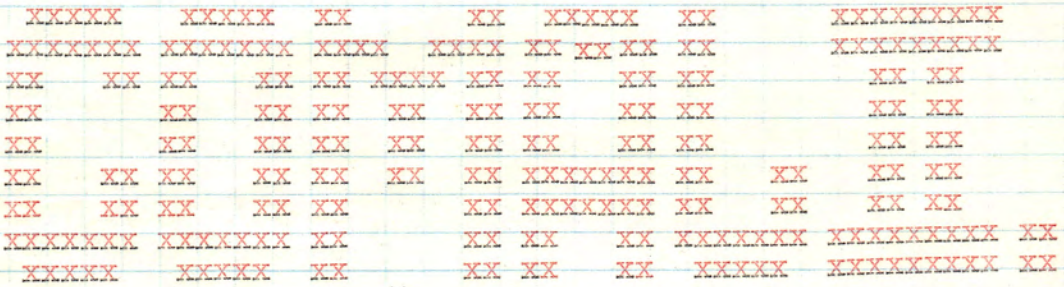
Hurtig referencebog i:



Skænket Vestfyens Gymnasium
– ihvertfald indtil videre.

1977
by Normann Aa. Nielsen

Forord.

Denne bog er blevet til i ønsket om en dansksproget referencebog i sproget COMAL II, og mere specielt til netop det hardware, der findes på Vestfyens Gymnasium. Referencebogen kan benyttes som supplemering til en eventuel edb-undervisning i BASIC, det være sig BATCH, EXTENDED eller - som her - COMAL. Bogen er udpræget specialiseret omkring netop det hardware (RC 7ooo) og software (skreen, lineprinter, cardreader & puntcher), der findes; dette ses tydeligt af en række specialartikler i bogen.

Bogens artikler står i den orden, som de fremkom i under skrivningen. Derved er selve indholdet ikke stillet op i en logisk orden, men man kan få et overblik over indholdet ved at se indholdsfortegnelsen, der skulle stå i alfabetisk rækkefølge. Angående specieltegn, der ikke findes på normale skrivemaskiner, er disse skrevet i hånden med kuglepen. Tallet nul, der benyttes flere steder, er skrevet i notationen (o) eller (Ø). Ved brug af maskinen er det vigtigt at notere sig, at tegnene ÆØÅ ikke findes på tastaturet. Skulle der i de eksempler, der er vist i bogen, være indsneget sig nogle af disse tegn, rettes disse blot til hhv. Æ AE, Ø eller AA.

Bogens indholdsfortegnelse må tages med det forbehold, at der i årenes løb kan indkomme nye artikler, dette sker uden forudgående meddelelse. Visse ting i bogen kan være udtrykt sprogligt forkert, andet kan stå almindeligt uklart, men man kan til enhver tid spørge forfatteren om datamatiske problemer i det hele taget. Skulle forfatteren ikke være til stede, evt. være dimiterende, kan adjunk Anders Crone spørges.

I tillæget især må fremhæves - lad mig bare kalde ham for konsulent - Bent Brun Kristensen, der er forfatteren til stor hjælp, hvad angår programkørsler især, og "udforskning" af maskinelle muligheder i almindelighed. Det er klart, at man også kan spørge ham, hvis man har problemer, men også hos ham er den mulighed, at han på et tidspunkt dimiterer. Under alle omstændigheder må jeg anbefale ham som en stærk kapacitet inden for BASIC, også specielt på RC 7ooo udstyret.

Skrevet 1977 af
Normann Aa. Nielsen p.t.
Bodebjergvej 29
562o Glamsbjerg

&
"Konsulent"
Bent Brun Kristensen p.t.
Skolevej 17
5572 Tommerup

Normann Aa. Nielsen

Bent Brun.

Register.

K: Kommandoord (anvendes udenfor program)
S: Sætningsord (anvendes indenfor program)

Andet

God fornøjelse !

Normann Aaboe Nielsen

1977

Bemærk: Sine steder kan tasten [ESC] være udtrykt som en *.
Da adjunk Anders Crone nu (1978) har forladt gymnasiet, må henvendelser om datamaten rettes til adjunk Finn Juul, eller - ind til videre (1979) - til Bent Brun eller Normann Aaboe Nielsen.

Bogen må ikke fjernes fra EDB-rummet uden min eller lærernes tilladelse.

Behandling af programmel og maskiner.

Tastaturet bør kun behandles af højst én person af gangen.
Tasterne bør ikke rokkes ved, rives i, eller hamres på....

Af tasterne (tryktasterne) på skærmens front bør kun TAPE aktiveres, undtagelser, se under TILLÆG TIL HURTIG REFERENCEBOG bagerst i bogen. ON/OFF kontakten sidder på siden af skærmen og på front af monitoren, ligesom lyskontrollerne sidder tilsvarende steder. Venligst, lad skærmens kontakt stå på ON, når maskinen forlades, da skærmen er ret langsom til at varme op.

Det bedes om, at man ikke afbryder datamaten, når denne forlades. Under normale omstændigheder, dvs. under almindelig kørsel, bør ingen kontakt på datamatens panel benyttes. Dette kan medføre stop af maskinen med muligvis efterfølgende ødelæggelse af softwaren, dvs. sproget.

Fremføringsskruen på linieskriverens side bør kun benyttes i det tilfælde, at papiret sidder i klemme. Skruen skal i så tilfælde trækkes ud fra linieskriveren, idet man drejer på den (man kan høre det, hvis man ikke har gjort det...).
Skal papir fremføres, gøres dette ved følgende procedure: Der kontrolleres, om øverste lampe på linieskriveren lyser. Gør den det, tastes trykknappen SELECT SWITCH, og lyset slukkes. Derefter tastes den sorte trykknap TOP OF FORM, og papiret ruller et stykke frem. Ønskes mere papir, tastes endnu en gang på TOP OF FORM. Endelig tastes igen SELECT SWITCH, og proceduren er slut.
Er linieskriveren slukket, tastes ON/OFF og SELECT SWITCH. Maskinen er da klar til at skrive (selvfølgelig, skal man ikke bruge skriveren, lader man den bare være...).

Klargørelse af hulkortlæser til læsning: Bag på læseren sidder alle de nødvendige kontakter til start. ON/OFF kontakten - der sidder øverst i højre hjørne - vippes opad, og omskifteren PUNCH/OPT.MARK skal stå i stilling OPT.MARK (undtagen hvis BATCH-BASIC skal indlæses). Omskifteren umiddelbart over førnævnte omskifter skal ~~ikke~~ stå på REMOTE. Derefter ilægges kortene, og der tastes RESET på fronten af læseren. Maskinen er nu klar.

Hulstrimler til strimmellæseren skal lægges som angiven på enhedens top. Så vidt muligt påses det, at hulstrimlerne under fremførsel ikke kommer i klemme, ligesom det påses, at "flossede" strimler ikke bliver revet i stykker - de kan efterlade små papirsstykker, der kan forhindre både en fremføring og en læsning af hulstrimlerne.
Klargørelse af strimmellæser til læsning: Hulstrimmel lægges i som angivet, det vil ofte være en fordel at "banke" lidt til den, så man er sikker på, at den lægges helt ind i læserens "gab". Under dette skal den sorte vippearm være oppe. Herefter lægges den sorte vippearm ned, og knappen RESET på læseren tastes. Strimmelen skal nu blive ført et stykke frem. Sker dette ikke, er strimmelen lagt forkert i, og vippearmen hæves for omlægning af hulstrimmelen.
Strimmellæseren er tilkoblet datamatens strømforsyning, og er derfor startet, når datamat er startet.

Hulstrimmelskriveren startes ved at taste ON/OFF på toppen af maskinen, derefter aktivere vippekontakten herunder.

I det hele taget: Behandel apparaturet med almindelig omhu, og meddel, når der er noget, der synes at være i vejen. Anlæget står i en pris af ca. 150.000 kr., såe...

2a

OPSTART

Opstart fra slukket tilstand 1 (Nøgle på OFF):

1) Nøgle drejes på ON
2) Kontakterne RESET og START aktiveres i denne rækkefølge
3) Nøgle på LOCK
4) Maskinen er klar.

Opstart fra slukket tilstand 2 (Kontakt på væg slukket):

1) Tænd for kontakten på væggen.
2) Starter maskinen ikke, check da nøglens stilling efter. Den skal stå på LOCK
3) Hvis lampen yderst til venstre på maskinen ikke blinker, følg da opstart fra slukket tilstand nr. 1.
4) Maskinen er klar.

Opstart fra blokeret tilstand:
1) Nøgle drejes på ON
2) Kontakter 12 og 14 vippes op, resten ned.
3) BINÆR LOADER (lille, sort strimmel) i strimmellæser.
4) RESET på strimmellæser
5) Kontakterne RESET og PROGRAM LOAD aktiveres i denne rækkefølge
6) Alle adressekontakter (talkontakter) op
7) COMAL II SINGEL USER (stor, grøn strimmel) i læseren
8) RESET på strimmellæser
9) RESET og START kontakterne aktiveres i denne rækkefølge
10) Adressekontakter fra og med Ø (nul) til og med 7 ned
11) Kontakterne RESET og START aktiveres i denne rækkefølge
12) På skærmen kommer der nu en spørgetekst med følgende udseende (brugers svar er indtastet med rødt):

SOS COMAL II LANGUAGE (STRUCTURED BASIC) REV.
UDGIVET AF A/S REGNECENTRALEN I SAMARBEJDE MED

DATAAFDELINGEN VED TONDER STATSSEMINARIUM.

VAERTSSPROG: DATA GENERAL SOS X. BASIC RE.

FULL ERROR MESSAGE (TEXT) 1=YES, Ø=NO: 1 car ret.

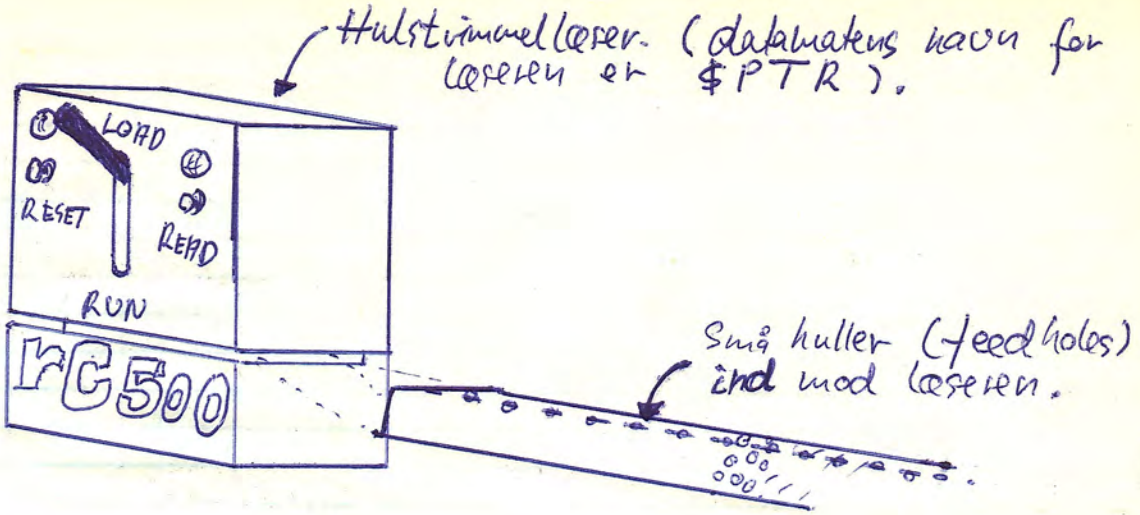
RESERVED FILE NAMES:
$LPT
$CDR
$PTP
$PTR car ret.

13) Hvis der ved de reserverede filnavne tastes car ret., betyder dette, at man ikke har flere navne. Maskinen skifter selv linie, hvis man indtaster de 4 filnavne korrekt. Dette kan virke ret overraskende...
14) Efter det sidste filnavn (her: $PTR) tastes der på tastaturet car ret., og ikke før under filnavnene! Rækkefølgen af navnene er ligegyldig.
15) Maskinen står i STAND BY position.
16) Kontakterne RESET og START aktiveres i denne rækkefølge
17) Nøgle på LOCK
18) Maskinen er klar!

Der gøres opmærksom på, at:

GØRES DER FEJL UNDER OPSTART FRA BLOKERET TILSTAND, SKAL HELE PROCEDUREN GENTAGES FORFRA!!!

