3
A=3
```

**LIST****TYPE** : Kommando**FORMAT**: LIST**LIST** *linjenummer*(-*linjenummer*)**VIRKNING**: Kommandoen LIST'er et program (LIST alene), dele af et program, eller en enkelt programlinje. Benyttes kommandoen i et program, stopper kørslen umiddelbart efter, at kommandoen er udført. Kommandoen har mulighederne:

LIST               lister alle programlinjer  
LIST ~~100~~-        lister alle programlinjer fra linje 100  
LIST ~~-200~~        lister alle programlinjer til og med 200  
LIST ~~100-200~~    lister programlinjer fra 100 til og med 200

## LOAD

TYPE : Kommando

FORMAT: LOAD "

LOAD "*programnavn*"(, *device*, *funktion*)

**VIRKNING:** Indlæser et program fra kassette eller diskette. LOAD kommandoen kan anvendes alene (svarer til et tryk på RUN-knappen), eller med navnet alene, hvis man indlæser fra kassette. Ved læsning fra diskette *skal* devicenummeret specificeres (På Commodore SX-64 virker RUN-knappen således, at første program på disketten indlæses). Der kan specificeres to funktioner for LOAD. Udelades værdien, eller angives værdien 0 (nul) indlæses programmet fra starten af BASIC området, og angives værdien til 1, indlæses programmet på samme sted i hukommelsen, som det blev SAVE't fra. Se side 106 og 85.

## LOCATE

TYPE : Kommando (+4)

FORMAT: LOCATE X,Y

**VIRKNING:** Placerer grafikkursoren på det specificerede punkt på skærmen (se også SCALE). Kommandoen kan ikke anvendes sammen med tekst.

**COMMODORE 64:** Se "PLOT" kommandoen på side 195.

## LOG

TYPE : Funktion - numerisk

**FORMAT:** LOG(*numerisk udtryk*)

**VIRKNING:** Returnerer den naturlige logaritme. Den inverse funktion af EXP(X). Funktionen må ikke forveksles med *titals-logaritmen*, som normalt bærer betegnelsen *log* her i landet!

**EKSEMPEL på anvendelse af LOG-funktionen:**

10 X=LOG(Y)

**LOOP**

**TYPE :** Kommando (+4)

**FORMAT:** LOOP

LOOP WHILE *betingelse*

LOOP UNTIL *betingelse*

**VIRKNING:** Markerer afslutningen på en DO løkke. Se DO kommandoen side 224.

**MID\$**

**TYPE :** Funktion

**FORMAT:** MID\$(*strengudtryk, startbogstav, antal tegn*)

**VIRKNING:** Funktionen returnerer et specificeret antal tegn, begyndende fra positionen *startbogstav*. For begge numeriske parametre gælder samme regler, som for LEFT\$ (se side 238).

**MONITOR**

**TYPE :** Kommando (+4)

**FORMAT:** MONITOR

**VIRKNING:** Indkobler MONITOR'en på Commodore C16 og Plus/4. Se kapitel 5 side 78.

**COMMODORE 64:** Kræver særligt program eller programmodul.

## **NEW**

**TYPE :** Kommando  
**FORMAT:** NEW

**VIRKNING:** Sletter indholdet i computerens hukommelse.

## **NEXT**

**TYPE :** Kommando  
**FORMAT:** NEXT  
          NEXT *variabelnavn*

**VIRKNING:** Markerer afslutningen på en FOR-NEXT løkke. Se kommandoen FOR på side 230. På Plus/4 og C16 computerne anvendes NEXT også i forbindelse med RESUME kommandoen.

**EKSEMPEL på anvendelse af NEXT kommandoen:**

```
100 NEXT X,Y,Z
```

## **NOT**

**TYPE :** Logisk eller binær operator  
**FORMAT:** NOT *numerisk variabel*

**VIRKNING:** (logisk operator) NOT anvendes til at invertere betydningen af et udsagn i en IF...THEN sætning. F.eks. vil udsagnet:

A>3



returnere værdien 0 (nul), hvis A er mindre end eller lig med nul, og værdien -1, hvis udsagnet er sandt. Udsagnet:

NOT(A>3)

returnerer værdien 1, hvis A er mindre end eller lig med nul, og værdien 0 (nul), hvis udsagnet er sandt. Den logiske NOT operator har følgende sandhedstabel:

udsagn	falsk	sand
<hr/>		
NOT forbindelse	sand	falsk

**VIRKNING:** (binær operator) returnerer en 16 bit binær komplementærværdi (two's complement) af et integerudtryk. Den resulterende værdi er således lig med komplementærværdien plus en:

NOT X% = -(X+1)

Se også AND operatoren på side 206.

## ON

**TYPE :** Kommando

**FORMAT:** ON *numerisk udtryk* GOTO *linjenumre*

ON *numerisk udtryk* GOSUB *linjenumre*

**VIRKNING:** Foretager et ubetinget (GOTO) eller subrutine-spring til en rutine, der udvælges på basis af værdien af det numeriske udtryk. F.eks. vil kommandoen:

ON X GOSUB 200,300,400

medføre, at der springes til linje 200, hvis X er 1, linje 300, hvis X er 2, og linje 400, hvis X er 3. Er X nul eller større end 3 udføres den efterfølgende programlinje. En negativ X-værdi giver en fejlmelding.

**OPEN****TYPE :** Kommando**FORMAT:** OPEN *If*, *device*, *sa*, ("FILNAVN")

**VIRKNING:** Åbner en kommunikationskanal med det logiske filnummer *If* til *device* med den sekundære adresse eller kanal *sa*. Se kapitlerne om skærm, maskinkode, disk-station, kassettebåndoptagere og RS-232.

**OR****TYPE :** Logisk eller binær operator**FORMAT:** *udtryk* OR *udtryk*

**VIRKNING:** (logisk operator) OR anvendes til at skabe en logisk ELLER forbindelse mellem to udsagn, som enten kan være "sande" eller "falske". Afhængigt af de enkelte udsagns sandhedsværdi, vil den resulterende ELLER forbindelse antage værdien "sand" (-1) eller "falsk" (0). For programsætningen:

IF *udsagn1* OR *udsagn2* THEN *aktion...*

gælder følgende sandhedstabel:

<i>udsagn1</i>	falsk	falsk	sand	sand
<i>udsagn2</i>	falsk	sand	falsk	sand
-----				
OR-forbindelse	falsk	sand	sand	sand

Kun hvis begge udsagn er "falske", vil den resulterende værdi være "falsk", og aktionen (udtrykket) efter THEN vil ikke blive udført.

**EKSEMPEL** på anvendelse af OR som logisk operator:

```

100 PRINT"SVAR JA ELLER NEJ (J/N)!"
110 GET A$:IF A$="" THEN 110
120 IF A$="J" OR A$="j" THEN RETURN
130 IF NOT(A$="N" OR A$="n") THEN 110
140 END

```

Sammenlign de logiske operatorer AND (side 206) og NOT (side 242).

**VIRKNING: (binær operator)** Udfører en 16 bit binær OR-operation på værdier i området -32768 til 32767. Værdier udenfor dette område, vil resultere i fejlmeldingen ILLEGAL QUANTITY. Den binære OR-funktion kan beskrives således (1-bit operation):

udtryk	resultat
0 OR 0	0
0 OR 1	1
1 OR 0	1
1 OR 1	1

**EKSEMPEL på anvendelse af OR som binær operator:**

1045 POKE 53265,PEEK(53265)OR 32

Kommandoen indstiller på Commodore til normal højopløsning. Sammenlign de binære operatører AND (side 206) og NOT (side 242).

## PAINT

**TYPE :** Grafik kommando (+4)

**FORMAT:** PAINT *farvekilde,x,y,virkning*

**VIRKNING:** Udfylder en vilkårlig figur, som rummer punktet x,y (se SCALE) indenfor afgrænsningen. Virkningen bestemmes med værdierne:

- 0 Udfylder figuren der er tegnet med den udvalgte farvekilde (normal)
- 1 Udfylder figuren der er omgivet med en vilkårlig farvekilde\*

I andet eksempel afgrænser alle farvekilder (se COLOR på side 216), bortset fra baggrunden det område, der udfyldes. Ikke specificerede værdier erstattes af sidst anvendte værdier. Grafikcursorens position efterlades på positionen x,y. Specificeres intet koordinat, tages udgangspunkt i det punkt, hvor grafik-cursoren sidst blev efterladt.

**EKSEMPEL på anvendelse af PAINT kommandoen:**

```
100 REM UDFYLD CIRKEL
110 CIRCLE,160,100,65,50
120 PAINT,140,105:REM KAN ERSTATTES MED PAINT
```

**COMMODORE 64:** Se "PLOT"-kommandoen side 195.

**PEEK**

**TYPE** : Funktion - numerisk

**FORMAT**: PEEK(*numerisk udtryk*)

**VIRKNING**: Læser et byte fra den *adresse* i computerens hukommelse, som angives af det numeriske udtryk. Værdien skal ligge i området 0 til 65535. Værdier uden for området resulterer i fejlmeldingen ILLEGAL QUANTITY.

**EKSEMPEL på anvendelse af PEEK-funktionen:**

```
100 X=PEEK(53281):REM LÆSER BAGGRUNDSFARVEN (C64)
```

**POKE**

**TYPE** : Kommando

**FORMAT**: POKE *adresse, numerisk udtryk*

**VIRKNING**: Placerer en værdi på den specificerede *adresse* i hukommelsen. Er området dækket af ROM, skrives værdien til den underliggende RAM hukommelse. Adressen *skal* være i området 0 til 65535, og den indlæste værdi i området 0 til 255. Decimalværdier afrundes. Værdier udenfor det tilladte giver fejlmeldingen ILLEGAL QUANTITY.

**EKSEMPEL på anvendelse af POKE kommandoen:**

```
100 POKE 53280,1:REM SÆTTER RAMMEN TIL HVID (C64)
```

**POS**

**TYPE** : Funktion

**FORMAT**: POS(*dummy variabel*)

**VIRKNING**: Funktionen returnerer nummeret på den *kolonne*, hvorfra næste tekstudskrift vil starte (værdien 0 til 39). X er en *dummy variabel* - dvs. at værdien af X er uden betydning - værdien påvirkes ikke.

**EKSEMPEL på anvendelse af POS funktionen:**

```
100 IF POS(A)>35 THEN PRINT
```

**PRINT**

**TYPE :** Kommando

**FORMAT:** PRINT (*udtryk*)

**VIRKNING:** Udskriver en tekst på skærmen, startende fra det sted, hvor cursoren blev efterladt i sidste PRINT-sætning. Cursoren efterlades i starten af næste linje, medmindre PRINT-sætningen afsluttes med *semikolon* (cursoren efterlades efter sidste tegn) eller *komma* (cursoren efterlades på en efterfølgende tabel-kolonne - position 0, 8, 16, 24 eller 32).

**PRINT#**

**TYPE :** Kommando

**FORMAT:** PRINT#*lf*, *udtryk*, ...

**VIRKNING:** Kommandoen udskriver de specificerede udtryk til det *logiske filnummer* - *lf* -, der er specificeret tidligere i en OPEN kommando. Læs mere om anvendelsen i kapitlet om brug af disketter).

**PUDEF**

**TYPE :** Kommando (+4)

**FORMAT:** PUDEF "*1 til 4 tegn*"

**VIRKNING:** Definerer op til 4 symboler til brug for USING kommandoen. De fire tegn har følgende betydning:

- Position 1: Udfyldningskarakter. Normalt mellemrum, men kunne f.eks.

ændres til "\*" i stedet.

- Position 2: Komma. Kommategnet anvendes i det amerikanske talsystem til at indikere tusinder positioner - f.eks. 10,000. Kan f.eks. erstattes med et punktum her i landet.
- Position 3: Punktum. Anvendes i engelsktalende lande i stedet for vort decimalkomma. Kan passende erstattes med et komma.
- Position 4: Dollartegn. Kan f.eks. erstattes med et pundtegn, eller en mellemrumskarakter, hvis dollar-tegnet ikke anvendes.

**EKSEMPEL** på anvendelse af PUDEF kommandoen:

```
100 PUDEF "*., "
```

**COMMODORE 64:** Funktionen kan ikke simuleres på enkel måde. Kommandoen er ikke inkluderet i nogen BASIC udvidelse, der er forfatteren bekendt.

## RCLR

**TYPE :** Funktion (+4)  
**FORMAT:** RCLR(*farvekilde*)

**VIRKNING:** Returnerer farvekoden, der er anvendt i forbindelse med den specificerede *farvekilde*.

**COMMODORE 64:** Læs den pågældende tegnposition i farvehukommelsen, eller læs de aktuelle farveregistre (se afsnittet om grafik).

## RDOT

**TYPE :** Funktion (+4)  
**FORMAT:** RDOT(*specifikation*)

**VIRKNING:** Afhængigt af den specificerede værdi returneres grafikcursorens position på X-aksen - RDOT(0) - grafikcursorens position på Y-aksen - RDOT(1) - eller den *farvekilde*, der anvendes til punktets farve - RDOT(2).

**COMMODORE 64:** Funktionen simuleres i enkelte BASIC udvidelser.

**READ****TYPE** : Kommando**FORMAT**: READ *variabel*,...

**VIRKNING**: Indlæser værdier fra DATA-sætninger til variabler. Et forsøg på at indlæse tekst i en *numerisk variabel*, eller at indlæse en værdi udenfor det tilladte område for en *integer variabel* (-32768 til 32767) vil resultere i en fejlmelding. Se ellers kommandoen DATA på side 218.

**REM****TYPE** : REM**FORMAT**: REM *vilkaarligt indhold*

**VIRKNING**: En REM kommando fortæller computeren, at alle efterfølgende tegn er uden betydning for programmet. REM er en forkortelse for REMARK - bemærkning eller kommentar - og kommandoen anvendes udelukkende for opbevaring af kommentarer til programmøren.

**RENAME****TYPE** : Disk-kommando (+4)**FORMAT**: RENAME "*gammelt navn*" TO "*nyt navn*"RENAME "*gammelt navn*" TO "*nyt navn*",D*drive*,U*device*

**VIRKNING**: Ændrer navnet på en fil. Hvis *drev* eller *device* ikke specificeres, antager computeren, at der er tale om device 8, drev 0.

**COMMODORE 64**: Identisk med kommandoen:OPEN *lf,device,15,"Rdevice:nyt navn=gammelt navn"*

## RENUMBER

**TYPE :** Kommando (+4)

**FORMAT:** RENUMBER

RENUMBER *ny startlinje, trin, gl. startlinje*

**VIRKNING:** Renummerer et program. Kommandoen kan kun udføres som en direkte indtastning (se side 70). Alle kommandoer indeholdende linjenumre - f.eks. GOTO, GOSUB, THEN, RESTORE, ON...GOTO osv. - renummereres også. Møder RENUMBER kommandoen f.eks. en GOTO kommando, der peger mod et ikke-eksisterende linjenummer, fås en fejlmelding. Kommandoen anvendes således:

RENUMBER	Renummerer i trin på 10. Første linjenummer bliver 10.
RENUMBER 1000,10,100	Renummerer fra linje 100 i trin på 10. De nye
	Linje 100 bliver til linje 1000, linje 110 til 1010 osv.
RENUMBER , ,100	Renummerer i trin på 10 fra linje 100.

**COMMODORE 64:** Kommandoen er inkluderet i de fleste BASIC udvidelser. Vær opmærksom på, at kommandoen ikke virker efter hensigten i SIMON'S BASIC.

## RESTORE

**TYPE :** Kommando

**FORMAT:** RESTORE

RESTORE *linjenummer* (kun Plus/4)

**VIRKNING:** Placerer DATA-pointeren i starten af programmet eller på et bestemt linjenummer - kun Plus/4 og C16. Se DATA-kommandoen på side 218.

## RESUME

**TYPE :** Kommando (+4)

**FORMAT:** RESUME

RESUME NEXT

RESUME *linjenummer*



**VIRKNING:** Genoptager kørslen af et program efter en fejlsituation, som er opstået efter anvendelse af TRAP kommandoen. Den "nøgne" RESUME kommando vil forsøge at udføre den fejlbehæftede kommando een gang til, RESUME NEXT vil fortsætte programkørslen i den følgende kommando, og specificeres et linjenummer fortsætter programmet kørslen herfra. Et ikke-eksisterende linjenummer vil udløse en fejlmelding. Se TRAP kommandoen.

**COMMODORE 64:** En lignende kommando findes i en del BASIC udvidelser.

## RETURN

**TYPE :** Kommando  
**FORMAT:** RETURN

**VIRKNING:** Returnerer programkørslen til den kommando, som følger umiddelbart efter den GOSUB kommando, der kaldte subrutinen. Møder programmet en RETURN kommando, selvom ingen GOSUB kommando er udført fås fejlmeldingen RETURN WITHOUT GOSUB.

## RGR

**TYPE :** Funktion (+4)  
**FORMAT:** RGR(*dummy variabel*)

**VIRKNING:** Returnerer den grafik-kode, der blev anvendt i sidste GRAPHIC kommando.

**COMMODORE 64:** Har ingen mening på Commodore 64.

## RIGHT\$

**TYPE :** Funktion  
**FORMAT:** RIGHT\$(*strengudtryk, antal tegn*)

**VIRKNING:** Returnerer en streng indeholdende det specificerede antal tegn, optalt fra slutningen af strengen. Virker iøvrigt analogt til LEFT\$ kommandoen på side 238.

## RLUM

**TYPE :** Funktion (+4)

**FORMAT:** RLUM(*farvekilde*)

**VIRKNING:** Returnerer det luminansniveau (0-7), der benyttes i den specificerede farvekilde (0-4). Se også kommandoen GRAPHIC på side 233.

**COMMODORE 64:** Funktionen har ingen mening på Commodore 64.

## RND

**TYPE :** Funktion - numerisk

**FORMAT:** RND(*numerisk udtryk*)

**VIRKNING:** Funktionen returnerer en (pseudo)-tilfældig talværdi til anvendelse i spil, statistiske eksperimenter eller afprøvning af sandsynlighedsberegninger m.m. RND funktionen har tre udgaver:

- Positivt argument. Returnerer et tilfældigt tal i området 0 til 0,9999... Første gang funktionen anvendes bør den ("nulstilles") med formlen RND(-1), efter første indtastning. Hver efterfølgende RND funktion skal indeholde en positiv værdi - f.eks. 1. Det sikrer maksimal "tilfældighed".
- Argumentet nul. Det returnerede tilfældige tal er genereret af informationer læst fra computerens hardware clock.
- Negativt argument. Udgangspunktet for rækken af tilfældige tal vil være X i funktionen (-X). Hvis samme X-værdi anvendes hver gang, vil computeren levere samme række af "tilfældige" tal. Anvendes især til fremstilling af reproducerbare tests i statistiske analyser af forskellig art. Bør kun anvendes i spil, hvis man ønsker at snyde! Man kan jo lære "kortenes" rækkefølge udenad!

**RUN****TYPE** : Kommando**FORMAT**: RUNRUN *linjenummer*

**VIRKNING**: Starter programkørslen fra første programlinje eller et specificeret linjenummer. Eksisterer linjenummeret ikke, fås en fejlmelding. RUN kommandoen kan erstatte programlinjen:

1000 CLR:GOTO *linjenummer***SAVE****TYPE** : Kommando**FORMAT**: SAVESAVE "*programnavn*"(*,device,funktion*)>>

**VIRKNING**: Ved SAVE til kassettebåndoptager er det ikke nødvendigt at specificere device-nummer. Kommandoen behandles udførligt i kapitlet omhandlende brug af diskette på side 105 og i kapitlet om maskinkode.

**SCALE****TYPE** : Kommando (+4)**FORMAT**: SCALE 0 eller 1

**VIRKNING**: Abner mulighed for at ændre koordinatsystemet, der benyttes i de forskellige grafik skærme. Udføres kommandoen SCALE 1, vil koordinatsystemet gå fra 0 til 1023 i både X- og Y-retningen. Computeren omregner selv værdierne, til dem der skal anvendes i praksis. Udføres SCALE 0 kommandoen ser koordinatsystemerne således ud:

Skærmtype	X	Y
Multicolor grafik	0-159	0-199
do - split screen	0-159	0-159
Normal grafik	0-319	0-199
do -split screen	0-319	0-159

**COMMODORE 64:** Se "PLOT"-kommandoen side 195. SCALE kommandoen kan simuleres således:

```

100 X1=X:Y1=Y:REM RUTINEN KALDES FOR HVERT PLOT
110 IF NOT(SCALE) THEN 140:REM SCALE=0 ELLER 1
120 X1=160*X1/1024:REM X1=0-1023
130 Y1=200*Y1/1024:REM Y1=0-1023
140 X1=X1-160*INT(X1/160):REM 0-159
150 Y1=Y1-200*INT(Y1/200):REM 0-199
160 SYS plotrutine,X1,Y1
170 RETURN

```

## SCNCLR

**TYPE :** Kommandoen (+4)

**FORMAT:** SCNCLR

**VIRKNING:** Sletter skærmen - uanset om der er tale om grafik eller tekst.

**COMMODORE 64:** Se "PLOT"-kommandoen side 195.

## SGN

**TYPE :** Funktion - numerisk

**FORMAT:** SGN(*numerisk udtryk*)

**VIRKNING:** Returnerer "fortegnet" på det *numeriske udtryk*, der optræder i argumentet. Værdien er enten -1, 0 (nul) eller 1. Sammenlign funktionen ABS(X) på side 206.

**EKSEMPEL på anvendelse af SGN-funktionen:**

```
10 X=SGN(Y)
```

## SIN

**TYPE** : Funktion - numerisk

**FORMAT**: SIN(*numerisk udtryk*)

**VIRKNING**: Returnerer sinus til det *numeriske udtryk*, opgivet i *radianer*. Er beregningerne baseret på *grader* benyttes formlen:

$$X = \text{SIN}(\text{PI} * \text{grader} / 2)$$

Commodore BASIC indeholder ikke den inverse funktion. Den følgende formel klarer det problem:

$$Y = \text{ATN}(X / \text{SQR}(-X * X + 1))$$

**EKSEMPEL på anvendelse af SIN-funktionen:**

10 X=SIN(Y)

## SOUND

**TYPE** : Kommando (+4)

**FORMAT**: *stemme, frekvens, varighed*

**VIRKNING**: Kommandoen udløser en tone, baseret på en værdi mellem 0 og 1023 (*frekvens*) og en varighed på mellem 0 og 65535/60 sekunder. Lyddelen er udstyret med 3 stemmer - eller lydgeneratorer. Nr. 1 og 2 producerer toner, mens nr. 3 producerer støj (anvendes til simulering af slagstøj, eksplosioner osv.). Er BASIC i Gang med at udføre en tidligere SOUND kommando, venter maskinen til denne er færdig, før en ny kommando til *samme* stemme udføres!

**COMMODORE 64**: Tilsvarende kommandoer findes i de fleste BASIC udvidelser. Commodore 64 har dog langt mere omfattende muligheder end Plus/4 og C16.

**SPC**

**TYPE** : Funktion - numerisk  
**FORMAT**: SPC(*numerisk udtryk*)

**VIRKNING**: Benyttes i PRINT-sætninger til at overspringe o til 255 tegnpositioner.

**EKSEMPEL på anvendelse af SPC-funktionen:**

```
10 PRINT SPC(120):REM SPRINGER EKSACT 4 LINJER NED
```

**SQR**

**TYPE** : Funktion - numerisk  
**FORMAT**: SQR(*numerisk udtryk*)

**VIRKNING**: Beregner kvadratroden af udtrykket. SQR(Y) er den inverse funktion af  $X^2$  eller  $X*X$ .

**EKSEMPEL på anvendelse af SQR-funktionen:**