(fig. 1)



	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
Carry	~~15~~	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
DATA		●	●	○	●	●	○	○	○	○	●	○	○	○	●	○

Load					Reset	Start		Prg. Load
	●	○	●	●	●	●		●
read	AC0	AC1	AC2	AC3	Stop	Cont		

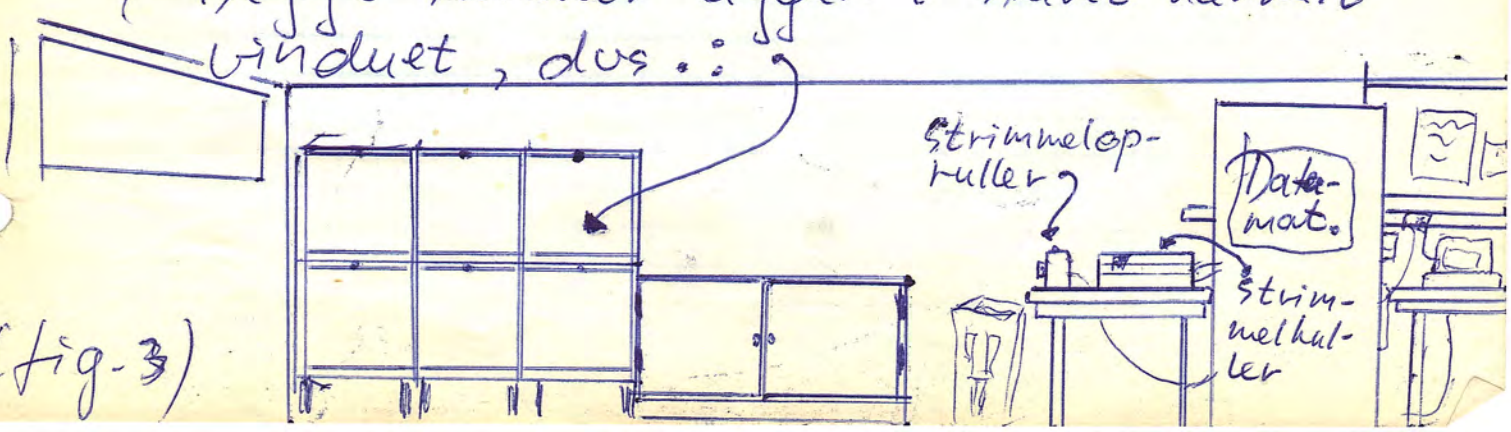
(fig. 2) Frontpanelet af RC7002 (skematisk)

Hulstrimler til opstart:

- Biner loader (binary loader)
  Lille, sort hulstrimmel.

- Comal II
  Stor, hvid eller grøn hulstrimmel.

Begge strimler ligger i skabet nærmest vinduet, dvs.:



(fig. 3)

OPSTART AF BATCH-BASIC

1) Nøgle drejes på ON
2) Kontakterne 12 og 14 op, resten ned
Herefter: Opstart med kort.
3.1) Sæt omskifter mrk. PUNCH - OPT.MARK på kortlæser på PUNCH.
4.1) Start kortlæser, og læg BATCH-BASIC kortene i læseren, excl. det forreste blå kort.
5.1) Tast RESET på kortlæser. Den grønne lampe mrkt. READY skal nu lyse.
6.1) CARD LOADER (minder om BINÆR LOADER) lægges i strimmellæser.
7.1) RESET på strimmellæser.
8.1) Kontakterne RESET og PROGRAM LOAD på datamaskine aktiveres i rækkefgl.
9.1) Herefter indlæses først strimmel, og umiddelbart derefter indlæses kortene. Sker dette ikke, ligger kortene antageligt forkert, og må rystes sammen.
Efter indlæsningen udskriver systemet (=datamaten) følgende ting, som man skal svare på. Der bliver spurgt om PRINT ON LPT (Y OR N), INCLUDING MATSTATEMENTS (Y OR N) - her vil jeg lade den enkelte bruger selv bestemme. Svaret hér skal indtastes uden tasten carret.
Derefter spørger systemet videre, og også hér vil jeg nøjes med at angive spørgsmålene, samt nogle mulige svar, således at brugeren selv kan bestemme opbygningen af BATCH-BASIC. Spørgsmålene er
T= (angiver den tid, et kortprogram må tage)(svar eks. 30)
C= (angiver søjleafstand ved brug af (,) i PRINT)(eks. 14)
H= (angiver antal tegn på en linie ved PRINT)(eks. 82)
V= (antal linier uden formularskrift)(eks. 10000)
Alle svar indtastes efterfulgt af tasten carret. Enhederne på de to første spørgsmål er hhv. sekunder og positioner.
Endelig tastes tasten ESC, og systemet svarer *READY
10.1) Kontakterne 12 og 14 ned.
11.1) Nøgle drejes på LOCK.
12.1) Omskifter mrkt. PUNCH - OPT.MARK sættes på OPT.MARK.

Herefter følger: Opstart med strimmel.
3.2) BINÆR LOADER lægges i strimmellæser.
4.2) RESET på strimmellæser.
5.2) Kontakterne RESET og PROGRAM LOAD aktiveres i denne rækkefølge.
6.2) Efter indlæsning: Alle adressekontakter op.
7.2) BATCH-BASIC strimmel lægges i (lille, gul strimmel).
8.2) RESET på strimmellæser.
9.2) RESET og START aktiveres i denne rækkefølge.
Herefter indlæses BATCH-BASIC, og systemet spørger nu på samme måde som før om nogle ting. Se 9.1) for at gå videre (spring(10.1) over).

Herefter er systemet indlæst.

Kommentare til BATCH-BASIC: Dette sprog er det laveste i BASIC-rækken, og man bør derfor ikke på forhånd stille for store krav til det. Når sproget officielt kaldes "skolebasic", og populært kaldes "babybasic", så får man allerede et indtryk af dets formåen. Generelt kan siges:
Variabelnavne består af ét bogstav, evt. efterfulgt af ét tal.
Man kan ikke udelade ordet LET i LET-sætninger.
Der må kun være én LET-sætning pr. linie.
BYE eksisterer ikke som slettefunktion, men giver udskrift af forløbne tid, siden sidste BYE og sidste NEW.
Af relationssløjfer eksisterer kun FOR - NEXT.
AUTO eksisterer ikke.
Underrutiner eksisterer kun i form af GOSUB - RETURN statements.

- osv. Det anbefales, at man prøver sig frem efter den brune manual til systemet RC-BATCH-BASIC. Prøv f.eks. at indtaste ordren TIME...

2C

OPSTART AF RC BASIC.

Følg Opstart fra blokeret tilstand, pkt. 1-9. Pkt. 7 erstattes med
7) RC BASIC SINGEL USER (lyserød strimmel) ilægges.

Når systemet er indlæst, kommer følgende udskrift på hovedkonsollen:

```
>COPS

DATE (YY.MM.DD) =
```

Man svarer ved at indtaste dags dato på formen år-måned-dag. Der afsluttes med RETURN-tasten.

Systemet skriver nu

```
TIME (HH.MM.SS) =
```

Det aktuelle klokkeslet indtastes på formen time-minut-sekund.

Da systemet ikke kontrollerer det indtastede, er det vigtigt, at det anviste format følges.

Eks.:

Datoen den 4. marts 1977 indtastes således: 77.03.04. (kan indtastes 1977.03.04

Klokkeslettet "5 minutter over 11" indtastes således: 11.05.00

Hvis man ikke ønsker at indtaste dato og klokkeslet, kan man besvare de to spørgsmål ved at trykke på RETURN-tasten. Systemet vil så indsætte værdierne: 00.00.00.

Når dato og klokkeslet er initialiseret, kommer udskriften:

```
YY.MM.DD., HH.MM.SS: TERMINAL XX IDLE
```

på alle de terminaler, som er tilsluttet. XX angiver terminalens nummer (=32 for hovedkonsol i skærm)

En terminal kan nu aktiveres ved, at man trykker på ESC-tasten. Systemet skriver

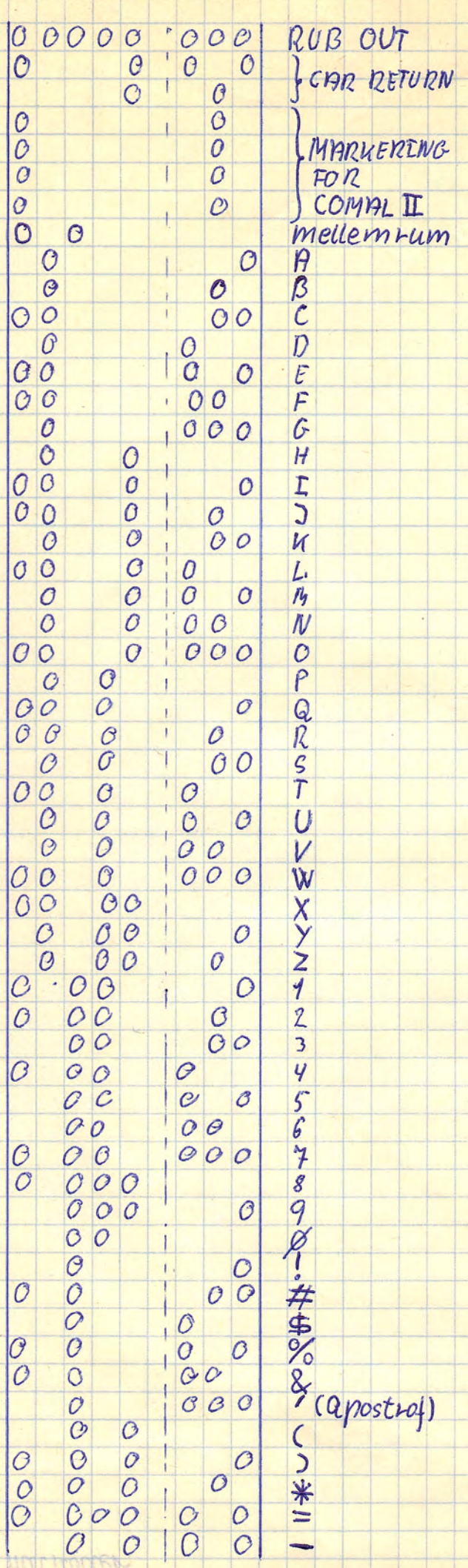
```
ATT
```

og der svares med BASIC (eller COMAL) efterfulgt af RETURN. Herefter kommer udskriften

```
BASIC REV. XX.XX
READY
*
```

og terminalen er klar.

HULKODER



Denne hulkode kan ikke bengttes på hulstrimler, der er fremstillet ved SKÅVE'ning!

Lige parket.

MASKINKODER

RUB OUT	<8>	>	<62>	
A	<65>	.	<46>	
B	<66>	<	<60>	
C	<67>	/	<47>	
D	<68>	?	<63>	
E	<69>	[	<91>	
F	<70>	]	<93>	
G	<71>	↑	<94>	
H	<72>	"	<34>	
I	<73>		<44>	
J	<74>	@	<64>	
K	<75>	←	<95>	
L	<76>	& (mellemrum)	<32>	
M	<77>	Sletter skærm		
N	<78>	eller virker som		
O	<79>	TOP OF FORM	<12>	
P	<80>	TOP OF FORM	<11>	
Q	<81>	Anbringer linker		
R	<82>	midt på skærm	<30> <6>	
S	<83>	Anbringer linker		
T	<84>	i top på skærm	<29>	
U	<85>	Linieskriveren		
V	<86>	skriver med		
W	<87>	stor skrift	<14>	
X	<88>	Aktiverer		
Y	<89>	lydgiver	<7>	
Z	<90>	Lader linker		
1	<49>	falde til næste		
2	<50>	linie uden at gå		
3	<51>	tilbage	<10>	
4	<52>	Aflyder linie-		
5	<53>	skriverens printer-		
6	<54>	kanal	<19>	
7	<55>	Lader linker gå		
8	<56>	1 linie tilbage	<26>	
9	<57>			
Ø	<48>	<0> Fjerner efterfølgende del af sætningen		
!	<33>			
#	<35>			
$	<36>	CTRL N - sætninger printes med start på LPT		
%	<37>			
&	<38>			
' (apostrof)	<39>			
(	<40>			
)	<41>			
*	<42>			
=	<61>			
-	<45>			
.	<58>			
+	<43>			
,	<59>			

Programmering

Programmeringsproceduren på RC 7ØØ2 med COMAL II er ganske enkel. Når man ønsker at indtaste et program skal følgende ting påses (hvis man lige får overtaget datamaten):

1) At datamaten er tændt (ellers, se Opstart 2)
2) At lampen på lyspanelet helt til venstre på datamaten blinker (ellers, a) spørg evt. tilstedeværende om et andet sprog er i brug, eller b) se Opstart 1)
3) At skærmen er tændt
4) Tasten [ESC] aktiveres. Der skal nu komme en stjerne (*) på skærmen i venstre side. Hvis ikke, se fejlskema efter dette.
5) Tast enten NEW – programmer i maskinen ønskes slettet
   RUN – programmer i maskinen ønskes kørt
   LIST – programmer i maskinen ønskes set
6) Hvis NEW er tastet, indtast så programmet, linie for linie.