```
10 X=SQR(100):REM X=10
```

**SSHAPE**

**TYPE** : Grafik-kommando (+4)  
**FORMAT**: SSHAPE *strengvariabel*,X1,Y1,(X2,Y2)

**VIRKNING**: Kommandoen læser et område af grafik skærmen ind i en strengvariabel. Områdets øverste, venstre hjørne defineres af X1,Y1 og nederste, venstre hjørne er X2,Y2 (eller det sidste punkt, grafikkursoren befandt sig på). En strengvariabel kan maksimalt rumme 255 bytes, og dermed begrænses også det maksimale skærmareal, der kan lagres. Størrelsen beregnes således:

Multicolor :  $\text{INT}((\text{ABS}(X2-X1)+1)/4+.99)*(\text{ABS}(Y2-Y1)+1)+4$   
 Normal grafik:  $\text{INT}((\text{ABS}(X2-X1)+1)/8+.99)*(\text{ABS}(Y2-Y1)+1)+4$

Punktberegningen gælder for SCALE 0 kommandoen. Billedet lagres linje for linje, fra lo-byte til hi-byte. De sidste 4 bytes består af størrelsen i vandret henholdsvis lodret retning (begge minus 1). Det største billede vi kan indlæse kunne have et format på 32x49 punkter - altså væsentligt større end en sprite på Commodore 64!

Figuren kan placeres overalt på skærmen, og skrives tilbage til grafik- eller multicolor skærmen med kommandoen GSHAPE, der har formatet:

**GSHAPE strengvariabel,X,Y,virkning**

X,Y er øverste venstre hjørne i figuren. Udelades denne specifikation anvendes sidste grafikcursor position. Der findes ialt fire forskellige måder, at skrive figuren til skærmen på. Måden specificeres i parameteren *virkning*, og værdierne er:

- 0 Skriv figuren, som den er (normalt).
- 1 Skriv figuren i negativ\*.
- 2 Skriv figuren med en OR-funktion.
- 3 Skriv figuren med en AND-funktion.
- 4 Skriv figuren med en XOR-funktion.

Funktionen, der udføres sker bit for bit. Funktionen mærket med stjerne "vender" alle bit'ene. I multicolor er det også ensbetydende med et farveskift, idet to bits udgør een farve. F.eks. bliver 00 til 11, 01 til 10, 10 til 01 og 11 til 00. OR funktionen OR'er alle bits - dvs. at de bits, der er 1 på skærmen eller 1 i figuren giver resultatet 1. AND funktionen sætter et bit til værdien 1, hvis både skærmbit'et og figurbit'et er 1. XOR funktionen er helt speciel. Den muliggør fuldstændig genskabelse af billedinformationen, som den var før figuren blev skrevet til skærmen. XOR funktionen sætter et bit, hvis enten skærmbit'et eller figurbit'et har værdien 1, men ikke begge. I alle situationer sættes bit'et på skærmen til nul, hvis de foregående betingelser ikke er opfyldt.

**EKSEMPEL på anvendelse af G/SSHAPE kommandoen:**

```
100 SSHAPE A$,10,30,20,20:REM LÆS FIGUR
110 GSHAPE A$,10,20,4:REM SLET FIGUR
115 GSHAPE A$,50,75,4
120 FOR X=50 TO 250
130 : GSHAPE A$,X+1,75,4
140 : GSHAPE A$,X,75,4
150 NEXT X
```

Figuren flyttes henover skærmen.

**COMMODORE 64:** Simuleres med sprites. Se kapitlet om grafik.

**ST**

**TYPE** : STATUS variabel

**VIRKNING:** Virkningen er gennemgået i detaljer på siderne 117, 173RS-232 og 86

**STEP**

**TYPE** : Kommando

**FORMAT:** STEP *numerisk udtryk*

**VIRKNING:** Bestemmer den værdi, der opskrives med, hver gang NEXT kommandoen udføres i en FOR-NEXT løkke (se side 230).

**STOP**

**TYPE** : Kommando

**FORMAT:** STOP

**VIRKNING:** Stopper kørslen af et program. Kommandoen anvendes normalt kun under fejlfinding i programmer, og den indsættes på steder, hvor man ønsker at stoppe programmet for at undersøge indholdet i variablerne. Et færdigt program bør ikke indeholde STOP kommandoer (sammenlign END kommandoen side 228).

**STR\$**

**TYPE** : Funktion



**FORMAT:** STR\$(*numerisk udtryk*)

**VIRKNING:** Omdanner et numerisk udtryk til den tilsvarende ciffer-streng. Sammenlign VAL funktionen og CHR\$ funktionen på side 212.

**EKSEMPEL på anvendelse af STR\$ funktionen:**

100 Y\$=STR\$(A)

## SYS

**TYPE :** Kommando

**FORMAT:** SYS *adresse*

**VIRKNING:** Kaldet en maskinkode rutine, der starter på den specificerede adresse (se kapitlet om maskinkode).

## TAB

**TYPE :** Funktion

**FORMAT:** TAB(*numerisk udtryk*)

**VIRKNING:** Placerer næste udskrift i kolonnen specificeret i TAB. Har især betydning ved printerudskrifter o.l. På skærmen er funktionen identisk med SPC funktionen på side 256.

## TAN

**TYPE :** Funktion - numerisk

**FORMAT:** TAN(*numerisk udtryk*)

**VIRKNING:** Beregner *tangens* til en vinkel udtrykt i *radianer*. Opgives vinklen i *grader* benyttes følgende formel:

$X = \text{TAN}(PI * \text{grader} / 2)$

Funktionen ATN (se side 208) er den inverse funktion af TAN.

**EKSEMPEL** på anvendelse af TAN-funktionen:

10 X=TAN(Y)

## TI

**TYPE** : Kontrolvariabel

**FORMAT**: TI

**VIRKNING**: Variablen indeholder tiden for computerens indbyggede ur. Tiden måles i 1/60 sekund. Det er ikke lovligt, at læse værdier ind i variablen.

## TI\$

**TYPE** : Kontrolvariabel

**FORMAT**: TI\$

**VIRKNING**: Variablen kan tildeles en bestemt værdi (6 cifre i 24 timers formatet timer/minutter/sekunder). Variablens indhold opdateres automatisk hvert sekund.

**EKSEMPEL** på anvendelse af TI\$:

100 TI\$="235948"

110 PRINT TI\$;:GOTO 110

## TRAP

**TYPE** : Kommando (+4)

**FORMAT**: TRAP *linjenummer*

**VIRKNING:** Kommandoen fortæller computeren, at den skal springe til den specificerede linje, når fejlen opstår. Computeren stopper ikke, men forventer at programmøren selv undersøger, retter og meddeler evt. fejl. Anvendes TRAP uden linjenummer afbrydes funktionen. Opstår der fejl i den del af programmet, der skal løse evt. problemer i TRAP rutinen, stopper computeren. Det er nødvendigt, da man ellers kunne lande i en "uendelige løkke", der kun kan afbrydes med RESET!

I TRAP rutinen kan man læse fejlkoden - variabelen ER - fejlmeldingen - variabelen ER\$ og det linjenummer, hvori fejlen opstod - variabelen EL. F.eks. kunne rutinen udnytte værdierne således:

```
1000 TRAP 1000
... program
1000 IF ER=30 THEN PRINT"STOP VIRKER IKKE":RESUME
1010 PRINT ER$(ER);" FEJL I LINJE ";EL
1020 TRAP
1030 END
```

RESUME kommandoen benyttes, hvis man ønsker at fortsætte det program, hvori fejlen opstod, efter at man har rettet fejlen - ellers hjælper det jo ikke ret meget! Se side 250.

**COMMODORE 64:** En tilsvarende kommando findes i mange BASIC udvidelser.

## TROFF

**TYPE :** Kommando (+4)  
**FORMAT:** TROFF

**VIRKNING:** Afbryder virkningen af kommandoen TRON.

## TRON

**TYPE :** Kommando (+4)  
**FORMAT:** TRON

**VIRKNING:** Indkobler tracing funktionen, som på skærmen udskriver alle de linjenumre, som programmet gennemløber. Benyttes primært til test af løkker og om "løber" den rigtige vej.

**COMMODORE 64:** TRON og TROFF findes i mange BASIC udvidelser.

**UNTIL****TYPE :** Kommando (+4)**FORMAT:** UNTIL *betingelse*

**VIRKNING:** Bestemmer den betingelse, der skal være opfyldt, for at en DO-LOOP løkke udføres. Se DO kommandoen side 224.

**USING****TYPE :** Format kommando (+4)

**FORMAT:** PRINT USING "*formatstreng*"; *variabler*  
 PRINT#*lf*, USING "*formatstreng*"; *variabler*

**VIRKNING:** USING formatterer de specificerede variabler i overensstemmelse med parametrene i *formatstrengen*, som ikke må være en variabel. De to PRINT USING kommandoer opfylder samme mission, blot er den første udgave begrænset til skærmudskrifter. Formatstrengen kan indeholde følgende tegn (se også kommandoen PUDEF på side 247:

KARAKTER	STRENG NUMERISK	
# (hash)	ja	ja
+ (plus)	nej	ja
- (minus)	nej	ja
. (decimalpunktum)	nej	ja
\$ (dollar tegn)	nej	ja
^^^^ (4 pile op)	nej	ja
= (lighedstegn)	ja	nej
> (større end)	ja	nej

Betydningen og virkningen af anvendelsen af tegnene er:

- "#" tegnet kan anvendes i forening med både streng- og numeriske variabler. Tegnet optræder som fyldkarakter i formatstrengen - f.eks. "#####", som fortæller, at der kun udskrives maksimalt 6 tegn fra en variabel. Er der tale om en strengvariabel svarer virkningen til LEFT\$(variabel,6). En numerisk variabel afrundes først til et helt tal, og udskrives derefter fra højre mod venstre. Ledige pladser fyldes ud med mellemrum.

- Plus tegnet (eller minustegnet) anvendes til at bestemme, hvor et fortegn skal stå - foran eller efter talværdien. Der kan kun specificeres eet tegn - f.eks. "#####-", som vil resultere i, at alle talværdier skrives med fortegnet placeret til sidst. Bemærk, at anvendelsen af et "+" tegn "tvinger" fortegnet plus frem i udskriften. Anvendes et minus-tegn i formatfeltet, udskrives et mellemrum i stedet for et plus-tegn. Er der for mange cifre (incl. fortegn) til, at de kan rummes indenfor format-strengen erstattes den normale udskrift af en række stjerner "\*\*\*\*\*"!
- Komma og punktum anviser, hvordan cifrene skal grupperes henholdsvis foran og bag decimalkommaet - f.eks. som her, hvor "###,###.##" formaterer talværdier, så de altid indeholder 6 cifre - heraf to bag kommaet. Talværdien 999999.456 tildeles formatet "999,999.46".
- Anvender vi dollar-tegnet oplyses computeren om, at valutaen er dollars. Afhængigt af placeringen skrives dollar foran eller efter ciferværdien - f.eks. "\$###.##", der udskrives \$99,45, hvis variablen var 99,446.
- De fire opadgående pile benyttes til at indikere, at der ønskes eksponentiel notering af variablen. F.eks. vil "+#^^^^" resultere i, at variablen 23456E21 bliver omformet til "-2E8"
- I forening med formattering af strenge, anvendes "-" karakteren til at centrere teksten indenfor formatfeltet. ">" karakteren benyttes for at fremstille en højrejusteret tekst.

#### EKSEMPEL på anvendelse af USING kommandoen:

```
PRINT USING "##.#+", "-.01      = -0.01
PRINT USING "#,###.##+", 1944.44 = 1,944.44+
PRINT USING "###", 1000        = ***
```

**COMMODORE 64:** Kommandoen kan delvist efterlignes af med STR\$ kommandoen m.fl. Eksemplet konverterer værdien i den nymmeriske variabel i X til et 4 cifret tal, med to pladser efter kommaet, fortegn placeret bagest, og "DKR " foran.