Fejlskema

1) Hvis tryk på tasten [ESC] ikke resulterer i en stjerne (*) yderst til venstre på skærmen, fortsæt da med følgende:
2) Tast @ (tegnet over P'et). Maskinen skal nu svare med udskriften:
   ERROR Ø1 – KAN IKKE GENKENDE ORD efterfulgt af en lyd
3) Tast ordet ABCDEFGHIJKLMN ind. Maskinen skal da svare:
   ERROR ØØ – FORMATFEJL efterfulgt af en lyd
4) Tast den fejlagtige ordre RUM ind. Maskinen skal da svare:
   ERROR Ø2 – SAETNINGEN ER IKKE RIGTIGT OPBYGGET efterfulgt af en lyd
5) Tast nu ordren RUN ind. Er der et program i maskinen, skal dette gå i gang. Hvis ikke, skal udskriften
   INTET PROGRAM komme, efterfulgt af en lyd
6) Hvis alle disse punkter er fulgt, og der kun kommer tallene ud (ERROR Ø2 i stedet for ERROR Ø2 – SAETNINGEN ER IKKE RIGTIGT OPBYGGET) efterfulgt af en lyd, prøv da tasten [ESC] igen.
   Kommer stjernen da ikke, eller forårsager dette en radikal og stabil ændring af lyspanelet, er maskinen blokeret, og må lades op igen. Se Opstart fra blokeret tilstand.
7) Kommer alle fejludskrifterne korrekt ud, er der antageligt tale om et kørende program, der sikkert indeholder en ON ESC. Man kan gøre to ting for at checke dette:
   A: Tast et komma (,) ind. Hvis der er tale om et sådant program, vil maskinen stoppe med en fejludskrift. Hjælper dette ikke, kan man taste AUTO, og undersøge denne funktion. Se under AUTO.
   B: Sluk for kontakten på væggen, og tænd igen. Tast nu på tasten [ESC], og der vil nu fremkomme skriften STROMAFBRYDELSE. Herefter kan man gå tilbage til: 1) At datamaten er ...
   Hvis checkningen er negativ, er maskinen blokeret, og man må lade den op igen. Se Opstart fra blokeret tilstand.

Generelle bemærkninger om programmer: Et program er et sæt detaljerede instruktioner til datamaten. I BASIC består et program af et antal sætninger med 1 BASIC (COMAL II) instruktion i hver. Hver sætning er udstyret med et linienummer. Se dette.

Linienumre

Linienumre kan ikke udelades, hvis man skriver et program i BASIC. I COMAL kan man ganske vist få datamaten til selv at sætte dem (se under AUTO), men de skal være foran hver sætning. Et linienummer er et helt tal, der er større end Ø og mindre end 1ØØØØ.
Med mindre rækkefølgen af udførslen af programmet ændres ved brug af en af følgende sætninger:

GOTO
GOSUB
EXEC
IF - statements
ON - statements
REPEAT - statements
SELECT - statements
WHILE - statements

vil programmet blive udført efter stigende linienummer.

Indskrivningen af sætningerne kan ske ~~xxxxx~~fuldkommen vilkårligt, da datamaten selv ordner sætningerne efter stigende linienummer.

Linienumrene kan ændres ved brug af RENUMBER, der også ændre linienumrene i GOTO - osv., så at programmet stadig er korrekt. Se under RENUMBER.

TAL

Datamaten kan regne med talstørrelser, der tilhører mængden:

$\{x \mid 5.4 \cdot 10^{-79} \leq |x| \leq 7.2 \cdot 10^{75} \vee x = 0\}$

Et decimalkomma skrives sådan: (.) (punktum). Eks.:
14,93 skrives 14.93
-6,789 skrives -6.789

Beregningerne foretages internt med 7 betydende cifre, der angives ved udskriften (externt) dog kun 6.

For "store" eller for "små" tal kan angives ved eksponentiel notation:

```
12ØØØØØØØØØØØ = 1.2E12
Ø.ØØØØØØØØØØ35= 3.5E-11
-13ØØØØØ       = -1.3E6
-Ø.ØØØØØØØØØØØØ3 = -3E-13
```

Her betyder E det samme som: $1Ø^{\uparrow}$

MAN KAN IKKE NØJES MED AT SKRIVE:
E6 for 1ØØØØØØ el. lign., da bare det at skrive E6 vil blive opfattet som en variabel.

Det er valgfrit om man vil skrive

```
1.E+Ø6          el.
1E+Ø6           el.
1EØ6            el.
1E6             el.        1ØØØØØØ

ØØØØØØ.Ø1ØØØ    el.
ØØØØØØ.Ø1       el.
Ø.Ø1            el.
.Ø1             el.        1E-2
```

osv.

REGNETEGN

Af regnetegn findes følgende:

"*"	multiplikationstegn, kan ikke udelades
"+"	additionstegn
"-"	subtraktionstegn
"/"	divisionstegn
"↑"	potensopløftningstegn, venstre side af tegnet: roden højre side af tegnet : eksponenten Roden skal være større end ~~xxxx~~ nul; grunden til dette, er at maskinen potensopløfter ~~x~~med hjælp af logaritmer.
"()"	paranteser

Følgende er ikke tilladt:

(3+5)(4-3)
mangler multiplikationstegn

2/Ø
division med nul

-3↑2
potensopløftning med negativ rod

((3-2)
mangler parantes

Derimod er det tilladt at gøre:

2+-1
2+((((((1))))))
+2+4 o.l.

PRIORITETSORDEN (HIRAKI)

Et udtryk udregnes efter følgende hirakiramme (rækkefølge):

1) FUNKTIONER (både BUILT-IN og selvdeffinerede)
2) PARANTESER (begyndende med inderste parantes)
3) POTENSOPLØFTNING
4) MULTIPLIKATION OG DIVISION
5) ADDITION OG SUBTRAKTION

Ved udregning af et udtryk som $((2^3)-(4x(3^4)))$ kan man "snyde", og skrive:

2↑3-4*3↑4 , hvorved man sparer paranteserne.

Dette udregnes nemlig sådan:

1)	ingen funktioner	2↑3-4*3↑4	=
2)	ingen paranteser	2↑3-4*3↑4	=
3)	POTENSOPLØFTNING:	8-4*81	=
4)	MULTIPLIKATION :	8-324	=
5)	SUBTRAKTION :	-316	
		=======	

Et udtryk som: ((2+3)↑(4-(2+3)*(4-3))) udregnes:

1)	ingen funktioner	((2+3)↑(4-(2+3)*(4-3)))	=
2)	PARANTESER:	(5↑(4-5*1))	=
3)	MULTIPLIKATION:	(5↑(4-5))	=
4)	SUBTRAKTION:	(5↑(-1))	=
5)	POTENSOPLØFTNING:	5↑-1	= 0.2
			=======

Da hele udtrykket var omgivet af paranteser skete der under hævningen af disse et vekselspil mellem regneopperatorene inde i paranteserne og deres hirakistilling under hævningen af paranteserne.

BUILT-IN FUNKTIONER

Af BUILT-IN FUNKTIONER (indbyggede funktioner) findes der i COMAL II følgende:

SIN(X)	sinus til x	x skal være i radianer
COS(X)	cosinus til x	x skal være i radianer
TAN(X)	tangens til x	x skal være i radianer
ATN(X)	arcus tangens t. x	$x \leq \lvert \frac{\pi}{2} \rvert$
LOG(X)	ln x	naturlig logaritme til x. $x > 0$
EXP(X)	e^x	antilog til naturlig log x. $x \leq \lvert 175 \rvert$
SQR(X)	$\sqrt{x}$	kvadratrod til x. $x \geq 0$
ABS(X)	$\lvert x \rvert$	numerisk værdi af x
INT(X)	$[x]$	hele del af x
SGN(X)	fortegn til x.	
RND(X)	tilfældigt tal mellem o og 1	
LEN(X$)	længden af x$. Se under strengvariable	
TAB(X)	tabulering med længden x. Se under PRINT	
SYS(X)	systemcheck	

Af disse funktioner kræver SGN(X), INT(X), RND(X) og SYS(X) en forklaring:

SGN(X) finder fortegnet for et tal eller udtryk indsat på x'ets plads. Her vises værdien af SGN(X), når x er som vist:

SGN(X)=1	når	x>0
SGN(X)=0	når	x=0
SGN(X)=-1	når	x<0

INT(X) giver den hele del af x, dvs. den del af tallet, der uden oprunding er helt. Eks.:

```
INT(7.6)= 7
INT(8)=   8
INT(0)=   0
```
Bemærk: `INT(-7.5)=-8`
Bemærk: `INT(-7.0001)=-8`

Følgende formel kan give afrunding til 2 decimaler. Benyttes ved kroner og ørerbeløb:

```
f(x)=INT(X*100 + 0.5)/100
```
EKS: `f(14.938)=INT(14.938*100+0.5)/100= INT(1494.3)/100= 1494/100= 14.94`

RND(X) giver tilfældige tal, der ligger jævnt fordelt mellem 0 og 1. x har ingen betydning, men skal være et udtryk, en konstant eller en variabel. Eks.:Første gang RND(X) benyttes kan resultatet se sådan ud:

```
RND(2)=   0.624397
```
men næste gang måske sådan:
```
RND(2)=   9.43868E-2
```
Se iøvrigt under RANDOMIZE.

SYS(X) giver oplysninger om systemet, dvs. nærmere oplysninger om datamaten. Oprindeligt skulle SYS(X) give oplysninger, når $x \in \{0,1,2,\ldots 8,9\}$. Men på grund af, at vores datamat ikke er et Time Sharing System (Tidsdelt System), har vi kun ét tal fra denne mængde, der er praktisk anvendeligt. Dette er funktionen SYS(7), der giver det sidste fejlnummer, der er registreret i datamaten. Om SYS(7), se denne.

RELATIONSTEGN

Relationstegn benyttes i reglen til sammenligninger ved følgende sætninger:

IF sætninger
REPEAT - UNTIL statements, hvor udtrykket forekommer i UNTIL sætningen
WHILE - ENDWHILE statements, hvor udtrykket forekommer i WHILE sætningen.

Af relationstegn findes der:

"<=" el. "=<"	:	mindre end eller lig med
">=" el. "=>"	:	større end eller lig med
"<>" el. "><"	:	forskellig fra
"="	:	lig med
"<"	:	mindre end
">"	:	større end

Relationstegn anbringes, og kan kun anbringes sådan:

relationssætning: tal; relationstegn; tal (; evt yderligere forbindelse) el.
relationssætning: strengvariabel; relationstegn; strengvariabel (; evt. yderligere forbindelse)

Et typisk eksempel:
```
IF A<>3 THEN DO
WHILE A$ = "PER" DO
```

En ikke tilladt relation er:
```
IF "PER">=3 THEN GOTO 17Ø
```

Normalt benyttes relationstegn kun i sætninger som disse. Men visse typer findes dog også andre steder:
"=" se under LET
"<" samt ">", der omklamre maskinkoden. Se under PRINT.

I de fleste relationssætninger, sætninger med relationstegn i, er der kun tale om en relation, dvs. ét relationstegn. Man kan dog komme ud for situationer, hvor det ville være nemmere at sætte flere relationer, og dermed flere relationstegn, i samme relationssætning. Hertil benyttes LOGISKE FORBINDELSE.

LOGISKE FORBINDELSER

Der findes 2 logiske forbindelser i COMAL II. Disse er:

AND	(og)	"∧" &
OR	(eller)	"∨"

BASIC	dansk	matematisk

Opstillingen ser sådan ud (skråstregen mellem AND og OR betyder valgfrit):

relationssætning: relation; AND/OR relation(; evt flere relationer, adskildt af AND/OR)

Typisk eksempel er: IF A+3=5 AND A$<>"HANS" OR 1/17 + B>=Q THEN DO

Ikke tilladte relationssætninger:

```
IF (2<3 OR 4<5) AND 3<4 THEN DO
IF 2<3 (OR) 4<5 (AND) 3<4 THEN DO
```

En relationssætning, hvori der kun forekommer AND, er sand, når og kun når alle relationer er sande.

En relationssætning, der kun indeholder OR, er falsk, når og kun når alle relationer er falske.

Boolske variable (variable, der kun har værdien sand (=1) eller falsk (=Ø)) findes ikke i COMAL II (undtagelser, se under ERR og ESC), derfor er det følgende forkert:

```
K=(2<3 OR 4=4)
K=2<3 OR 4=4
```

Men dette kan klares sådan (det forudsættes, at K har værdien Ø fra tidligere):

```
IF 2<3 OR 4=4 THEN LET K=1
```

ERR og ESC

I COMAL II findes, som de eneste, disse to faste Boolske variable. De har normalt værdien Ø, men kan ved ydre eller indre påvirkning få værdien 1. Denne nye værdi beholdes dog kun til en eventuel undersøgelses af variablene er slut, derefter skifter de om til Ø igen.

Man kan kun benytte disse variable i én form for relationssætning, en sætning, der ovenikøbet kun er en relationssætning, når en af de Boolske variable hér benyttes. Det drejer sig om sætningerne:

ON - THEN GOTO el. ON - THEN GOSUB

Man får ved den første variabel:

1Ø ON ERR THEN GOTO 15Ø el. 1Ø ON ERR THEN GOSUB 15Ø

Denne sætning udfører følgende: Sætningen undersøges konstant ved skift fra én programlinie til en anden, FORUDSAT at ON ERR sætningen står som den første linie i programmet. Står linien i midten af programmet, checkes der kun for hver linie efter sætningen.

Den Boolske variabel her er i dette tilfælde ERR (dansk: Fejl). Hvis der under udførelsen af programmet opstår en fejl, vil variablen få tildelt værdien 1, og maskinen vil da gå til linienumret efter GOTO el. GOSUB. Dette er til tider meget praktisk; hvis man for eksempel ved, at på et tidspunkt under beregningerne vil der opstå en fejl. Normalt vil maskinen da stoppe, når fejlen "mærkes", men ikke hvis man har en ON ERR sætning i programmet - ikke nødvendigvis, da. I så fald udfører maskinen et "hop" til linienumret efter GOTO el. GOSUB.

Eks.: Under en beregningsekvens ved man, at der vil opstå en fejl i programmet. Derfor vælger man at benytte ON ERR for at undgå en stopning af kørslen:

```
1Ø ON ERR THEN GOSUB 6Ø
2Ø FOR I=-1Ø TO 1Ø
3Ø PRINT I, 1/I
4Ø NEXT I
5Ø END
6Ø PRINT "1/Ø"
7Ø RETURN
```

Når datamaten under kørslen når til ordren: Divider 1 med Ø, hvilket som bekendt er ulovligt, bliver den Boolske variabel ERR=1, og datamaten "hopper" til linie 6Ø, hvor den printer teksten "1/Ø". Derefter går den tilbage til linie 4Ø NEXT I, og fortsætter derved til ordren END nås.

Det er en dårlig idé at konstruere programmer, hvor datamaten ikke kan komme ud fra kørslen. Et eksempel på dette er:

```
1Ø ON ERR THEN GOSUB 4Ø
2Ø LET N= 1/Ø
3Ø GOTO 2Ø
4Ø PRINT "FEJL"
```

I dette program optræder dels ON ERR og dels to potentielle fejl: 1/Ø og et manglende RETURN. Disse to fejl ville få ERR til konstant at være 1, og dermed maskinen til konstant at gå til linie 4Ø. Maskinen må herefter nødstoppes med tasten [ESC].

En sætning, der næsten har den samme funktion som ON ERR er sætningen:

ON ESC THEN GOTO el. ON ESC THEN GOSUB

Den fysiske opbygning er ens ved de to sætninger, men i modsætning til ON ERR, der kun reagerer på indre påvirkninger - opståede fejl under kørslen -

reagerer ON ESC kun over for ydre påvirkninger, nemlig tastning af tasten [ESC].

Hvis man altså udefra ønsker at ændre på en sekvensfølge, for eksempel gå til en anden del af programmet el.lign. kan man indbygge en ON ESC i programmet som første linie. Men man skal passe gevaldigt på med ON ESC!! Tasten [ESC] benyttes nemlig også til at nødstoppe datamaten med, og er en ON ESC indbygget, er det ikke altid lige nemt at stoppe maskinen. Nogen gange kan det gå såvidt, at ligegyldigt hvad man gør, så kan man ikke stoppe maskinen fra tastaturet. Er der så yderligere bygget en ON ERR på programmet, kan, når uheldet er ude, problemet blive værre og værre... Se bare på det følgende programeksempel (bør ikke benyttes!):

```
1Ø ON ESC THEN GOTO 3Ø
2Ø ON ERR THEN GOTO 4Ø
3Ø LET N=-1Ø
4Ø LET N=N+1
5Ø LET I=1/N
6Ø GOTO 4Ø
```

Hvad sker der her? Jo, maskinen begynder i den lille GOTO-sløjfe med at gå fra -1Ø til Ø, men idet maskinen når Ø, og divisionen indtræder, opstår der en fejl - og variablen ERR=1. Derved går maskinen til 4Ø, der forøger N med 1 til 1. På nuværende tidspunkt vil maskinen fortsætte normalt, på grund af GOTO sætningen i 6Ø. Men tastes der nu på tasten [ESC] vil variablen ESC=1, den øverste sætning (sætning 1Ø) træder ik kraft, og sender maskinen tilbage til 3Ø, der sætter N til -1Ø. Maskinen begynder i den lille GOTO-sløjfe osv.. Maskinen kan med andre ord ikke standses via tastaturet!

Skulle man ved et uheld rende ind i et lignende tilfælde, kan dette i de fleste tilfælde afhjælpes ved at slukke for datamaten på væggen ved kontakten, og derefter starte igen. I andre tilfælde er det muligt at klare problemet ved at dreje nøglen på OFF, og derefter starte maskinen op igen. Hvis programmet så - mod alle beregninger - fortsætter med at køre(!), så er der god grund til at blive hed om ørene...

Selv benytter jeg ON ESC i tilfælde som disse:

```
1Ø ON ESC THEN GOTO 4Ø
2Ø LET Q=RND(5)
3Ø GOTO 2Ø
4Ø LET E=E+1
5Ø IF E>=2 THEN STOP
6Ø ----------
     --------
       ---
```

hvor jeg ønsker et totalt tilfældigt tal tildelt Q. Efter RUN kører den lille sløjfe, indtil man taster på tasten [ESC]. Efter andet tryk på tasten [ESC] bliver variablen E=2, og dermed er relationssætningen i 5Ø sand, og maskinen stopper.

VARIABLE

1) Almindelige variable.

Almindelige variable består af maksimalt 8 tegn fra det engelske alfabet, hvor der dog må sættes tal efter de eventuelt 8 tegn. Her vises eksempler på almindelige variable:

KASSE1 , KASSENI9 , A , A1 , AØ , ABCDEFGH3 osv.
forskellige

Det er derimod ikke tilladt at skrive:

1HANS , 1234 , K!E , C-3 el.lign.

Skriver man K(E) betyder dette en indiceret variabel med navn K og nummer E. Se disse.
Almindelige variable skal ikke erklæres med DIM sætninger.
Man må ikke benyttes COMAL II ord som variable, heller ikke som indicerede.
Maskinen kender som før nævnt forskel på variabelnavne som:

A , AØ , A1 , AO
KA , KAØ , KA1 , KAO osv.

2) Indicerede variable.

I COMAL II skeldnes der mellem 2 former for indicerede variable (forkortet: i.v.), nemlig enkelt - og dobbelt i.v. . En i.v. består af et variabelnavn af samme slags som under almindelige variable, efterfulgt af en parantes, hvor der ved enkelt i.v. er ét nummer (tal) i; ved dobbelt i.v. er der 2 nummer (tal) i, der er adskildt af et komma.
I stedet for tal som nummer, er det muligt at benytte variable eller regneudtryk o.lign. .
Indicerede variable bør altid erklæres. Se under DIM.

Her vises elementerne i en enkelt i.v., der er erklæret til 4:
KASSE(Ø) , KASSE(1) , KASSE(2) , KASSE(3) , KASSE(4) - altså i virkeligheden 5 elementer.

Her vises elementerne i en dobbelt i.v., der er erklæret til (2,3):
A(Ø,Ø) A(Ø,1) A(Ø,2) A(Ø,3)
A(1,Ø) A(1,1) A(1,2) A(1,3)
A(2,Ø) A(2,1) A(2,2) A(2,3)

Som nævnt kan man benytte regneudtryk på nummerets plads, man skal så blot huske på, at parantestallet skal være helt, så maskinen tager automatisk den hele del af regneudtrykket, og sætter det på nummerets plads - uden oprunding.

Hvis man bruger enkelt i.v., og man på et tidspunkt ønsker alle sine sine i.v. nulstillet, er der ikke anden mulighed end:

```
1Ø DIM A(1Ø)
2Ø FOR X=Ø TO 1Ø
3Ø   A(X)=Ø
4Ø NEXT X
```

eller lignende sløjfer.

Med dobbelte i.v. kan man også benytte sløjfer:

```
1Ø DIM A(1Ø,1Ø)
2Ø FOR X=Ø TO 1Ø
3Ø   FOR Y=Ø TO 1Ø
4Ø     A(X,Y)=Ø
5Ø   NEXT Y
6Ø NEXT X
```

Men dette fylder dels megen plads op i programmet, og dels tager det forholdsvis megen datamattid at udføre. I sådanne tilfælde er det absolut bedre at benytte en enkelt MAT sætning (se iøvrigt under MAT statements).
Eks.:

```
1Ø DIM A(1Ø,1Ø)
2Ø MAT A=ZER
```

eller kortere

```
1Ø MAT A=ZER(1Ø,1Ø)
```

, der erklærer og nulstiller den dobbelt indicerede variable A - hvor vi før brugte 6! MAT sætninger kan desværre ikke benyttes over for enkelt indicerede variable.

Maskinen kender ikke forskel på:

AØ , A(Ø) eller A(Ø,Ø) men derimod på

A , A1 , A(1,1),A(1)

==========

3) Strengvariable.

Strengvariable er variable, der ikke nødvendigvis indeholder tal, som i givent fald ikke kan behandles som tal. En strengvariabel kaldes også for en tekstvariabel eller en alfanumerisk variabel. En strengvariabel er opbygget af et normalt variabelnavn (dog uden "påhængte" tal, som f.x. A3) efterfulgt af et $ (dollartegn). Strengvariable skal erklæres! Se DIM.
Eksempler på gyldige strengvariable:

A$, HANSENS$, FUNKTION$, SVAR$

En nulstillet strengvariabel har intet indhold, ikke engang et mellemrum, da selv mellemrum jo fylder op. I det følgende angives mellemrum med Ø, når det menes nødvendigt.

De eneste tegn, der ikke må komme med i en tekstvariabel er anførselstegn ("), komma (,) og skråstreg (\)(det sidste tegn findes over L'et). Maskinkoderne må meget gerne komme i strengvariable.

Man kan ved brug af en opstilling af strengvariablene, der minder om indicerede variable, få udskrevet dele af variablene. Her følger en oversigt:

skrivemåden:	udskriver:
A$	Hele variablen
A$(Ø)	Hele variablen
A$(k)	Fra og med k'te tegn til ud
A$(a,b)	Fra og med a'te tegn til og med b'te
A$(a,a)	Kun tegn nummer a

a skal være mindre end b. a og b skal være større end Ø. k må ikke være mindre end Ø.

Tekstvariable og talvariable må ikke blandes ukritisk!
Eks.:

```
LET A=A$+B
```

FORKERT!

Man må derimod gerne skrive:

```
LET A=LEN(A$)+B
```

, der bestemmer summen af den aktuelle længde af A$ og B. Længden af en strengvariabel er altid i tal.

LEN(strengvariabel) giver altså det aktuelle antal tegn i en strengvariabel.
Eks.:

```
1Ø DIM P$(2ØØ)
2Ø LET P$="1234ØABCD"; A=LEN(P$)
3Ø PRINT P$, A
```

giver ved kørsel udskriften:

Selv om P$ blev erklæret (dimensioneret) til at kunne indeholde ialt 2ØØ tegn, er der kun 9 aktuelle.

Man må ikke skrive: LET A$=B !