```
90 IF X>=100 THEN X$="***** ":GOTO 190
100 X$=STR$(X):P=0
110 FOR N=LEN(X$) TO 1 STEP -1
120 : IF MID$(X$,N,1)="-." THEN P=N:N=0
130 NEXT N
140 IF P=0 THEN X$=X$+"00":GOTO 160
150 X$=LEFT$(X$+"00",P+2)
160 X$=RIGHT$(" "+X$,5)
170 IF SGN(X)=1 THEN X$=X$+"+"
180 X$=LEFT$(X$+"-",6)
190 PRINT "DKR ";X$:RETURN
```

**USR**

**TYPE** : Funktion - numerisk  
**FORMAT**: USR(*numerisk udtryk*)

**VIRKNING**: Funktionens anvendelse og muligheder er forklaret i detaljer på side 56.

**VAL**

**TYPE** : Funktion  
**FORMAT**: VAL(*strengudtryk*)

**VIRKNING**: Funktionen omdanner en ciffer-streng til en numerisk eller integer variabel. Eksisterer der ingen cifre i begyndelsen af strengen, returneres værdien 0 (nul). Sammenlign funktionen STR\$ på side 258.

**EKSEMPEL** på anvendelse af VAL funktionen:

1000 X=VAL(X\$)

**VERIFY**

**TYPE** : Kommando  
**FORMAT**: VERIFY  
VERIFY "*filnavn*", *device*

**VIRKNING**: VERIFY kommandoen sammenligner et program i computerens hukommelse med indholdet, der er lagret på diskette eller kassette. Er der ingen uoverensstemmelser, melder computeren "OK" - ellers udskrives fejlemdingen "VERIFYING ERROR". Benyttes VERIFY uden parametre, foretages sammenligningen automatisk med et program på kassettebåndoptager. Device nummeret kan være 1 (kassette) eller 8-15 (diskette - se side 105). VERIFY kommandoen behandles også i kapitlerne om maskinkode og MONITOR.

**EKSEMPEL på anvendelse af VERIFY kommandoen:**

**SAVE"PROGRAMNAVN",8:VERIFY"PROGRAMNAVN",8**

## **VOL**

**TYPE :** Kommando (+4)

**FORMAT:** VOL *lydstyrke*

**VIRKNING:** Indstiller lydstyrken til en værdi fra 0 (OFF) til 8 (fuld styrke). Lydstyrken gælder for alle tre generatorer.

**EKSEMPEL på anvendelse af VOL kommandoen:**

**LØØ VOL Ø:REM AAH, DEJLIGT!**

**COMMODORE 64:** En tilsvarende kommando er indbygget i de fleste BASIC udvidelser.

## **WAIT**

**TYPE :** Kommando

**FORMAT:** WAIT *adresse, værdi1, værdi2*

**VIRKNING:** En detaljeret gennemgang findes på side 16.

## **WHILE**

**TYPE :** Kommando (+4)

**FORMAT:** WHILE *betingelse*

**VIRKNING:** Bestemmer den betingelse, der skal være opfyldt, før en DO-LOOP udføres. Se DO kommandoen på side 224.

# A.1 Oversigt over Commodore 64 BASIC

Commodore 64 BASIC rummer færre ord end PLUS/4 BASIC. Benyttes denne opslags tabel, er det let at finde de BASIC-ord, som gælder for Commodore 64.

BASIC-ord	side	BASIC-ord	side
ABS	206	NEXT	242
AND	206	NOT	242
ASC	208	ON	243
ATN	208	OPEN	244,107
CHR\$	212	OR	244
CLOSE	213,119	PEEK	246
CLR	214	POKE	246
CMD	214,110	POS	246
CONT	216	PRINT	247
COS	218	PRINT#	247,109
DATA	218	READ	249
DEFFN	221	REM	249
DIM	222	RESTORE	250
END	228	RETURN	251
EXP	229	RIGHT\$	251
FN	230	RND	252
FOR-TO	230	RUN	253
FRE	230	SAVE	253
GET	231	SGN	254
GET#	231,113	SIN	255
GOSUB	232	SPC	256
GOTO	232	SQR	256
IF-THEN	235	ST	258,86,117,173
INPUT	235	STEP	258
INPUT#	236,111	STOP	258
INT	237	STR\$	258
LEFT\$	238	SYS	259
LEN	239	TAB	259
LET	239	TAN	259
LIST	239	TI	260
LOAD	240	TI\$	260
LOG	240	USR	264,56
MID\$	241	VAL	264
NEW	242	VERIFY	264
		WAIT	265,16



## A.2 Oversigt over DOS 5.1 kommandoer

- /programnavn**  
Normal LOAD - BASIC.
- %programnavn**  
Absolut LOAD - maskinkode.
- ^programnavn**  
Normal LOAD med efterfølgende RUN af programmet.
- <-programnavn**  
SAVE BASIC-program.
- <-@programnavn**  
SAVE BASIC-program med REPLACE.
- @**  
Læs fejlkanal.
- >\$**  
Læs directory (indhold).
- >#drivenr.**  
Skift til drive *drivenr.*
- >Q**  
Quit (forlad) DOS 5.1. Indkobling sker med SYS 52224.
- >NØ:diskettenavn,ID**  
Diskette-formatting.
- >NØ:diskettenavn**  
Sletning af BAM og directory.  
**SØ:filnavn**  
Sletning af fil. *Må ikke anvendes til at fjerne ikke CLOSE'de filer!*
- >RØ:nyt navn-gammelt navn**  
Ændring af filnavn.
- >CØ:kopinavn=originalnavn**  
Kopiering af en fil på samme diskette. *Ej relative filer.*
- >I**  
Indlæsning af diskettens BAM (sektorbelægning) i DOS'ens arbejdslager. Nødvendig, hvis flere disketter med samme ID benyttes.
- >V**  
Opdaterer diskettens sektor-belægning (BAM) ud fra de filer, der er opført i directory. *Random files* slettes fra BAM. Ikke CLOSE'de filer, fjernes fra directory.

## Appendix B

### BASIC fejlmeldinger

I det følgende præsenteres det komplette udvalg af fejlmeldinger, der kan optræde i Commodore BASIC 2.0 og 3.5. Selvom listen er omfattende, er der ingen grund til panik - enkelte af fejlmeldingerne vil snart blive lige så velkendte indslag i hverdagen, som TV-avisen, søndagskrydderen m.m.

Der er ingen grund til at frygte fejlmeldingerne. De er en fast del af enhver programmørs hverdag, hvadenten han er en rutineret veteran eller nybegynder. Begge laver fejl. Igen og igen. Ikke altid de samme, men alligevel fejl.

Problemerne med at finde fejlene er lige store for begynderen og den mere erfarne. Med rutine og erfaring bliver programmerne mere avancerede og dermed også fejlene!

Betragt ikke fejl som et nederlag - de er det nemlig ikke. Fejl er ikke andet end de kommunikationsvanskeligheder, der kan opstå når intelligente mennesker skal omforme kreative ideer, så en komplet tåbe - computeren - kan forstå dem. Prøv at forklare din hund, hvorfor den skal hente morgenavisen. Umuligt. Derimod kan det godt lade sig gøre at oplære hunden til at udføre dette hverv perfekt. Det kan tage lidt tid, og når hunden ikke vil aflevere avisen til dig de første par gange, er der ikke tale om andet end en fejlmelding. Dit træningsprogram var måske alligevel ikke helt fejlfrit. Tænk i den forbindelse på, at computeren ikke har mere intelligens end en amøbe eller en bakterie.

#### B.1 Der er forskel på fejl

Mange mener, at et program er fejlfrit, når computeren ikke kommer med nogen fejlmeldinger. Det er langt fra altid tilfældet! Lad os se på et eksempel.

```
100 INPUT"BELOB";BE
110 PRINT"PRIS EXCL. MOMS:";BE/X

200 X=1.222
210 INPUT"BELOB";BE
220 PRINT"PRIS EXCL. MOMS:";BE/X
```

Begge programmer indeholder fejl, men kun det første eksempel giver en

fejlmelding (X er ikke defineret, og sættes derfor til nul, og en division med nul .....). Det andet eksempel er langt farligere, idet fejlen ikke er åbenlys - eller rettere: fejlen kan være forbistret svær at finde i et program på 500, 1000 eller endnu flere programlinjer.

Der går mange historier om, at man kan få programmer, der kan finde fejl i andre programmer. Sådanne produkter findes, men de kan ikke afsløre en fejl, som den i linje 200 herover. Hvordan skulle et sådant fejlfindingsprogram vide, at vi har 22% moms i Danmark, og at værdien 1.222 egentlig skulle være 1.22 - eller at der overhovedet er tale om en momsberegning?

Det viste eksempel er meget enkelt, men med erfaringen kommer også indsigten - 90-95% af de fejl, vi laver, er enkle og indlysende, når vi har fundet dem.

Vær derfor altid forberedt på, at et fejlfrit program er lige så sjældent, som at finde en perle i en østers!

## B.2 BASIC Fejlmeldinger

Hver enkelt fejlmelding er udstyret med et nummer. På Commodore 64 anvendes disse numre kun internt, men på Commodore Plus/4 og C16 har nummereringen også betydning for programmøren (Se EL side 228, ER side 228, ERR\$ side 229, RESUME side 250 og TRAP side 260).

### 1. TOO MANY FILES

For mange åbne filer. Der må maksimalt være 10 åbne filer på een gang - incl. skærm og tastatur (2 filer). Se kommandoen OPEN side 244.

### 2. FILE OPEN

Filen, som forsøges åbnet er allerede åben - dvs. en fil med *samme logiske filnummer* er åben i forvejen! Se kommandoen OPEN side 244.

### 3. FILE NOT OPEN

Der er gjort forsøg på at læse fra eller skrive til en fil, der ikke forud er åbnet med et OPEN-statement - alternativt benyttes et forkert *logisk filnummer* i en PRINT# (Se side 247), CMD (Se side 214), INPUT# (Se side 236) eller GET# kommando (Se side 231). Se også kommandoen OPEN side 244.

### 4. FILE NOT FOUND

Ingen fil med det pågældende navn findes på disketten, eller end *end-of-file-marker* (EOF) er læst ind fra kassettebåndoptageren (se side 86).

### 5. DEVICE NOT PRESENT

Der er gjort et forsøg på at åbne en fil til et *device nummer*, som ikke eksisterer. Enten er der benyttet et forkert *device nummer* (se side 25), det kaldte *device* er ikke tilsluttet, eller der er ikke tændt for strømmen på det pågældende *device*.

6. NOT INPUT FILE

Der er gjort et forsøg på at læse fra et *device*, der er specificeret som output - f.eks. en printer. Læs mere om *device numre* på side 25).

7. NOT OUTPUT FILE

Der er gjort forsøg på at skrive til et *device*, der er specificeret som input - f.eks. tastaturet. Læs mere om *device numre* på side 25).

8. MISSING FILE NAME

En OPEN (se side 244), SAVE (se side 253) eller LOAD kommando (se side 240) er benyttet uden filnavn, hvor dette kræves - normalt til disk-station.

9. ILLEGAL DEVICE NUMBER

Der er anvendt et *device*-nummer, som ikke er tilladt i den pågældende kommando - f.eks. SAVE til skærm, printer eller RS-232, eller LOAD fra skærm, tastatur eller RS-232. Læs mere om *device numre* på side 25).

10. NEXT WITHOUT FOR

Computeren har mødt en NEXT kommando på "ulovlig" vis, f.eks. ved at der springes ind i en FOR-NEXT løkke, som i eksemplet herunder

```
100 GOTO 230
.....
220 FOR X=2 TO 20 STEP 4
230 PRINT X
240 NEXT X
```

BEMÆRK, at hvis der springes ud fra een løkke og ind i en anden, vil denne fejlmelding ikke vise sig, hvis der benyttes et NEXT, uden efterfølgende variabel. Benyt derfor altid formen NEXT XX i stedet for NEXT. Så er chancen for at en sådan fejl meldes langt større! Læs mere om løkker på side 43, FOR side 230, NEXT side 242 og STEP side 258.

11. SYNTAX ERROR

Computeren har mødt en kommando, som den ikke kan forstå - f.eks. en stavfejl i en kommando, manglende parenteser m.m.

12. RETURN WITHOUT GOSUB

Computeren har mødt en RETURN kommando på "ulovlig" vis, f.eks. fordi der er anvendt en GOTO kommando i stedet for GOSUB, som i eksemplet herunder:

```
100 GOTO 230
.....
230 PRINT X
```

## 240 RETURN

## 13. OUT OF DATA

Programmet har prøvet at udføre en READ kommando, men der er ikke flere DATA elementer. Læs mere om READ side 249, RESTORE på side 250 og DATA på side 218.

## 14. ILLEGAL QUANTITY

Den værdi, der benyttes i programmet overstiger det tilladte - f.eks. en værdi udenfor området 0 til 255 i funktionen CHR\$ (se side 212) eller en værdi udenfor området -32767 til + 32767 for heltalsvariabler (se side 35).

## 15. OVERFLOW

Resultatet - eller mellemresultatet! - i en udregning overstiger det maksimalt tilladelige, nemlig  $1.701411833 \times 10^{38}$  (se side 34).

## 16. OUT OF MEMORY

Computerens hukommelse er fyldt - f.eks. på grund af dimensionering af et for stort array (se DIM side 222) - overvej evt. brugen af et integer array, i stedet for et almindeligt numerisk array (læs mere herom på side 37).

## 17. UNDEF'D STATEMENT

Der henvises til et linjenummer, som ikke eksisterer i programmet, f.eks. vil GOTO 105 (side 232), GOSUB 105 (side 232) eller RUN 105 (side 253) give denne fejlmelding i følgende program:

```
100 X=0.22*Y
```

```
110 PRINT"MOMS:";Y
```

På Commodore Plus/4 og C16 vil kommandoerne RESTORE 105 (side 250 eller RESUME 105 (side 250) give samme fejlmelding.

## 18. BAD SUBSCRIPT

Programmet har forsøgt at læse et element, som befinder sig udenfor det område, der er defineret i en DIM kommando - eller hvis DIM kommandoen, ikke har været anvendt, har det været forsøgt, at læse et element med et nummer højere end 10 (læs mere om DIM side 222 eller arrays side 36).

## 19. REDIM'D ARRAY

Programmet har mødt en DIM kommando med et variabelnavn, der allerede er anvendt ved en tidligere dimensionering. Redimensionering af arrays kan kun ske efter sletning af variabelhukommelsen - f.eks. med CLR eller RUN linjenummer kommandoen. Redimensionering tillades ikke i compilere! (Læs mere om DIM side 222 eller arrays side 36).

## 20. DIVISION BY ZERO

Division med nul - kan f.eks. skyldes fejl i variabelnavn eller en variabel, som ikke er tildelt en værdi (se side 35).

## 21. ILLEGAL DIRECT

Der er foretaget direkte indtastning af en kommando, som kun kan udføres i et program - se GET side 231, GET# side 231,

GETKEY side 232 , INPUT side 235 og INPUT# side 236 samt side 70..

**22. TYPE MISMATCH**

Computeren forventer andre data til en variabel, f.eks. fordi en numerisk variabel sættes lig en tekstvariabel - f.eks. som i de følgende eksempler (se også side 34):