Man kan "addere" to eller flere strengvariable. Eks.:

```
1Ø DIM A$(5Ø),BB$(5Ø), C$(5Ø)
2Ø LET A$="PER HANSEN"; BB$="ØER"
3Ø LET C$=A$,BB$,"ØSKUDT","Ø!)"
4Ø PRINT C$                                    giver udskriften:
PER HANSEN ER SKUDT!)
```

Se iøvrigt under LET.

Man kan benytte alle relationstegnene mellem dele af eller hele strengvariable. Man kan på denne måde ordne strengvariable, for eksempel alfabetisk. Dette gøres meget nemt, da hvert eneste tegn i maskinen har en talbetydning, således at maskinen bare skal sammenligne summen af hvert ords talinformation. Vi har for eksempel, at A "er mindre end" B, og 1 "er mindre end" A. For eksempel kan man ordne to strengvariable alfabetisk således:

```
1ØØ IF A$>B$ THEN DO
11Ø   PRINT A$, B$
12Ø ELSE
13Ø   IF A$<B$ THEN DO
14Ø     PRINT B$, A$
15Ø   ELSE
16Ø     PRINT "A$ OG B$ REPRÆSENTERER DET SAMME ORD."
17Ø   ENDIF
18Ø ENDIF
```

Som en afsluttende bemærkning skal det siges, at benytter man sig af relationstegn, må man være opmærksom på at følgende tre ord ikke har samme betydning (eksemplet er generelt):

PETER
PETERØ
P E T E R

FILER

Inden for datamatikken defineres en fil ved "en navngiven lokalitet, for eksempel et lager eller lignende, hvor man kan sende eller hente data fra". På vores datamat er der tale om 1 lagerplads, hvor man kunne tænke sig at have filprogrammer (filer) stående; det er ferritlageret i maskinen. Uheldigvis kan der kun være ét program i maskinen ad gangen, og dermed ingen filprogrammer. Derimod har vi ved maskinen mulighed for at sende eller hente data fra følgende maskiner:

dansk navn	engelsk navn	filnavn	egenskab
Linieskriver	LinePrinTer	$LPT	udlæsning
Strimmelhuller	Paper Tape Puncher	$PTP	udlæsning
Strimmellæser	Paper Tape Reader	$PTR	indlæsning
(Hulkortlæser)	(CarD Reader)	($CDR)	(indlæsning)

Hulkortlæseren arbejder, idet dette skrives, ikke tilfredsstillende i COMAL II, men filnavnet eksisterer.
For at kunne arbejde med en af disse filer, må man åbne adgangen til dem med en bestemt ordre, nemlig ordren OPEN FILE (se denne) - dette forudsat, at filen er lukket (se CLOSE og CLOSEFILE).

PRINT

Ved en PRINT sætning forstår man en sætning, der skriver en tekst, tal eller på anden måde foretager en udskrift. For at en PRINT sætning skal kunne skrive en tekst ud, må teksten enten stå i en strengvariabel eller stå mellem anførselstegn ("tekst"). Eks.:

```
1Ø PRINT 3.4,5.6
2Ø PRINT "VAR TO TAL"                    giver udskriften
3.4              5.6
VAR TO TAL
```

I linie 1Ø er et (,) (komma) indført mellem de to tal. Dette bevirker, at den næste tekst, eller det næste tal i dette tilfælde, efter kommaet flytter hen i en ny kolonne. Fra begyndelsestegn i en kolonne til begyndelsestegn i næstfølgende er der 15 pladser til tegn o.lign. Man kan højest få 5 kolonner på skærmen/linieskriveren, da der ialt er 82 pladser på skærmen, og ca. 8o pladser på linieskriveren.
Ved at skrive: 1Ø PRINT ,,,,5.6
flyttes tallet 5.6 hen i 4. kolonne, grundet de 4 kommaer.

Et (;) (semikolon) mellem 2 tal, bogstaver el. lign., bevirker, at skriften efter tegnet flytter netop én plads længere frem på linien.

Man kan valgfrit sætte sin tekst et hvilket som helst sted på papiret ved brug af funktionen TAB(X). Eks.:

```
1Ø PRINT TAB(35)"OVERSKRIFT"                    giver udskriften

                                    OVERSKRIFT
```

Dvs. argumentet til TAB(X) flyttede blinkeren/skrivestiften ialt 35 pladser frem, og derefter skrive teksten OVERSKRIFT begyndende ved plads nummer 36. Man kan få en særlig effekt ved dette lille program:

```
1Ø LET N=N+1; A=SIN(N)
2Ø PRINT TAB(N);A
3Ø GOTO 1Ø
```

, der ved RUN giver udskriften:

```
o.841471
 o.909297
  o.14112
   -o.756802
    -o.958924          osv.
```

Her ser man, at hver gang variablen N forøges, flytter udskriften 1 plads længere ud. Man kan benytte vil give en udskrift af en sinuskurve:

```
1Ø LET N=N+o.25
2Ø PRINT TAB(4Ø*SIN(N)+4Ø);"*"
3Ø GOTO 1Ø
```

Nøjagtigheden af denne kurve kan ændres ved at regulere på variablen N, dvs. ændre på forøgelsesværdien (her: Ø.25). Senere under FOR - NEXT vil jeg vise samme program, bare lidt forbedret.
Teorien hér er følgende: Variablen N forøges med en værdi, udtrykket for TAB beregnes, PRINT TAB(udtryk) flytter blinker/skrivestift det antal pladser frem, som den hele del af resultatet angiver, og hér sættes der en *. Dernæst går maskinen til 1Ø, osv.

Man kan ikke benytte negative argumenter i en TAB sætning, men det gør derimod ikke så meget, hvis argumentet er større end 8o, da blinker/skrivestift alligevel flytter det angivne antal pladser - markøren fortsætter bare i den følgende linie.

Man kan ved enkelte beregninger undlade LET sætningen, og istedet benytte

PRINT sætningen. Eks.:

```
1Ø PRINT"PRQVE:",3+SQR(16)
```

der giver udskriften

```
PRQVE:          7
```

Idet udtrykket 3+SQR(16) først regnes ud, og derefter skrives tekst og resultat.

Ønsker man at skrive sine resultater op på en pæn måde, kan man benytte PRINT USING. Eks.:

```
1Ø PRINT USING "NUMMERET ER: ## OG ##.#",12.5, 1.33
```

giver udskriften:

```
NUMMERET ER: 13 OG  1.3
```

Det ses, at der foretages automatisk oprunding efter normale regler, når en del af tallet ikke kan komme med. Nummertegnene (#) benyttes til den varierende del af teksten. Formatet for en PRINT USING ser sådan ud:

```
linienummer PRINT USING "tekst ####.#(evt. tekst osv.)",variabel/udtryk/
                                                  konstant(, konstant osv.) (,/;)
```

Til slut i en PRINT USING sætning er det altså tilladt at benytte enten ,(komma) eller ; (semikolon) efter de føromtalte regler. Som variabel og/eller konstant kan også benyttes tekstvariabel.
Udskriftsprogrammet fra før kan vi forbedre med PRINT USING :

```
1Ø LET N=N+Ø.25
2Ø PRINT USING "-#.##",SIN(N);
3Ø PRINT TAB(4Ø*SIN(N)+4Ø);"*"
4Ø GOTO 1Ø
```

Minusset (-) foran nummertegnene betyder, at et tal, der er negativt, får sit fortegn med sig. Eks.:

```
"-##",N      og N=-13  giver udskriften   -13
"-##",N      og N=12   giver udskriften    12
"##",N       og N=-13  giver udskriften   13
```

Jeg vil ikke skitsere udskriften fra foregående program, men man kan selv prøve med en RUN.

Ved at skrive PRINT USING "-#.##↑↑↑↑",N opnår man at få skrevet N ud med fortegn (hvis negativt) og i eksponentiel notation. Antallet af nummertegn er valgfrit, ligesom kommaets (.) placering, men der skal være 4 pile for en sådan udskrift virker efter hensigten.

Bliver tekst eller tal større end det antal nummertegn, der er afsat i USING delen, skriver maskinen stjerner (*) i hele det felt, hvor nummertegnene er placeret. Eks.: PRINT USING "### ",12345 giver udskriften

```
****
```

Endnu en ting, der kan styre maskinens udskrift totalt(!) er maskinkoderne. Disse særlige ordrer er især velegnede til skærmen. Her ses nogle eksempler:

```
1Ø PRINT "DETTE ER<1Ø> EN PRQVE"
```

giver

```
DETTE ER
        EN PRQVE
```

```
1Ø PRINT"DETTE ER<26><26> EN PRØVE"
```

giver

```
EN PRØVE
DETTE ER
```

```
1Ø PRINT"<34>PRØVE<34><63>"
```

giver

"PRØVE"?

En liste (dog ufuldstændig) over de forskellige maskinkoder er angivet

andetsteds. Iøvrigt tilrådes det, at man prøver sig frem med disse, og kombinationer mellem dem. Som hjælp bemærkes det, at:

1Ø PRINT"⟨12⟩⟨3Ø⟩" sletter skærm og sætter blinker midt på skærmen
2Ø PRINT"⟨3Ø⟩⟨12⟩⟨3Ø⟩" virker som PRINT"⟨29⟩"

Maskinkoder kan indsættes i tekstdelen i INPUT sætninger, anbringes i strengvariable og inputtes i strengvariable. Datamaten vil søge at oversætte så mange som muligt inden kørslen, f.eks ⟨65⟩ oversættes til A. Ved brug af dem i udskriftssætninger (PRINT sætninger), skal de opfattes som tekst, dvs. omgives af anførselstegn (")!

PRINT, se endvidere PRINT FILE samt Forenklinger.

N.B.: Skærmens udskriftshøjde (y-retningen) er på ialt 23 linier.

LET

Ved en LET sætning forstår man en sætning, der tildeler en variabel (ikke en Boolsk, se ERR og ESC) en værdi. Eks.: variablen A skal have værdien 2, B skal have værdien 5 og C værdien -3:

```
1Ø LET A=2
2Ø LET B=5
3Ø LET C= -3
```

Dette kan i COMAL II skæres ned til én linie, der udfører det samme. Eks.:

```
1Ø LET A=2; B=5; C=-3
```

Se iøvrigt under Forenklinger.
En variabel kan få et udtryks værdi ved hjælp af LET sætningen:

```
1Ø LET A=-1Ø+5; B=SIN(A)-A
```

Her udregnes først værdien A til -5, dernæst værdien B=SIN(-5)+5

Man kan ikke lade flere variable få den samme værdi ved en opbygning som:

```
LET A=B=C=D=1Ø
```

Men må skrive:

```
LET A=1Ø; B=1Ø; C=1Ø; D=1Ø
```

Det er heller ikke tilladt at skrive:

```
LET 2+A=3 el. lign.,
```

men man må derimod gerne skrive:

```
LET A=A+1 o.a.
```

Denne sætning adderer 1 til A's indhold. F.eks., hvis A før udførelsen af sætningen var 1Ø, bliver A nu: A=A+1 = 1Ø+1 = 11. Denne slags sætninger er meget nyttige i tabelopgaver, hvor man benytter REPEAT statements.

LET sætninger benyttes også til at tildele strengvariable værdier, og også til en vis grad udtryk. Denne tildeling kan ske samtidigmed alm. variable:

```
LET A$="1234"; A=3+B                    osv.
```

Bemærk, at værdien til strengvariablen skal stå omgivet af anførselstegn ("tekst"). De eneste udtryk, som en strengvariabel kan deltage i, er fgl.:

```
LET A$=B$,A$          , der faktisk virker som LET A$=A$
LET A$=A$,B$          , der "adderer" B$'s indhold til A$. Det er naturligvis tilladt at skrive:
LET A$=A$,"1234"      , der "adderer" ordet 1234 til A$
```

Formatet af en LET sætning ser sådan ud:

```
linienummer LET variabel= udtryk/variabel/konstant (; variabel2 = ...osv.)
```

Om der skal være tale om udtryk, variabel eller konstant er selvfølgeligt valgfrit.

Iøvrigt noteres, at:
I en sammensat LET sætning foregår udregninger, osv., fra venstre mod højre, altså sådan:

```
LET A=5; B=A*-1Ø; C=SIN(A+B↑2)
```

NEW

Ordren NEW benyttes, når man ønsker at meddele datamaten at et program skal slettes. NEW bør indtastes før man føder et nyt program ind i maskinen, da denne ellers vil tage de nye linier som rettelser til det allerede bestående program. NEW benyttes også hvis man arbejder uden for maskinens programdel, dvs. hvis man benytter BASIC (COMAL II) sætninger uden linienumre; hér til at nulstille maskinens variabellager.
NEW kan indbygges i programmer. Eks.:

```
IF A=1 THEN NEW
```

Her vil programmet da "NEW'es", når A får værdien 1 .