```
100 A$=AB:XY="DETTE ER OGSAA FORKERT"  
110 A(23,1,1)=BD$  
120 IF A=B$ THEN ....
```

**23. STRING TOO LONG**

Der er gjort forsøg på at skabe en strengvariabel eller et element i et streng-array, der vil resultere i en længde på mere end 255 tegn - evt. som et mellemresultat i en større udregning! Ved indlæsning i variable fra kassette, diskette eller RS-232 indgangen, optræder fejlen, hvis der gøres forsøg på at indlæse mere end 88 karakterer. Se også INPUT# side 236 samt siderne 34 og 112).

**24. FILE DATA**

Der er konstateret fejl i de data, der er indlæst fra kassettebåndoptageren (se side 84).

**25. FORMULA TOO COMPLEX**

Den anvendte formel er for indviklet for computeren - anvend færre parenteser eller del formlen op i to udtryk.

**26. CAN'T CONTINUE**

CONT kommandoen (se side 216) er benyttet efter en fejlmelding, efter, at der er rettet i programmet, eller før programmet har været kørt (RUN).

**27. UNDEF'D FUNCTION**

Programmet refererer til brugerdefineret funktion, som endnu ikke er defineret. Definitionen skal altid udføres før den brugerdefinerede funktion kan benyttes. En compiler kræver yderligere, at placeringen også rent fysisk befinder sig forud i programmet - dvs. har lavere linjenummer end selve funktionskaldet. Se også DEF FN side 221 og FN side 230).

**28. VERIFY**

Computeren har konstateret forskelle mellem programmet i hukommelsen, og det program, der befinder sig på bånd eller diskette. Læs mere om VERIFY på side 264 og 105.

**29. LOAD**

Der er problemer med indlæsningen af et program. Prøv igen.

**30. BREAK**

Programmet er stoppet med STOP-knappen eller med en STOP-kommando (se side 258). Fejlnummeret har kun betydning på Commodore Plus/4 og C16.

## B.3 Ekstra fejlmeldinger Commodore Plus/4

## 31. CAN'T RESUME

Computeren har mødt en RESUME kommando (se side 250) før en TRAP kommando er udført (side 260. Se også EL (side 228), ER (side 228) og ERR\$ (side 229).

## 32. LOOP NOT FOUND

Programmet har mødt en DO kommando, men kan ikke finde den tilhørende LOOP kommando. Læs mere om løkker på side 43, DO side 224, EXIT side 229, LOOP side 241, UNTIL side 262 og WHILE side 265.

## 33. LOOP WITHOUT DO

Programmet har mødt en LOOP kommando, men ingen aktive DO kommandoer findes, f.eks. på grund af et ulovligt spring ind i en DO...LOOP løkke - se side 43.

## 34. DIRECT MODE ONLY

Programmet har mødt en kommando, som ikke kan udføres i et program, men kun som en direkte indtastning. Se side 70, AUTO side 209, HELP side 234 og RENUMBER side 250.

## 35. NO GRAPHICS AREA

Programmet har mødt en grafik kommando, før grafik-skærmen er defineret - se GRAPHIC side 233.

## 36. BAD DISK

Der er opstået en fejl under formattering af en diskette - enten fordi HEADER kommandoen (se side 234 og side 122) benyttes uden ID på en diskette, der ikke er formatteret, eller fordi disketten er defekt.

## B.4 Commodore 1541 fejlmeldinger

Se side 120 om læsning af fejlkanalen, samt de reserverede variabler DS (side 226) og DS\$ (side 227 på Commodore Plus/4 og C16.

Fejlkoderne i området 2 til 19 og over 74 anvendes ikke.

00 OK Ingen fejl.

01 FILES SCRATCHED NN 00

NN er antallet af slettede filer (maksimalt 144).

- 20 READ ERROR** Block header findes ikke.  
Disk controlleren kan ikke finde header'en på den datablok, der ønskes læst. Enten er header'en beskadiget eller en ulovlig sektor er specificeret.
- 21 READ ERROR** Ingen sync karakter.  
Disk controlleren kan ikke finde synkroniseringsmærket for den pågældende sektor. Misjustering af hovedet eller brugen af en uformatteret eller beskadiget diskette. Der kan muligvis også være tale om en fejl/defekt i disk-stationen.
- 23 READ ERROR** Checksum fejl i datablok  
Der er afvigelser i et eller flere databytes. Kan også tyde på, at det er nødvendigt at sørge for en jordforbindelse til disk-stationen.
- 24 READ ERROR** Byte fejl under dekodering  
Afvigelser i det forventede bitmønster indikerer hardware fejl eller problemer med jordforbindelsen.
- 25 WRITE ERROR** Write-verify fejl  
Controlleren har konstateret afvigelser mellem de bytes den har skrevet og indholdet i en af diskbufferne. Normalt er det et tegn på en beskadiget diskette, men hardware-fejl kan også være årsagen.
- 26 WRITE PROTECT ERROR**  
Disketten er forsynet med slettebeskyttelse.
- 27 READ ERROR** Checksum fejl i headeren  
Controlleren har konstateret afvigelser i den header, der går forud for de egentlige databytes. Beskadiget diskette, snarere hoved eller jordproblemer kan være årsagen.
- 28 WRITE ERROR** Lang data blok  
Sync mærket er ikke dukket op indenfor det fastlagte tidsinterval. Fejlen kan skyldes snarere hoveder, dårlig formattering, beskadiget diskette eller hardware fejl.
- 29 DISK ID MISMATCH**  
Disketten er initialiseret eller har en dårlig eller beskadiget header.
- 30 SYNTAX ERROR** Generel syntax fejl  
Disk-stationen kan ikke forstå kommandoen, den har modtaget.
- 31 SYNTAX ERROR** Ugyldig kommando  
Disk-stationen kan ikke forstå selve kommandoen.
- 32 SYNTAX ERROR**  
Kommandoen rummer for mange karakterer.
- 33 SYNTAX ERROR**  
Ugyldigt filnavn - det er ikke tilladt at anvende jokere, som del af et filnavn ved SAVE eller LOAD.
- 34 SYNTAX ERROR**  
Filnavnet mangler i en kommando - typisk skyldes det et manglende



kolon i kommandoen.

### 39 SYNTAX ERROR

Ugyldig kommando. Disk-stationen genkender ikke kommandoen.

### 50 RECORD NOT PRESENT

Typisk skyldes det, at en GET# eller INPUT# kommando har forsøgt at læse "forbi" sidste post i en fil. Fejlmeldingen modtages også, når man for første gang prøver at skrive til en relativ post, der ikke har været aktiveret forud. I dette tilfælde kan fejlmeldingen ignoreres.

### 51 OVERFLOW IN RECORD

Der er gjort forsøg på at skrive flere karakterer til en post (PRINT#), end posten er specificeret til at rumme. Overskydende karakterer smides bort. (Husk, at RETURN m.fl. kontrolkarakterer også tæller med i postens længde).

### 52 FILE TOO LARGE

Der er gjort forsøg på at skrive til en post i en relativ fil, som ikke kan rummes på disketten.

### 60 WRITE FILE OPEN

Der er gjort forsøg på at åbne en fil for læsning, selvom den allerede er åben for skrivning (USR, PRG og SEQ). Husk altid at lukke filer, når de ikke skal anvendes mere.

### 61 FILE NOT OPEN

Der er gjort forsøg på at læse eller skrive til en fil, der ikke er åbnet. Fejlmeldingen er sjælden, idet disk-stationen normalt ignorerer den slags "tusserier".

### 62 FILE NOT FOUND

Filen eksisterer ikke på disketten i det specificerede drive.

### 63 FILE EXISTS

Filnavnet eksisterer allerede på disketten (filtypen er underordnet). Optræder normalt fordi man glemmer at anvende replace i en OPEN eller SAVE kommando.

### 64 FILE TYPE MISMATCH

Filnavnet stemmer, men filtypen afviger - f.eks. fordi man har forsøgt at åbne en programfil som en sekventiel fil.

### 65 NO BLOCK

Den blok, man forsøger at reserve med B-A kommandoen er allerede belagt. De returnerede værdier efter fejlteksten viser den næste frie sektor med samme eller højere track-nummer. Returneres værdierne nul, bør man prøve en B-A på track 1, sektor 0, inden man opgiver!

### 66 ILLEGAL TRACK AND SECTOR

Disk-stationen har forsøgt at læse en blok, der ikke eksisterer - tyder normalt på fejl i pointeren til den næste blok.

### 67 ILLEGAL SYSTEM TRACK AND SECTOR

Illegal system track og sektor.

**70 NO CHANNEL**

Den ønskede kanal står ikke til rådighed (random files), eller alle kanaler er optagede.

**71 DIRECTORY ERROR**

Der er problemer med BAM'en - f.eks. fordi den er blevet overskrevet i computerens interne hukommelse. Foretag en ny Initialize af disketten. Enkelte filer kan være beskadigede!!

**72 DISK FULL**

Der er ikke mere plads på disketten.

**73 DOS MISMATCH (73 CBM DOS V2.6 1541)**

Man har forsøgt at skrive til en diskette, der er formatteret i et afvigende format. Når disk-stationen tændes, kan ovennævnte meddeles læses fra kommandokanalen.

**74 DRIVE NOT READY**

Ingen diskette i disk-stationen, eller for mange åbne filer, eller andre soft-errors.

## Appendix C

## Commodore 1541 disk-format

## C.1 1541 BAM format

byte	indhold	beskrivelse
0	18	næste track
1	1	næste sektor
2	65	ASCII karakter A
3	0	Til fremtidig brug
4-143	--	BAM bit-map for track 1 til 35
144-161	--	Navnet, udfyldt med CHR\$(160)
162-163	--	ID-kode (2 tegn)
164	160	
165-166	50/65	Tegnene 2A (DOS-version
167-168	160/160	
171-255	--	Anvendes ikke

## C.2 Directory opbygning

byte	indhold
0	næste track
1	næste sektor
2-31	Fil-optegnelse 1
34-63	Fil-optegnelse 2
66-95	Fil-optegnelse 3
98-127	Fil-optegnelse 4
130-159	Fil-optegnelse 5
162-191	Fil-optegnelse 6
194-223	Fil-optegnelse 7
226-255	Fil-optegnelse 8

### C.3 En enkelt optegnelse i directory

byte	indhold	beskrivelse
0	128+type          +64	Filtyperne er: 0 = en slettet fil 1 = SEQ - sekventiel fil 2 = PRG - program fil 3 =USR - user fil 4 = RRL - relativ fil Slettebeskyttelse
1	track	spornummeret på første blok i filen
2	sektor	sektornummer på første blok i filen
3-18	filnavn	filnavnet kan bestå af op til 16 tegn. Ikke belagte felter er udfyldt med SHIFT SPACE - karakternummer 160 eller hex A0.
19	track	spornummer for første <i>side sector block</i> i relative filer.
20	sektor	sektornummer for første <i>side sector block</i> i relative filer.
21	størrelse	antal bytes i hver <i>record</i> i relative filer.
22-25		anvendes ikke
26	track	spornummer for første blok i en fil, der erstatter den bestående. Se opdatering af en fil.
27	sektor	sektornummer for første blok i en fil, der erstatter den bestående ved opdatering.
28	lo-byte	antal blokke, som filen beslaglægger
29	hi-byte	



# FORKORTELSER FOR BASIC NØGLEORD

Kom-mando	Forkor-telse	ser således ud på skærm	Kom-mando	Forkor-telse	ser således ud på skærm
ABS	A <b>SHIFT</b> B	A	END	E <b>SHIFT</b> N	E
AND	A <b>SHIFT</b> N	A	EXP	E <b>SHIFT</b> X	E
ASC	A <b>SHIFT</b> S	A	FN	NONE	FN
ATN	A <b>SHIFT</b> T	A	FOR	F <b>SHIFT</b> O	F
CHR\$	C <b>SHIFT</b> H	C	FRE	F <b>SHIFT</b> R	F
CLOSE	CL <b>SHIFT</b> O	CL	GET	G <b>SHIFT</b> E	G
CLR	C <b>SHIFT</b> L	C	GET#	NONE	GET#
CMD	C <b>SHIFT</b> M	C	GOSUB	GO <b>SHIFT</b> S	GO
CONT	C <b>SHIFT</b> O	C	GOTO	G <b>SHIFT</b> O	G
COS	NONE	COS	IF	NONE	IF
DATA	D <b>SHIFT</b> A	D	INPUT	NONE	INPUT
DEF	D <b>SHIFT</b> E	D	INPUT#	I <b>SHIFT</b> N	I
DIM	D <b>SHIFT</b> I	D	INT	NONE	INT

Kom-mando	Forkor-telse	ser således ud på skærm	Kom-mando	Forkor-telse	ser således ud på skærm
LEFT\$	LE <b>SHIFT</b> F	LE	RIGHT\$	R <b>SHIFT</b> I	R
LEN	NONE	LEN	RND	R <b>SHIFT</b> N	R
LET	L <b>SHIFT</b> E	L	RUN	R <b>SHIFT</b> U	R
LIST	L <b>SHIFT</b> I	L	SAVE	S <b>SHIFT</b> A	S
LOAD	L <b>SHIFT</b> O	L	SGN	S <b>SHIFT</b> G	S
LOG	NONE	LOG	SIN	S <b>SHIFT</b> I	S
MID\$	M <b>SHIFT</b> I	M	SPC(	S <b>SHIFT</b> P	S
NEW	NONE	NEW	SQR	S <b>SHIFT</b> Q	S