BYE

Ordren BYE minder meget om NEW; den sletter også programmer og nulstiller variable, men i modsætning til NEW, så "låser" ordren BYE også tastaturet med undtagelse af tasten ESC . Dvs., ved BYE må man trykke på tasten ESC for at "låse" tastaturet op igen.
BYE kan indbygges i programmer på samme måde som NEW. Eks.:

```
10 ON ERR THEN GOTO 30
20 GOTO 40
30 BYE
```

Her vil programmet "BYE'es", når der under udførelsen opstår en fejl.

SIZE

Systemordren SIZE giver oplysninger om, hvor meget af maskinens indre lager, der er i brug (benævnt ved USED:), og tilsvarende, hvor meget der er tilbage (benævnt ved LEFT:). Oplysningerne er tal i BYTES, hvor det gælder at 1 BYTES svarer til ca. 8 BIT, der igen svarer til ca. 8 adresseceller.

RUN

Systemordren RUN giver centralenheden i datamaten besked på at starte udførslen af programmet. Ønsker man at starte midt inde i programmet, kan dette gøres ved at taste RUNlinienr., hvor linienr. er nummeret på den linie, hvor man ønsker at starte programudførelsen fra. RUNlinienr. udmærker sig ved ikke at slette variable, som man eventuelt har indtastet udenfor programmet uden linienummer; dette i modsætning til RUN.

RUNL

Systemordren RUNL udfører det samme som RUN, men alle PRINT sætninger (og kun dem!) bliver skrevet udå på linieskriveren. Undtagelser: MAT PRINT sætninger bliver ikke udskrevet ved RUNL,: Man må her benytte PRINT FILE.
Også RUNLlinienr. findes. Eks.:

```
10 INPUT A
20 PRINT A
RUNL
```

RUNL bevirker her, at spørgsmålstegnet, der angiver INPUT sætningen, vil komme på skærmen, men PRINT sætningen skrives på linieskriveren.

CON eller CONL

Stoppes programmet, enten via en STOP sætning (s.d.) eller ved et tryk på tasten ESC , vil man kunne fortsætte fra det sted, hvor standsningen skete, ved systemordren CON eller CONL, hvor CONL virker som RUNL. Man må ikke skrive CONlinienr., eller CONLlinienr.!

STOP

Programordren STOP benyttes til at bringe datamaten til standsning et givent sted i programmet. STOP lader variable beholde deres værdier. Standsningen ved STOP kan effektueres enten ved en relationssætning, eller ved formatet:

linienummer STOP

Ved en relationssætning kan formatet se sådan ud:

IF (variabel/udtryk/konstant) relation (variabel/udtryk/konstant) THEN STOP

```
Eks.:          IF A$="STOP" THEN STOP
Også           ON ERR THEN STOP                    el.
               ON ESC THEN STOP                    findes.
```

Et brugt eksempel er:

```
1Ø PRINT"DEFFINER FUNKTIONEN EFTER STOP I LINIE 3Ø SOM DEF FNA(X)"
2Ø STOP
3Ø DEF FNA(X)=X+SIN(X/2)
      ======
       ====
        ==
```

END

Programordren END markerer afslutningen af et program. Denne er, til forskel fra STOP, ikke tilladt at anbringe direkte i relationssætninger eller ERR og ESC sætninger. Men man kan gå omveje. Eks.:

```
1Ø ON ERR THEN GOTO 3ØØ
      ======
       ====
3ØØ END
```

Efter ordet END er det tilladt at skrive, efter et mellemrum, en valgfri tekst, for eksempel, hvorfor programmet slutter her. Teksten, der må indeholde alle tastaturets tegn, står kun som oplysning til programmøren, og vil ikke blive trykt. Eks.:

```
3ØØ END  PROGRAMMET SLUTTER, DA DER ER OPSTÅET FEJL. SE SYS(7).
```

Med mindre der findes relationssløjfer, GOTO sætninger o.l., er det ikke nødvendigt at benytte END, da maskinen stopper efter udførelsen af programmets sidste linie.

REM

Programordet REM giver mulighed for at indføre kommentare og bemærkninger (REM: forkortelse for REMARKS) i ens program. Disse sætninger udføres ikke, men står i programmet som forklarende tekst. Da det til tider kan være svært at gennemskue, hvad et bestemt program udfører, er det som regel en hjælp at indføre REM sætninger. Der kan henvises til REM sætninger fra f.eks. GOTO sætninger, men som før nævnt bliver REM sætningerne ikke udført, datameten går bare videre til næste linie. Eks.:

```
1Ø REM DETTE PROGRAM OPSTILLER EN TABEL MED BEREGNINGER AF FØLGENDE FUNK-
2Ø REM TIONER: SIN(X), COS(X), TAN(X) OG EN SELVDEFFINERET FUNKTION; FNA(X).
                ==========
                = = = = =
               =  =  =  =
15Ø GOTO 1Ø
```

RENUMBER

Ønsker man at forandre mellemrummet mellem sine linienumre, altså forandre linienumrene homogent, gøres dette med systemordren RENUMBER. Dette kan gøres sådan:

RENUMBER	der giver første linienummer=1Ø, og springet mellem linierne=1Ø.
RENUMBERtal	skal give første linienummer=tal, og springet=tal.
RENUMBERtal a,tal b	giver første linienummer=tal a, og springet=tal b.

I skrivende stund vides ikke med bestemthed om ordren

RENUMBER STEP tal	er lovlig. I givet fald giver den første linienummer =1Ø og springet=tal.

Det højeste linienummer i maskinen er 9999. Giver en RENUMBER linienumre ud over dette, skifter maskinen over automatisk til

RENUMBER 1,1

Har man under indskrivningen henvist med GOTO - eller med lignende sætninger - til ikke-eksisterende linienumre, og tastes der umidelbart efter sidst indskrevne linie systemordren RENUMBER, får man fejludskrifter under maskinens udførsel af RENUMBER'en. Disse udskrifter viser stedet, hvor en henvisning til ikke-eksisterende linienummer har fundet sted, og det fejlagtige nummer bliver erstattet med linienummeret (ikke-eksisterende):

ØØØØ. Ellers vil RENUMBER'en forløbe normalt.

RANDOMIZE

Ordren RANDOMIZE skulle få maskinen under brug af RND(X) til at starte fuldstændigt tilfældigt i maskinens indbyggede tabel, der indeholder ialt 32.ooo forskellige tilfældige tal. Desværre er denne funktion i COMAL II blevet til en "dummy"funktion, dvs. en funktion, som man kan indtaste, men som ingen virkning har. Jeg har ingen anelse om, hvorfor denne funktion ikke virker i COMAL II.

LIST

Systemordren LIST giver en udskrift af hele programmet i det indre lager (det aktuelle program).

LISTlinienr.1,linienr.2 giver udskrift af programmet i indre lager fra og med linienr.1 til og med linienr.2.

I skrivende stund vides det ikke om ordren LIST TO lininr. findes i COMAL II Denne skulle i givet fald give udskrift af programmet fra og med første linie til og med lininr..

Ved ønsket om udskrift af programmet på linieskriveren skal en af følgende ordre skrives, efterfulgt af skriverens filnavn:

```
LIST"$LPT"
LISTlinienr.1,linienr.2"$LPT"          og måske
LIST TO linienr."$LPT"
```

Ønskes programmet ud på hulstrimmel til eventuel senere brug (se ENTER) erstattes blot filnavnet "$LPT" med filnavnet på hulskriveren "$PTP".

Forsøges at LIST'e når der ikke er noget program i maskinen, svarer denne med: INTET PROGRAM som under RUN uden program.

SAVE

Ordren SAVE giver et direkte binært billede af det aktuelle program og de tilhørende variabelhukommelser osv. . SAVE kan ikke benyttes uden filnavn.

Stoppes en programkørsel med tasten ESC, og skrives der derefter SAVE"$PTP vil en hulstrimmel, der indeholder programmet, alle variabelværdier, regnedelen og en slags "pil", der "peger" på det sted, hvor maskinen blev stoppet. Man kan ved en ordre LOAD indlæse denne hulstrimmel.

Tastes SAVE"$LPT" fås billedet på linieskriveren. Da billedet indeholder maskinkoder, vil linieskriveren opfører sig noget besynderligt: Skrive på en linie flere gange, pludselig reagere som om der kom en maskinkode 12 ind, og måske enda slukke lampen, der markerer at man kan skrive på linieskriveren. Sker dette sidste, tændes der blot for lampen igen, og linieskriveren vil fortsætte. Det er ikke muligt at se ret meget ud af en sådan SAVE'ning, men man kan få et indblik i, om der er noget galt med maskinen; men dette dog kun, hvis man er meget øvet.

ENTER

Ordren ENTER benyttes til indlæsning af normale hulstrimler (eller hulkort, der dog i skrivende stund kan benyttes), det vil sige hulstrimler, der er hullet ved brug af LIST.

Ved indlæsning af hulstrimler bruges ENTER således:

ENTER"$PTR" , hvor PTR er filnavnet for strimmellæser.

Herefter bliver den ilagte hulstrimmel automatisk kørt igennem med en hastighed på ca. 2.5 meter pr. sec., der svarer til ca. looo tegn pr. sec. Vides det ikke, hvordan hulstrimler lægges i, se da under Behandling af programmel og maskiner.

ENTER kan indbygges i programmer. Eks.:

```
1Ø INPUT A
2Ø IF A=1Ø THEN ENTER"$PTR"
3Ø GOTO 1Ø
```

Når man indtaster 1Ø vil et givent program i strimmellæseren blive indlæst, og da der ikke er givet en form for stopordre, vil det nye program omgående begynde at køre.
Under en indlæsning vil linier i et eventuelt ikke slettet program blive erstattet med linier i det indlæste program.

==========

LOAD

Ordren LOAD virker omtrent som ENTER, her kan man dog kun indlæse SAVE'ede programmer. Efter en

LOAD"$PTR"

der ikke kan indbygges i programmer, kan man taste CON, og programmet vil herefter fortsætte med hvad den var i færd med i det øjeblik, da maskinen blev stoppet. Man kan også taste RUN el. lign., og ordrene vil virke som om programmet blev indtastet normalt

AUTO

Systemordren AUTO benyttes, når man ikke selv vil skrive sine linienumre. AUTO benyttes især når der er tale om et program, hvor man har rettet linienumrene ind med regelmæssige spring. Eksempler på regelmæssige spring er jo 1-2-3..., men hvis man ikke er helt sikker på, om en given linie skal ind eller ej, er det en god idé at benytte 1Ø-2Ø-3Ø... . AUTO er indrettet sådan, at indtaster man kun selve ordet AUTO, vil linienumrene netop blive udskrevet i 1Ø-2Ø-3Ø osv. . Her følger en oversigt:

AUTO	linienumrene sådan:	1Ø-2Ø-3Ø...
AUTOtal	" fra:	tal-tal+1Ø- tal+2Ø ...
AUTOtal a,tal b	" fra:	tal a-tal a+tal b- tal a+2 tal b osv...

Såvidt vides må man ikke skrive AUTO STEP tal, der skulle (om muligt) virke som en art RENUMBER STEP tal (se RENUMBER).

Tastes der på tasten [car ret.] når en AUTO er i funktion, vil datamaten fortsætte med linienummeret til næste linie. Ønsker man at komme ud at linienummerskrivningen, se da CLEAR.

AUTOPROC

Systemordren AUTOPROC ligner lidt AUTO. Men i modsætning til AUTO er AUTOPROC fast indstillet på noget, der minder om AUTOtal, 1. Her er (tal) det næstfølgende linienummer i 1Ø-2Ø-3ø rækken. Dette må vises ved en oversigt:

	indtastningstid	virker som
AUTOPROC	som først indtastet efter NEW	AUTO 1Ø,1
AUTOPROC	efter f.x. linie 2Ø	AUTO 3Ø,1
AUTOPROC tal	efter f.x. linie tal	AUTO tal+1Ø,1

AUTOPROC benyttes i de tilfælde, hvor man ønsker at skrive små underrutiner ind i programmer. Eks.:

```
1ØØ GOTO 1Ø
11Ø END
*
AUTOPROC
12Ø PROC HANS
121 REM DETTE ER UNDERRUTINE 1.
```

(dette var en del af et tænkt program.)