NEXT	N <b>SHIFT</b> E	N	STATUS	ST	ST
NOT	N <b>SHIFT</b> O	N	STEP	ST <b>SHIFT</b> E	ST
ON	NONE	ON	STOP	S <b>SHIFT</b> T	S
OPEN	O <b>SHIFT</b> P	O	STR\$	ST <b>SHIFT</b> R	ST
OR	NONE	OR	SYS	S <b>SHIFT</b> Y	S
PEEK	P <b>SHIFT</b> E	P	TAB(	T <b>SHIFT</b> A	T
POKE	P <b>SHIFT</b> O	P	TAN	NONE	TAN
POS	NONE	POS	THEN	T <b>SHIFT</b> H	T
PRINT	?	?	TIME	TI	TI
PRINT#	P <b>SHIFT</b> R	P	TIME\$	TI\$	TI\$
READ	R <b>SHIFT</b> E	R	USR	U <b>SHIFT</b> S	U
REM	NONE	REM	VAL	V <b>SHIFT</b> A	V
RESTORE	RE <b>SHIFT</b> S	RE	VERIFY	V <b>SHIFT</b> E	V
RETURN	RE <b>SHIFT</b> T	RE	WAIT	W <b>SHIFT</b> A	W



## BASIC 3.5 FORKORTELSER





















NØGLEORD	FORKORTEELSE	TYPE
ABS	a SHIFT B	funktion - numerisk
ASC	a SHIFT S	function - numerisk
ATN	a SHIFT T	funktion - numerisk
AUTO	a SHIFT U	kommando
BACKUP	b SHIFT A	kommando
BOX	b SHIFT O	udtryk
CHAR	ch SHIFT A	udtryk
CHR\$	c SHIFT H	funktion - streng
CIRCLE	c SHIFT I	udtryk
CLOSE	cl SHIFT O	udtryk
CLR	c SHIFT L	udtryk
CMD	c SHIFT M	udtryk
COLLECT	col SHIFT L	kommando
COLOR	co SHIFT L	udtryk
CONT	c SHIFT O	kommando
COPY	co SHIFT P	kommando
COS	none	fuktion - numerisk
DATA	d SHIFT A	udtryk
DEC	none	funktion - numerisk
DEFFN	d SHIFT E	udtryk
DELETE	de SHIFT L	kommando
DIM	d SHIFT I	udtryk
DIRECTORY	di SHIFT R	kommando
DLOAD	d SHIFT L	kommando
DO	none	udtryk
DRAW	d SHIFT R	udtryk
DSAVE	d SHIFT S	kommando
END	e SHIFT N	udtryk
ERR\$	e SHIFT R	funktion - streng
EXP	e SHIFT X	funktion - numerisk










NØGLEORD	FORKORTEELSE		TYPE
FOR	f	SHIFT O	udtryk
FRE	f	SHIFT R	funktion - numerisk
GET	g	SHIFT E	udtryk
GETKEY	getk	SHIFT E	udtryk
GET#		none	udtryk
GOSUB	go	SHIFT S	udtryk
GOTO	g	SHIFT O	udtryk
GRAPHIC	g	SHIFT R	udtryk
GSHAPE	g	SHIFT S	udtryk
HEADER	he	SHIFT A	kommando
HEX\$	h	SHIFT E	funktion - streng
IF...GOTO		none	udtryk
IF...THEN...ELSE		none	udtryk
INPUT		none	udtryk
INPUT#	i	SHIFT N	udtryk
INSTR	in	SHIFT S	funktion - numerisk
INT		none	funktion - numerisk
JOY	j	SHIFT O	funktion - numerisk
KEY	k	SHIFT E	kommando
LEFT\$	le	SHIFT F	funktion - streng
LEN		none	funktion - numerisk
LET	l	SHIFT E	udtryk
LIST	l	SHIFT I	kommando
LOAD	l	SHIFT O	kommando
LOCATE	lo	SHIFT C	udtryk
LOG		none	funktion - numerisk
LOOP	lo	SHIFT O	udtryk
MID\$	m	SHIFT I	funktion - streng
MONITOR	m	SHIFT O	udtryk
NEW		none	kommando
NEXT	n	SHIFT E	udtryk
ON...GOSUB	on...go	SHIFT S	udtryk
ON...GOTO	on...g	SHIFT O	udtryk
OPEN	o	SHIFT P	udtryk
PAINT	p	SHIFT A	udtryk
PEEK	p	SHIFT E	funktion - numerisk
POKE	p	SHIFT O	udtryk
POS		none	funktion - numerisk
PRINT	?		udtryk
PRINT#	p	SHIFT R	udtryk
PRINT USING	?us	SHIFT I	udtryk
PUDEF	p	SHIFT U	udtryk
RCLR	r	SHIFT C	funktion - numerisk







































NØGLEORD	FORKORTEELSE			TYPE
RDOT	r	SHIFT	D	funktion - numerisk
READ	r	SHIFT	E	udtryk
REM			none	udtryk
RENAME	re	SHIFT	N	kommando
RENUMBER	ren	SHIFT	U	kommando
RESTORE	re	SHIFT	S	udtryk
RESUME	res	SHIFT	U	udtryk
RETURN	re	SHIFT	T	udtryk
RGR	r	SHIFT	G	funktion - numerisk
RIGHT \$	r	SHIFT	I	funktion - streng
RLUM	r	SHIFT	L	funktion - numerisk
RND	r	SHIFT	N	funktion - numerisk
RUN	r	SHIFT	U	kommando
SAVE	s	SHIFT	A	kommando
SCALE	sc	SHIFT	A	udtryk
SCNCLR	s	SHIFT	C	udtryk
SCRATCH	sc	SHIFT	R	kommando
SGN	s	SHIFT	G	funktion - numerisk
SIN	s	SHIFT	I	funktion - numerisk
SOUND	s	SHIFT	O	udtryk
SPC	s	SHIFT	P	funktion - special
SQR	s	SHIFT	Q	funktion - numerisk
SSHAPE	s	SHIFT	S	udtryk
STatus	st	SHIFT	A	reserveret - numerisk variabel
STOP	s	SHIFT	T	udtryk
STR\$	st	SHIFT	R	funktion - streng
SYS	s	SHIFT	Y	udtryk
TAB(	t	SHIFT	A	funktion - special
TAN			none	funktion - numerisk
TI			none	reserveret - numerisk variabel
TI\$			none	reserveret - streng variabel
TRAP	t	SHIFT	R	udtryk
TROFF	tro	SHIFT	F	udtryk
TRON	tr	SHIFT	O	udtryk
UNTIL	u	SHIFT	N	udtryk
USR	u	SHIFT	S	funktion - special
VAL			none	funktion - numerisk
VERIFY	v	SHIFT	E	kommando
VOL	v	SHIFT	O	udtryk
WAIT	w	SHIFT	A	udtryk
WHILE	w	SHIFT	H	udtryk

# APPENDIX E

## SKÆRM KODER



















SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	C	c	3	F	f	6
A	a	1	D	d	4	G	g	7
B	b	2	E	e	5	H	h	8
I	i	9	%		37		A	65
J	j	10	&		38		B	66
K	k	11	,		39		C	67
L	l	12	(		40		D	68
M	m	13	)		41		E	69
N	n	14	*		42		F	70
O	o	15	+		43		G	71
P	p	16	,		44		H	72
Q	q	17	—		45		I	73
R	r	18	.		46		J	74
S	s	19	/		47		K	75
T	t	20	0		48		L	76
U	u	21	1		49		M	77
V	v	22	2		50		N	78
W	w	23	3		51		O	79
X	x	24	4		52		P	80
Y	y	25	5		53		Q	81
Z	z	26	6		54		R	82
[		27	7		55		S	83
£		28	8		56		T	84














SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
]		29	9		57		U	85
↑		30	:		58		V	86
←		31	;		59		W	87
SPACE		32	<		60		X	88
!		33	=		61		Y	89
"		34	>		62		Z	90
#		35	?		63			91
\$		36			64			92























































SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
		93			105			117
		94			106			118
		95			107			119
SPACE		96			108			120
		97			109			121
		98			110			122
		99			111			123
		100			112			124
		101			113			125
		102			114			126
		103			115			127
		104			116			









# APPENDIX E

## ASCII OG CHR\$ KODER

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
	0		17	"	34	3	51
	1		18	#	35	4	52
	2		19	\$	36	5	53
	3		20	%	37	6	54
	4		21	&	38	7	55
	5		22	.	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
DISABLES  	8		25	*	42	;	59
ENABLES  	9		26	+	43		60
	10		27	,	44	=	61
	11		28	-	45		62
	12		29	.	46	?	63
	13		30	/	47	@	64
	14		31	0	48	A	65
	15		32	1	49	B	66
	16	!	33	2	50	C	67

D	68		97		126	Gray 3	155
E	69		98		127		156
F	70		99		128		157
G	71		100	Orange	129		158
H	72		101		130		159
I	73		102		131		160

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	 141			170
T	84		113	 142			171
U	85		114		143		172
V	86		115	 144			173
W	87		116	 145			174
X	88		117	 146			175
Y	89		118	 147			176
Z	90		119	 148			177
[	91		120	Brown	149		178
£	92		121	Lt. Red	150		179
]	93		122	Grey 1	151		180
↑	94		123	Grey 2	152		181
←	95		124	Lt. Green	153		182
	96		125	Lt. Blue	154		183

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
	184		186		188		190
	185		187		189		191

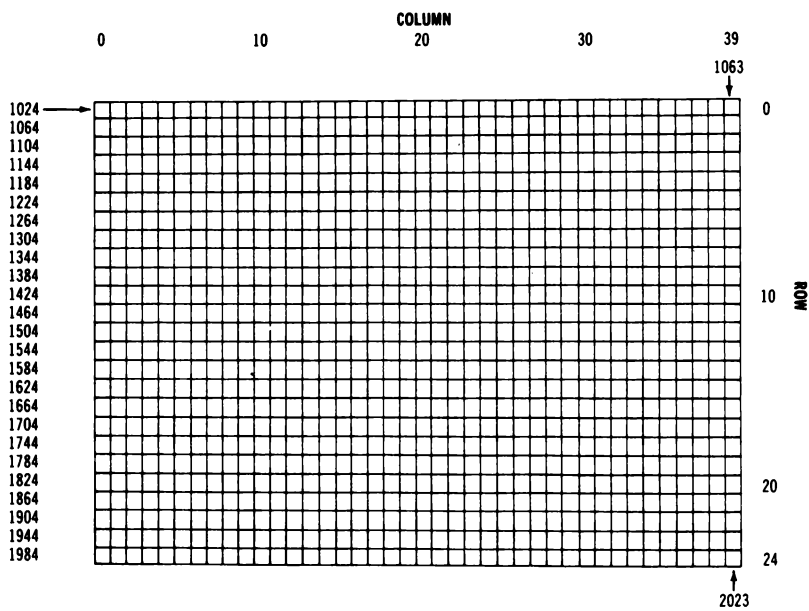
**CODES**  
**CODES**  
**CODE**

**192-223**  
**224-254**  
**255**

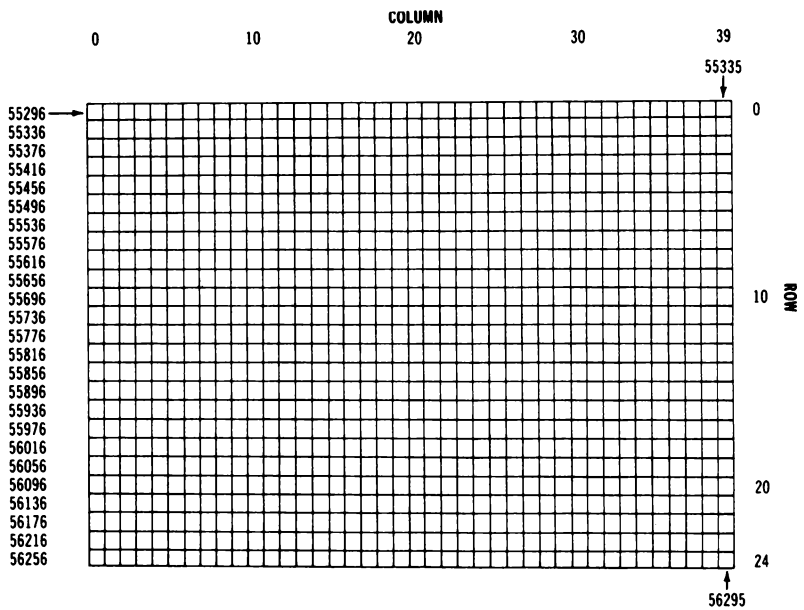
**SAME AS**  
**SAME AS**  
**SAME AS**

**96-127**  
**160-190**  
**126**

OVERSIGT OVER SKÆRM-LAGER



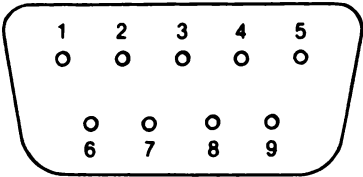




BENFORBINDELSER FOR INPUT/OUTPUT

Control Port 1

Pin	Type	Note
1	JOYA0	MAX. 50mA
2	JOYA1	
3	JOYA2	
4	JOYA3	
5	POT AY	
6	BUTTON A/LP	
7	+5V	
8	GND	
9	POT AX	



Control Port 2

Pin	Type	Note
1	JOYB0	MAX. 50mA
2	JOYB1	
3	JOYB2	
4	JOYB3	
5	POT BY	
6	BUTTON B	
7	+5V	
8	GND	
9	POT BX	

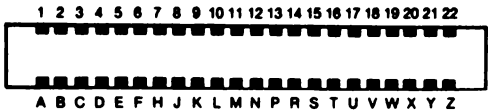
**Cartridge Expansion Slot**

Pin	Type
22	GND
21	CD0
20	CD1
19	CD2
18	CD3
17	CD4
16	CD5
15	CD6
14	CD7
13	DMA
12	BA

Pin	Type
Z	GND
Y	CA0
X	CA1
W	CA2
V	CA3
U	CA4
T	CA5
S	CA6
R	CA7
P	CA8
N	CA9

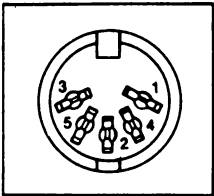
Pin	Type
11	ROML
10	1/02
9	EXROM
8	GAME
7	1/01
6	Dot Clock
5	CR/W
4	IRQ
3	+5V
2	+5V
1	GND

Pin	Type
M	CA10
L	CA11
K	CA12
J	CA13
H	CA14
F	CA15
E	S02
D	NMI
C	RESET
B	ROMH
A	GND



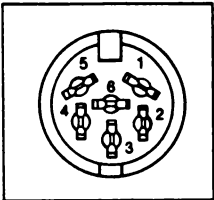
**Audio/Video**

Pin	Type	Note
1	LUMINANCE	
2	GND	
3	AUDIO OUT	
4	VIDEO OUT	
5	AUDIO IN	



**Serial I/O**

Pin	Type
1	SERIAL SRQIN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	RESET



## STIKORDSREGISTER

A - fil-kommando 139  
ABS - funktion 206  
Afrundingsfejl m.m. 33  
AND - binær tabel 207  
AND - operator 206  
AND - sandhedstabel 206  
Antal åbne disk-kanaler 108  
APPEND - disk-kommando 139  
APPEND af BASIC programmer 68  
APPEND via COPY kommandoen 124  
Arrayhukommelse 67  
Arrays 36  
Arrays - pladskrav 37  
ASC - funktion 208  
ASCII-værdi 208  
Assembler 51  
ATN - funktion 208  
AUTO - kommando 209  
Auto repeat tastatur 14  
Avanceret INPUT i BASIC 29

B-A - block allocate kommando 154  
B-E - block execute kommando 155  
B-F - block free kommando 154  
B-P - buffer pointer kommando 153  
B-R - block read kommando 153  
B-W - block write kommando 153  
Back-up 96, 98  
BACKUP - disk-kommando 135, 210  
Backup af programmer på 1541 125  
BAD DISK ERROR 273  
BAD SUBSCRIPT ERROR 271  
BAM 98, 137  
BAM diskettens indholdsfortegnelse 122  
BAM initialisering 123  
BASIC ROM 68, 68  
BASIC program laver BASIC program 72  
Baudrate, RS-232 166  
Behandling af disketter 96  
Beskadigede disketter 97  
Betingede hop 41  
Binære operatorer 40  
Binære tal 51  
Bit 51  
Block allocate kommando 154  
Block Availability Map 122  
Block kommandoer 152  
Blok 95  
Bottom of BASIC pointer 67  
BOX - grafik-kommando 210  
BREAK-melding 272  
Brug af flere disk-stationer 160  
Brugerdefinerede funktioner 221, 230  
Brugerdefinerede tegn 17  
Brugerdefineret tastatur 22  
Byte 51

C - disk-kommando 124  
 CAN'T CONTINUE ERROR 272  
 CAN'T RESUME ERROR 273  
 Carriage return 170  
 Centronics interface 177  
 Chaining af programmer 138  
 CHAR - grafik-kommando 211  
 CHR\$ - funktion 212  
 CIRCLE - grafik-kommando 212  
 CLOSE - kommando 27, 119, 169, 213  
 CLOSE - maskinkode 63  
 CLR - kommando 214  
 CMD - kommando 110, 170, 214  
 COLLECT - disk-kommando 135, 215  
 COLOR - grafik-kommando 216  
 Commodore 1541 89  
 Commodore 64 BACKUP-program 125  
 Commodore 64 grafik 186  
 Computerens hukommelse 66  
 CONT - kommando 216  
 COPY - disk-kommando 124, 135, 217  
 COS - funktion 218  
 CTRL-knap og listninger 99  
 Cursor on/off fra BASIC 25  
 Cursor styring i maskinkode 59  
 Cursor-styring fra BASIC 24  
  
 DATA - kommando 72, 218  
 Datakanaler til disk 107  
 Datakommunikation via RS-232 170  
 DEC - funktion 220  
 Decimal til hextal - programeksempel 207  
 DEF FN - definitions-kommando 221  
 Device 25  
 DEVICE NOT PRESENT - disk 119  
 DEVICE NOT PRESENT ERROR 269  
 Device nummer 25  
 Device og maskinkode 60  
 Device typer 25  
 Devicenummer - disk 90  
 DIM - kommando 222  
 Direct disk access 152  
 Direct mode 70  
 DIRECT MODE ONLY ERROR 273  
 Directory 98  
 DIRECTORY - disk-kommando 100, 135, 223  
 Directory og fil-udvalg 103  
 Direkte indtastning 70  
 Direkte kommando 70  
 DISK ADDR CHANGE - program 160  
 Disk kommandoer 121  
 Disk program header 63, 137  
 Disk BASIC - forskelle og ligheder 89  
 Disk kommando - maksimal længde 125  
 Disk kommandoer - Plus/4 135  
 Disk operationer på sektor-niveau 152  
 Diskette-typer 95  
 Diskettens inddeling 94  
 Disketter og kassetter - forskelle 93  
 Diskoperativsystem 89, 98

DIVISION BY ZERO ERROR 271  
 DLOAD - disk-kommando 106, 224  
 DO - kommando 224  
 DOS 5.1 161  
 DOS 5.1 "%" kommando 161, 267  
 DOS 5.1 "/" kommando 161, 267  
 DOS 5.1 "<-" kommando 161, 267  
 DOS 5.1 "<@" kommando 161, 267  
 DOS 5.1 ">#" kommando 161, 267  
 DOS 5.1 ">\$" kommando 161, 267  
 DOS 5.1 ">C" kommando 162, 267  
 DOS 5.1 ">I" kommando 162, 267  
 DOS 5.1 ">N" kommando 162, 267  
 DOS 5.1 ">Q" kommando 162, 267  
 DOS 5.1 ">R" kommando 162, 267  
 DOS 5.1 ">S" kommando 162, 267  
 DOS 5.1 ">V" kommando 163, 267  
 DOS 5.1 "@" kommando 161, 267  
 DOS 5.1 "^" kommando 161, 267  
 DOS 5.1 - læsning af fejlkanal 161, 267  
 DOS 5.1 LOAD - kommando 161, 267  
 DOS 5.1 LOAD - kommando, maskinkode 161, 267  
 DOS 5.1 re-initialisering 162  
 DOS-wedge 161  
 DRAW - grafik-kommando 226  
 DS - kontrolvariabel 121, 226  
 DS\$ - kontrolvariabel 121, 227  
 DSAVE - disk-kommando 105, 227  
  
 EL - kontrolvariabel 228  
 END - kommando 228  
 End-of-file marker 65  
 End-of-program 67  
 EOF indikering, STATUS 87  
 EOF marker 65  
 EOI indikering, STATUS 118  
 ER - kontrolvariabel 228  
 ERR\$ - kontrolvariabel 229  
 EXIT - kommando 229  
 EXP - funktion 229  
 Extended color tekst 188  
  
 Farvehukommelse for tekst 186  
 Farvekontrol på tekstskræmen 186  
 Fejlkanal - Commodore 64 120  
 Fejlkanal - Commodore Plus/4 121  
 Fejlkanal - læsning 120, 161  
 Fejlmeldinger - BASIC 268  
 Fejlmeldinger Commodore 1541 273  
 Fejlmeldinger Commodore Plus/4 273  
 Fil-pointer og relative filer 142  
 FILE DATA ERROR 272  
 FILE NOT FOUND ERROR 269  
 FILE NOT OPEN ERROR 269  
 FILE OPEN ERROR 269  
 Filer 98  
 Filer - navngivning 101  
 Filtyper 102  
 Floating-point akkumulator 57  
 FN - numerisk funktion 230

FOR..TO..(STEP..) - kommando 230  
 Forgreninger 41  
 Forkortet indtastning 72  
 Formattering 95, 122, 210, 234, 273  
 FORMULA TOO COMPLEX ERROR 272  
 FRE - funktion 230  
 Full duplex 169  
 Full-screen editor 13  
 Funktioner 38  
 Fysisk skærmlinje 70

Garbage collection 44, 46, 67  
 GET - kommando 26, 231  
 GET# - kommando 27, 29, 113, 137, 170, 231  
 GETKEY - kommando 232  
 GOSUB - kommando 232  
 GOTO - kommando 232  
 Grafik på Commodore 64 186  
 GRAPHIC - kommando 233  
 Grænseflade 27  
 GSHAPE - grafik-kommando 233  
 Gyldige device numre 26

Half duplex 169  
 HEADER - disk-kommando 135, 234  
 HELP - kommando 234  
 Heltals-arrays 36  
 Heltalsvariabler 34  
 HEX\$ - funktion 234  
 Hexadecimale tal 54  
 Hexadecimale tal - hvorfor 55  
 Hukommelse, RAM og ROM 32  
 Højopløsningsgrafik 192

I - disk-kommando 123  
 IF..THEN.. - kommando 235  
 ILLEGAL DEVICE NUMBER ERROR 270  
 ILLEGAL DIRECT ERROR 271  
 ILLEGAL QUANTITY ERROR 271  
 IN/OUTPUT i maskinkode 60  
 Indbygget ur-funktion 260  
 Indekserede variabler 36  
 Indlæsning af maskinkode 106, 161, 267  
 Indlæsning af nul karakter fra disk 113  
 Initialisering af BAM 123  
 INITIALIZE - disk-kommando 123  
 INPUT - kommando 26, 29, 30, 235  
 Input buffer 70, 70  
 INPUT med default værdier 29  
 INPUT og kontrol karakterer 30  
 INPUT# - kommando 27, 70, 111, 137, 170, 236  
 Input/output kredse 69  
 INSTR - funktion 236  
 INT - funktion 237  
 Integer arrays 36  
 Integer variabler 34  
 Intelligent disk-station 89  
 Interaktive programmer 27  
 Interface 27  
 Interrupts, skærmbetinget 201

Jokere 100  
JOY - funktion 237

Kassettebåndoptager 83  
KERNAL ROM 25, 68, 68  
KERNAL ROM rutiner 57  
KEY - kommando 238  
Kommunikation via user porten 164  
Kompatible disk-stationer 95  
Kontrol-karakterer 14  
Kopi-program, enkelt 1541 125  
Kopiering af ROM til RAM 23

Lagring af maskinkode 66  
LEFT\$ - funktion 238  
LEN - funktion 239  
LET - kommando 239  
Line editor 13  
Line feed 170  
Line-feed og OPEN - kommandoen 108, 170  
LIST - kommando 239  
Listning til disk 110  
LOAD "\$",8 99  
LOAD - kommando 99, 106, 240  
LOAD - kommando, maskinkode 63, 106, 161, 267  
LOAD ERROR 272  
LOAD program under BASIC ROM 64  
LOCATE - kommando 240  
LOG - funktion 240  
Logisk skærmlinje 70  
Logiske operatorer 40  
Lokale tegn fra tastaturet 17  
LOOP - kommando 241  
LOOP NOT FOUND ERROR 273  
LOOP WITHOUT DO ERROR 273  
Lukning af en disk-kanal 119  
Lukning af kanel - maskinkode 63  
Lukning af RS-232 kanal 169  
Lyspen 201  
Lyspen og interrupts 201  
Læsning af directory 99, 161  
Læsning af directory - program 114  
Læsning af fejlkanal 120, 161  
Læsning af filer 107  
Læsning af ikke-lukkede filer 140  
Læsning af joystick i maskinkode 75  
Læsning af skærmens indhold 25  
Læsning fra disk 111  
Læsning fra kanal - maskinkode 61  
Læsning fra skærm 27  
Løkker 43  
Låsning af tastatur 15

M - fil-kommando 140  
M-E kommando 156  
M-R kommando 156  
M-W kommando 156  
Maskinkode 49  
Maskinkode - Commodore Plus/4 78



- Maskinkode - ind/udlæsning fra devices 60
- Maskinkode - LOAD 63
- Maskinkode - lukning af kanal 63
- Maskinkode - læsning fra kanal 61
- Maskinkode - placering 69
- Maskinkode - ROM rutiner 60, 60
- Maskinkode - skrivning til kanal 62
- Maskinkode - VERIFY 63
- Maskinkode - åbning af kanal 61
- Maskinkode lagring 67, 67
- Maskinkode monitor 78
- Maskinkode og BASIC 56
- Maskinkode SAVE 65
- Maskinkode til DATA sætninger 72
- Matematiske beregninger i maskinkode 58
- Matricer 36
- Memory execute kommando 156
- Memory kommandoer 156
- Memory read kommando 156
- Memory write kommando 156
- MID\$ - funktion 241
- Mini-disketter 95
- MISSING FILE NAME ERROR 270
- Mnemonics 51
- MOB grafik 186
- Monitor 51, 78
- MONITOR - kommando 78, 241
- Monitor kommandoer -Commodore Plus/4 78
- Multicolor grafik 194
- Multicolor spritegrafik 192
- Multicolor tekst 187
  
- N - disk-kommando 122
- Navngivning af filer 124
- NEW - disk-kommando 122
- NEW - kommando 67, 99, 161, 242
- NEXT - kommando 242
- NEXT WITHOUT FOR ERROR 270
- NO GRAPHICS AREA ERROR 273
- Normal spritegrafik 189
- NOT - operator 242
- NOT INPUT FILE ERROR 270
- NOT OUTPUT FILE ERROR 270
- Numeriske arrays 36
- Numeriske variabler 34
  
- ON - kommando 243
- ON ... GOSUB 41
- ON ... GOTO 41
- OPEN - kommando 26, 107, 169, 244
- OPEN - maskinkode 61
- OPEN med replace 109
- Operativsystem 25
- Operator-hieraki 38
- Operatorer 38
- Optageformat - kassette 84
- OR - binær tabel 245
- OR - operator 244
- OR - sandhedstabel 244
- OUT OF DATA ERROR 271



RESTORE - udkobling 15  
 RESUME - kommando 250  
 RETURN - kommando 251  
 RETURN WITHOUT GOSUB ERROR 270  
 RGR - funktion 251  
 RIGHT\$ - funktion 251  
 RLUM - funktion 252  
 RND - funktion 252  
 ROM memory 32  
 ROM rutiner 60  
 RS-232 - baudrate 166  
 RS-232 - parity 166  
 RS-232 - stopbits 166  
 RS-232, CLOSE - kommando 169  
 RS-232, OPEN - kommando 169  
 RUN - kommando 253  
 Rutiner i KERNAL ROM 57

S - disk-kommando 123  
 Sammenligninger 39  
 SAVE - kommando 104, 253  
 SAVE i maskinkode 65  
 SAVE med replace 105, 123, 161  
 SCALE - kommando 253  
 SCNCLR - kommando 254  
 SCRATCH - disk-kommando 123, 135  
 Scrolling, hardware 199  
 Sector adressering 152  
 Sektor 95  
 Sektor pointer 136, 137, 140  
 Sekventiel læsning af programfiler 137  
 Sekventielle filer 83, 136  
 Seriell kommunikation 92  
 SGN - funktion 254  
 SIN - funktion 255  
 Skrivning til disk 109  
 Skrivning til kanel - maskinkode 62  
 Skrivning til relative filer - pas på! 151  
 Skærm, specielle funktioner 199  
 Skærmbetingedede interrupts 201  
 Skærmen 13  
 Skærmformater, specielle 200  
 Sletning af variable 35  
 Slettebeskyttelse - diskette 157  
 Slettebeskyttelse - program 157  
 Slukning af skærmen 200  
 SOUND - kommando 255  
 SPC - funktion 256  
 Specielle skærmformater 200  
 Specielle skærmfunktioner 199  
 Specifikationer 1541 90  
 Spor 94  
 Sprite adresse 189  
 Sprite aktivering 189  
 Sprite farve 190  
 Sprite kollision 191  
 Sprite koordinater 190  
 Sprite størrelse 189, 190  
 Sprite/tekst prioritet 190  
 Spritegrafik 188

SQR - funktion 256  
 SSHAPE - kommando 256  
 ST - kontrolvariabel 86, 117, 173, 258  
 Start-of BASIC 67  
 STATUS 258  
 STATUS variabel og disk 117  
 STATUS variabel og RS-232 173  
 STEP - kommando 258  
 STOP - kommando 258  
 Stop-bits, RS-232 166  
 STR\$ - funktion 258  
 Streng arrays 36  
 Strengvariabler 34  
 STRING TOO LONG ERROR 272  
 SYNTAX ERROR ERROR 270  
 SYS - kommando 56, 259  
 Systemhukommelse 66

TAB - funktion 259  
 Talsystemer 51  
 TAN - funktion 259  
 Tape header 63, 85  
 Tastatur - læsning fra BASIC 16  
 Tastatur buffer 13  
 Tastatur buffer - i egne programmer 13  
 Tastatur forandringer 17  
 Tastatur med autorepeat 14  
 Tastaturet 13  
 TEST/DEMO-diskette programmer - DISK ADDR CHANGE 160  
 TI - kontrolvariabel 260  
 TI\$ - kontrolvariabel 172, 260  
 Tidsmåling 260, 260  
 TIME-OUT - disk 118  
 Tokenizing 70, 71  
 TOO MANY FILES ERROR 269  
 Top of BASIC pointer 67  
 Tracks 94  
 TRAP - kommando 260  
 TROFF - kommando 261  
 TRON - kommando 261  
 TYPE MISMATCH ERROR 272

U1 - block read kommando 153  
 U2 - block write kommando 153  
 Udnyttelse af RAM-området under ROM 19  
 UN-NEW - program 35  
 UNDEF'D FUNCTION ERROR 272  
 UNDEF'D STATEMENT ERROR 271  
 UNTIL - kommando 262  
 Ur-funktion 260, 260  
 User port 164  
 USING - kommando 262  
 USR - funktion 56, 264

V - disk-kommando 123  
 V - kommando, monitor 106  
 Variabelhukommelse 67  
 VAL - funktion 264  
 Valg af filtyper 103  
 VALIDATE - disk-kommando 123

Variabelnavne 34  
Variabeltyper 34  
Variabler 34  
Variabler - defaultværdier 35  
VERIFY - kommando 105, 264  
Verify - monitor 106  
VERIFY af maskinkode programmer 106  
VERIFY ERROR 106, 272  
VERIFY i maskinkode 63  
VOL - kommando 265

WAIT - kommando 16, 265  
WHILE - kommando 265  
Wildcards 100  
Write protect 97, 157

Ændring af filnavn 124

Abning af en disk-kanal 107  
Abning af kanel - maskinkode 61  
Abning af random fil 152  
Abning af RS-232 kanal 169  
Abning med replace 109  
Abning og lukning af en kanal 26











# HVORFOR HAR HELE FAMILIEN BRUG FOR EN COMMODORE COMPUTER?

Fordi langt de fleste familier i dag er tvunget til at leve med besværligheder som husholdningsbudget, selvangivelse, terminsydelser, banklån, checkkonto og pensionsordning.

Fordi en Commodore Computer gør det sjovere og enklere at holde styr på det hele.

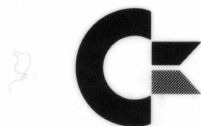
Fordi alle gerne vil have det maximale ud af deres penge. Og sikre sig, at der også bliver overskud til fornøjelser.

Fordi en Commodore Computer hurtigt og professionelt regner ud, om det overhovedet kan betale sig at få naturgas installeret. Eller om det er god økonomi at købe en brændeovn. Eller om man bør udskifte bilen med en ny og mere benzinøkonomisk model. Eller om det på langt sigt er en fordel at isolere - bare for at give nogle eksempler.

Fordi det er så betydningsfuldt for hele familien Danmark at have kendskab til edb. Nu og i fremtiden.

Fordi du med en Commodore Computer f. eks. også kan komponere musik. Lære engelsk. Eller matematik. Eller noget helt andet. Og så er Commodore Computer i øvrigt oplagt til leg og spil.

Derfor er det måske ikke kun en del af familien, der har brug for en Commodore Computer.



**Commodore**

***Fordi fremtiden forlængst  
er begyndt.***