CLEAR

Man kan meget nemt komme ud af AUTO eller AUTOPROC ved at aktivere tasten [ESC] . Men hvis man synes, det ikke er morsomt, kan man i stedet skrive systemordren CLEAR, umidelbart efter linienummeret. Dette vil virke som om man benyttede tasten [ESC], dog uden at sætte stjerne. CLEAR har sådan set ikke andre funktioner.

FOR - NEXT

Til tider kan det være meget nyttigt i et program at få en bestemt del i det gentaget et bestemt antal gange, for eksempel INPUT til indicerede variable. I stedet for at skrive en hel række INPUT sætninger efter hinanden, for måske at fylde 4Ø adresser i en indiceret variabel op, kan man nøjes med at skrive 1 inde i en sløjfe, der automatisk vil blive gennemløbet det aktuelle antal gange; her altså 4Ø. I COMAL'II findes der ialt 3 sådanne sløjfetyper; en af disse kaldes for en FOR - NEXT sløjfe (eller et FOR - NEXT statement). Formatet til dette ser sådan ud:

```
linienummer FOR variabela=konstant/tal/variabel TO konstant/tal/variabel
                                  (STEP konstant/tal/variabel    STEP kan
                                   udelades i visse situationer)
          ------- linier i sløjfen ----
linienummer NEXT variabela
```

Eks.: En indiceret variabel A, der erklæres til 2ØØ, skal tildeles data i ialt ANTAL adresser:

```
1Ø DIM A(2ØØ)
2Ø INPUT ANTAL
3Ø FOR ADRESSE = 1 TO ANTAL STEP 1
4Ø   INPUT A(ADRESSE)
5Ø NEXT ADRESSE
```

I linie 2Ø beder maskinen om antallet af gennemløb, der ønskes - her, hvor mange adresser, der skal benyttes. I linie 3Ø begynder maskinen med sløjfen, idet den siger: FRA ADRESSE=1 TIL ANTAL MED ET SPRING PÅ 1. STEP kan undlades, når STEP'et - som her - er lig med 1. I 4Ø spørger maskinen om data til A med ADRESSE = 1; altså til A(1). Og i linie 5Ø er ordren NÆSTE ADRESSE, der betyder det samme som "læg 1 til ADRESSE, og gå tilbage til FOR ADRESSE...". Hermed virker linie 3Ø altså også som relationssætning, for er variablen ADRESSE lig med eller større end fortsætter maskinen med linienummeret efter 5Ø.

Det er muligt at sætte beregninger ind i FOR sætningen:

```
FOR Y=SIN(1.3) TO 2+LOG(3) STEP Ø.3/TAN(3.1)
```

Og man må også lade TO - og STEP værdierne ændre sig under sløjfens gennemkørsel:

```
1Ø LET A=1Ø; B=1
2Ø FOR X=1 TO A STEP B
3Ø   LET A=A+2; B=B+3
4Ø   PRINT X,A,B
5Ø NEXT X
```

Jeg vil ikke skitsere udskriften, men prøv selv. Det eneste, man skal huske, er at sløjfen først forlades, når kontrolvariablen (her X) er større end slutvariablen (her A).

Det er tilladt at sætte sløjferne sammen, hvis der er flere, blot skal man påse, at man ikke krydser sløjferne:

Gyldigt

```
FOR X...
  FOR Y...
  ........
  NEXT Y
NEXT X
```

Ugyldigt

```
FOR X...
  FOR Y...
  ........
  NEXT X
NEXT Y
```

Et gennemløb med flere FOR - NEXT statements:

```
1ØØ FOR I=1 TO 2Ø
11Ø    FOR A1 = A(I) TO A(I)+3Ø
12Ø       LET A(I-1)=SIN(A+I)+A(I-1)
13Ø    NEXT A1
14Ø NEXT I
```

Ialt må 7 sådanne sløjfer sættes inden i hinanden (i niveauer; eksemplet her havde niveauet 2).

Det er naturligvis tilladt at benytte negative værdier i en sløjfe (her i FOR sætningen):
FOR I=1Ø TO -1ØØ STEP -2   , hvor sløjfen forlades, når I er mindre end -1ØØ

Det er muligt at forlade en FOR - NEXT sløjfe ved følgende statements:
GOSUB - RETURN
ON - GOSUB - RETURN
IF - GOSUB - RETURN
EXEC - PROC - ENDPROC
IF - EXEC - PROC - ENDPROC   eller med GOTO, hvis der hurtigt efter GOTO sætningens "landingspunkt" kommer en sætning, der fører udførslen tilbage til det sted, hvorfra den kom.

= =

Under PRINT blev der vist et lille program til udplotning af sinuskurver, og jeg lovede at forbedre dette under FOR - NEXT. Dette være hermed gjort:

```
1Ø DEF FNA(X)=X*355/(113*18o)
2Ø ON ESC THEN 3Ø
3Ø PRINT"HVIS STOP QNSKES, SKAL SPRING=STARTVAERDI=SLUTVAERDI."
4Ø PRINT"PROGRAM TIL UDPLOTNING AF SINUSKURVE."
5Ø PRINT"================================================="
6Ø INPUT"MAKSIMALT SPRING (I GRADER):  ",SPRING
7Ø INPUT"STARTVAERDI (GRADER):  ",START,"  SLUTVAERDI:  ",SLUT
8Ø IF SPRING=START AND START=SLUT THEN GOTO 16Ø
9Ø FOR X=START TO SLUT STEP SPRING
1ØØ    PRINT USING"-#.## ",SIN(FNA(X));
11Ø    PRINT TAB(4Ø*SIN(FNA(X))+4Ø);"*"
12Ø NEXT X
13Ø PRINT"================================================="
14Ø PRINT
15Ø GOTO 6Ø
16Ø END  COPYRIGTH: NORMANN AA. NIELSEN, 1977
```

DIM

Ordren DIM benyttes til at DIMensionere eller erklære indicerede eller strengvariable; dvs. afsætte en nøje defineret mængde plads i maskinens lager til disse variable. Formatet ser sådan ud:

DIM enkelt/dobbelt/strengvariabel a (, enkelt/dobbelt/strengvariabel b, osv)

Eks.: I en variabel A$ må der højest være plads til 1Ø tegn:
1Ø DIM A$(1Ø)

Skriver man DIM A(2ØØ), B(5,7)
er variablen { A dimensioneret til ialt (2ØØ+1) pladser, da første nummer er Ø
B dimensioneret til ialt (5+1)•(7+1) pladser; første nummer er (Ø,Ø).

Der gælder, at dimensionerer man en variabel, er antallet af pladser for en enkelt indiceret = N+1 (dimensioneret til N), og for en dobbelt indiceret = (N+1)•(M+1) (dimensioneret til (N,M))

Benytter man indicerede variable, behøver man ikke at dimensionere dem, hvis samlet antal pladser ikke er større end

{ ved enkelt indiceret 11 pladser
ved dobbelt indiceret 121 pladser

Det er dog altid en fordel at dimensionere sine indicerede variable.

Strengvariable skal altid erklæres!

Man kan få en nul - plads strengvariable ved: DIM A$(Ø), men denne kan vist ikke benyttes til ret meget...

Man må dimensionere med tal, der er større end eller lig med nul. Dette er således forkert:
1Ø DIM A(-3,5), B(-7), A$(-1)

DIM må meget gerne benyttes uden for programmer.

READ

Ordren READ virker som en form for INPUT sætning: Den tildeler nogle variable data, der dog ikke er hentet externt, som ved INPUT, men internt, fra maskinens lager. READ benyttes de steder, hvor man ellers ville sætte INPUT sætninger ind. Eks.:

```
1Ø DIM A$(2Ø)
2Ø READ A,B,C,A$
3Ø PRINT A,B,C,A$
4Ø GOTO 2Ø
```

Formatet er altså således:
(linienummer) READ variabel 1,variabel 2, ...,variabel n

DATA

Programordren DATA adviserer maskinen om, at det følgende, der står efter ordet DATA er - som ordet siger - data til videre behandling i maskinen. DATA sætninger kan anbringes overalt i programmet, og behøver ikke at stå klods op af hinanden. Ved starten af programmet anbringer maskinen alle dataene i en DATA-blok (i rækkefølge v.h.a. linienumrene). Herefter sætter maskinen en imaginær "pil" på den første værdi i blokken. Når den første READ sætning mødes, vil værdien blive læst ind i variablen i READ sætningen, og pilen flyttes én plads længere frem, således at der hele tiden læses en ny værdi ind. Er der ikke flere data til READ, stopper maskinen med en fejludskrift.
Det er muligt at indføre alfanumeriske konstanter (ord) i en DATA sætning. Man skal da huske på, at en sådan konstant altid skal være omsluttet af anførselstegn ("). Eksemplet fra før fortsat med DATA sætninger:

```
5Ø DATA 1,2,3,"PETER",4
6Ø REM DETTE VAR FØRSTE DATA SAETNING
7Ø DATA 5,6,"HANS"
8Ø DATA 7,8,9.2,"S"
9Ø REM DETTE VAR SIDSTE DATA SAETNING.
```

Det er ikke tilladt at skrive:

```
5Ø DATA 1,2,3,PETER
```

Formatet ser således ud:

linienummer DATA værdi 1, værdi 2, ..., værdi (n-1), værdi n

RESTORE

RESTORE benyttes til at sætte "pilen" i DATA-blokken tilbage på blokkens første plads. Herved begynder den følgende READ sætning altså forfra i blokken.
Formatet er:
(linienummer) RESTORE

OPEN FILE

Ordren OPEN FILE benyttes til at åbne en ønsket fil, men kun hvis denne fil allerede er lukket (ellers, se CLOSE og CLOSE FILE). Formatet er således:

(linienummer) OPEN FILE(a,b),"filnavn"
a er et navn på filen, som brugeren bestemmer. $\emptyset < a \leq 7$.
b angiver filens funktion: 1 hvis man vil skrive på filen ($LPT, $PTP)
3 hvis man vil læse på filen ($PTR, $CDR)
Filnavnet er navnet på den ønskede fil (se ovenfor eller under FILER).

Det er tilladt at give samme fil 2 forskellige numre i a (ikke b numre).
Det er ulovligt at give forskellige filer samme ax nummer.

Eks. (alle fire mulige filer åbnes):
```
1Ø OPEN FILE(1,1),"$LPT"
2Ø OPEN FILE(2,3),"$PTR"
3Ø OPEN FILE(3,1),"$PTP"
4Ø OPEN FILE(4,3),"$CDR"
```

CLOSE

Ønsker man at lukke allerede åbnede filer, eller sikre sig, at der ikke er nogen filer, der er åbnede, benytter man ordren CLOSE. Denne lukker alle filer på automatisk en efter en. Formatet er:

(linienummer) CLOSE
Hvis man samtidig skal starte på et nyt program, kan man benytte NEW, der sletter programmet i maskinen og lukker samtidig alle filer.
Man må gerne søge at lukke en allerede lukket fil.

CLOSE FILE

Ønsker man kun at lukke én bestemt fil, gøres dette ved ordren CLOSE FILE (2 ord!). Formatet er:

(linienummer) CLOSE FILE(a)
hvor a var brugerens eget navn til filen. Se OPEN FILE.

MAT - statements

MAT - statements kan være nyttige inden for COMAL II, da disse sætninger kan spare en hel masse andre sætninger. Jeg vil i det følgende berøre følgende sætninger:

```
MAT X=ZER
MAT X=ZER(a,b)
MAT INPUT X
MAT INPUT(a,b) X
MAT PRINT X
MAT - beregninger
```

Men først: Hvad er en matrix (matricer i flertal)? En matrix opfattes i maskinen som en dobbelt indiceret variabel, hvor man blandt andet kan udføre specielle beregninger, som matrixmultiplikation osv. Da dette ikke giver sig ud for at være en lærebog i hverken matematik eller datamatik, vil jeg overlade matrixmultiplikationerne o.a til speciel litteratur. Her vil jeg kun berøre de matrixberegninger, der kan benyttes som en lettelse i COMAL II.

1) MAT X=ZER

Antager vi, at vi har en dobbelt indiceret variabel A, der er erklæret til (2Ø,3Ø), og som vi ønsker at nulstille, kunne det gøres således:

```
1Ø FOR X=1 TO 2Ø
2Ø   FOR Y=1 TO 3Ø
3Ø     LET A(X,Y)=Ø
4Ø   NEXT Y
5Ø NEXT X
```

Men der er den ulempe, at dette tager (forholdsvis) lang tid, og at datamaten i den tid ikke udfører andet end at fylde nuller ind i variablen A. Det er både nemmere, hurtigere, mere effektivt, mere pladsbesparende og også mere elegant at gøre således:

```
1Ø MAT A=ZER
```

Som også nulstiller A, men med en helt anden fart!

2) MAT X=ZER(a,b)

Vil man på den samme linie både dimensionere og nulstille A, kan det ikke gøres på anden måde end ved brug af:

```
1Ø MAT A=ZER(2Ø,3Ø)
```

Som igen udfører en nulstilling, men <u>først</u> dimensionerer variablen A til (2Ø,3Ø). Skulle man gøre det uden brug af MAT - sætninger ville programmet fylde mindst 6 linier; ville man benytte end anden MAT - sætning, ville der gå 2 linier til. Her klares det på én!

3) MAT INPUT X

Har man en dobbelt indiceret variabel, HANS, der er erklæret til (2,3Ø), og ønsker man at sætte ialt 6Ø tal ind i den - med andre ord, fylde variablen op - kan dette gøres ved:

```
1ØØ FOR X=1 TO 2
11Ø   FOR Y=1 TO 3Ø
12Ø     INPUT HANS(X,Y)
13Ø   NEXT Y
14Ø NEXT X
```

Men dette kan igen klares ved kun 1 MAT - sætning:

```
1ØØ MAT INPUT HANS
```

der bliver ved med at sætte spørgsmålstegn på skærmen, indtil hele variablen er fyldt op med tal. Input'tet starter med HANS(1,Ø), så HANS(1,1) osv.

4) MAT INPUT(a,b) X

Som både dimensionerer og starter inputtet af en dobbelt indiceret variabel. Eksemplet fra før:

```
1Ø MAT INPUT(2,3Ø) HANS
```

Er variablen allerede dimensioneret, når man benytter MAT INPUT(a,b), skal a og b være mindre end de værdier, som man dimensionerede med:

```
1Ø DIM A(25,4)
2Ø MAT INPUT(24,3) A
```

5) MAT PRINT X
Ønsker man at udskrive indholdet af en dobbelt indiceret variabel kan dette gøres sådan (variabelnavnet er B og dimensioneret til (2,3)):

```
1Ø FOR X=Ø TO 2
2Ø    FOR Y=Ø TO 3
3Ø       PRINT B(X,Y),
4Ø    NEXT Y
5Ø    PRINT
6Ø NEXT X
```

Hvis B kun indeholder nuller får vi følgende udskrift:

```
Ø             Ø             Ø             Ø
Ø             Ø             Ø             Ø
Ø             Ø             Ø             Ø
```

Men det samme billede kan opnås meget hurtigere ved 1 enkelt MAT-sætning:

```
1Ø MAT PRINT B
```

Ordet PRINT skal skrives i ét ord, ikke noget med forenklinger!
Har man MAT PRINT sætninger i sit program, og køres dette med RUNL, vil disse sætninger dog ikke blive skrevet ud. Ønskes dette, bliver man nødt til at gøre sådan:

```
1Ø OPEN FILE(1,1),"$LPT"
   ================
15Ø MAT PRINT FILE(1),B
```

Det er ikke tilladt - som i normale PRINT sætninger - at lade maskinen udføre MAT- eller andre beregninger i en MAT PRINT sætning. Kun matricer må skrives ud!
Det er tilladt at skrive:

```
MAT PRINT A,B
```

Maskinen skriver først matrix A ud, dernæst 1 tom linie, og til sidst matrix B.

6) MAT-beregninger
Som tidligere nævnt vil jeg ikke berøre teorierne bag matricemultiplikation eller do. addition (eller subtraktion), andet end jeg vil vise, hvordan man skriver ordrene til disse opperationer (plus til andre opperationer).

```
MAT A=B*C
```

Dette udtryk multiplicerer de 2 matricer B og C, og anbringer resultatet i matrixen A. B og C skal være dimensioneret ens. Matrix A behøver ikke at være dimensioneret, dette sker automatisk, når resultatet anbringes i den (dette gælder iøvrigt også fremover).
Det er muligt at addere og subtrahere matricer:

```
MAT A=B+C
MAT A=B-C
```

Man må ikke skrive:

```
MAT A=B*C; B=A + C                    el. lign.
```

Vil man gange et tal/udtryk/variabel ind i alle en dobbelt indiceret variabels forskellige adresser gøres dette således:

```
MAT A=(2)*B
MAT B=(K)*A
MAT A=(SIN(3)/2+H)*B
```

Parantesen omkring udtryk/tal/variabel kan ikke udelades!
Man kan ikke dividere en matrix med et tal/udtryk/variabel; dette må gøres sådan:

```
MAT A=(1/3)*B
```

Ved omtrent samme metode kan man addere og subtrahere tal osv. ind i en matrix:

```
MAT A=(H) + B
MAT B=(F) - B
```

Maskinen kan kun opperere med 2 matricer eller tal/matrix ad gangen. Derfor er følgende ulovligt:

```
MAT A=B*C*D
```

WHILE - ENDWHILE

WHILE statementet tilhører samme kategori som REPEAT statementet og til en vis grad også FOR - NEXT statementet; forstået på den måde, at WHILE statementet også er en relationssløjfe, der udfører en programdel et antal gange, indtil et kriterium er opfyldt. Formatet er således:

```
linienummer WHILE relationssætning THEN DO
   ================
    Programdel
   ================
linienummer ENDWHILE
```

I modsætning til de to andre sløjfer, der stopper udførslen af programdel, når relationssætningen bliver sand, så stopper WHILE - sløjfen kun, når relationssætningen bliver falsk! Eks.:

```
1Ø LET S=Ø
2Ø INPUT A
3Ø WHILE A>1Ø THEN DO
4Ø    LET S=S+S/2; A=A-2
5Ø ENDWHILE
6Ø PRINT S,A
```

I et WHILE statement kan alle relationstegn og logiske forbindelser bruges. Det er tilladt at brugestrengvariable i WHILE statementet.
Iøvrigt henvises til REPEAT - UNTIL og FOR - NEXT, da disse sløjfer minder en hel del om WHILE - ENDWHILE.

GOTO

Ønsker man i løbet af et program at springe til et andet sted i det samme program, kan dette gøres ved programordren GOTO. Ordren kaldes for en ubetinget hopordre; der er ingen betingelser, der skal være opfyldt før maskinen går til det linienummer, der er efter ordet GOTO. Eks.:

```
1Ø LET A=2
2Ø LET A=A+A/SIN(A)
3Ø PRINT A
4ø GOTO 2Ø
```

DEF

Ønsker man at benytte et - mere eller mindre - komplekst udtryk, hvori højest 2-3 variable indgår, og forefindes dette udtryk flere gange (med skiftende variabelværdier), så kan man benytte en selvdeffineret funktion. Formatet af en sådan selvvalgt funktion ser sådan ud:
(linienummer) DEF FN(bogstav)(variabelnavn)= udtryk
Efter FN skal der stå et selvvalgt bogstav, der angiver navnet på funktionen. Der er kun tale om følgende funktionsnavne:
FNA,FNB,FNC, ... ,FNW,FNX,FNY,FNZ (ialt 26)

Variabelnavnet i parantesen skal indgå i udtrykket! Eks.:

ulovligt

```
DEF FNA(X)=S+4
DEF FNB(G)=RND(Ø)
```

gyldigt

```
DEF FNA(X)=X+4
DEF FNB(G)=RND(G-G)
```

Det er tilladt at have DEF sætninger af følgende udseende:

```
DEF FNA(X)=X+A/B
```

Her skal A og B være tildelt værdier inden et udtryk, hvori funktionsnavnet indgår, bliver beregnet.
Når en selvvalgt funktion er deffineret (selvDEFfineret FunktioN), kan den benyttes på nøjagtig samme måde som en BUILT-IN funktion. Dog må man ikke benytte LEN-funktionen i en selvvalgt funktion. Det er forbudt at benytte strengvariable til funktionsargument.
Eks.:

```
1Ø DEF FNA(X)=X/3+A
2Ø INPUT A
3Ø FOR I=1 TO 5
4Ø   LET Z=FNA(I)+Z; Z=Z/A
5Ø   PRINT Z
6Ø NEXT I
7Ø GOTO 1Ø
```

Det er ikke tilladt at skrive

```
DEF FNA(X)=X+3, FNB(D)=RND(D)
```

el. lign. Reglen siger: Een DEF sætning pr. linie!

REPEAT - UNTIL

REPEAT statementet er en relationssløjfe, noget lig FOR statementet (s.d.). Ligesom FOR - NEXT sløjfen bliver en programstump - nemlig dén, der er inde i sløjfen - udført et antal gange, indtil et bestemt kriterium er opfyldt. Men hvor der ved FOR - NEXT ikke var mulighed for at gøre brug af andre relationstegn end "lig med" (=), så kan man i REPEAT statementet benytte alle relationstegn, inklusive de logiske forbindelser. Formatet:

```
linienummer REPEAT
   ========
    Programdel
   ============
linienummer UNTIL relationssætning
```

Programdelen bliver udført indtil relationssætningen i UNTIL delen er sand; herefter går maskinen videre til næste sætning efter UNTIL sætningen.
Eks.:

```
1Ø INPUT A
2Ø REPEAT
3Ø   LET A=A-1; N=A↑2/(N+1)
4Ø UNTIL A=Ø OR N>=1ØØØ
5Ø PRINT N
6Ø GOTO 1Ø
```

Det er ikke svært at se, hvad dette lille program udfører. Oversætter man ganske simpelt ordene i programmet får man det at vide!
Det er en dårlig idé at konstruere programmer, som:

```
1Ø LET B=1Ø
2Ø REPEAT
3Ø   LET B=B-1
4Ø   PRINT B
5Ø UNTIL B=B+1 OR B<>B
```

Dette program vil køre i det uendelige, med mindre man stopper kørslen.
Hvorfor er ikke svært at se.
REPEAT statementet kan benyttes til meget. Også strengvariable er tilladt at benytte i UNTIL sætningen:

```
   ============
1ØØ REPEAT
   ============
2ØØ UNTIL A$="SLUT"
```

Her er et lille "afslapningsprogram", hvor jeg bruger REPEAT statementet.

```
1Ø LET TAL=INT(RND(Ø)*1ØØ)
2Ø PRINT"JEG TAENKER PAA ET TAL MELLEM Ø OG 1ØØ. HVILKET?"
3Ø REPEAT
4Ø   INPUT GAET
5Ø   PRINT"ER DU NU SIKKER PAA DET?"
6Ø UNTIL TAL=GAET
7Ø PRINT"JOE, DEN ER GOD NOK! JEG TAENKTE PAA"; TAL
8Ø PRINT
9Ø GOTO 1Ø
```

INPUT

Ved ordren INPUT tilkendegiver man, at man på det aktuelle sted i eller udenfor programmet ønsker manuelt at tildele nogle variable et antal værdier, og det være sig alle former for variable. Ved Matricer benyttes dog en særlig konstruktion, se MAT-statement. Formatet er således:

(linienummer) INPUT ("tekst",) variabela (, variabelb/("tekst",),...)

Formatet ser måske lidt kompliceret ud, så et eksempel vil være en hjælp:
1Ø INPUT"TAL: ",A," TEKST: ",A$;
Teksten i anførselstegnene vil blive trykt som ved en PRINT sætning, derefter venter maskinen på at et tal skal skrives ind. Dette tal tildeles variablen A. Herefter skrives ordet TEKST ud, og maskinen venter nu på en værdi til at tildele A$. Endelig bevirker semikolonnet til slut, at det næste, der bliver udskrevet - eller den næste INPUT sætning - begynder på den samme linie, som maskinen lige har skrevet på.

Skrives kun INPUT A$ vil maskinen sætte et spørgsmålstegn (?) for at tilkendegive, at den venter på en værdi at tildele A$.
Skriver man INPUT A,B,C vil der ved indtastning af kun 1 tal, efterfulgt at car return, blive skrevet endnu et (?) ud. Man kan i stedet for her 3 tastninger af car return skrive tallene ind, adskildt af kommaer. Eks.:
1Ø INPUT A,B,C,"OG DET SIDSTE: ",D
RUN
? H/? 3,4,5 car returnOG DET SIDSTE: 6
Det er tilladt at skrive maskinkoder i anførselstegnene ved INPUT, ligesom det er tilladt at, ved strengvariable, inputte maskinkoder i variablene.

PRINT FILE

Vil man skrive på en fil (enten $PTP eller $LPT), det være sig med ord el. tal, skal man benytte ordren PRINT FILE. Dette er dog forudsat af, at den aktuelle fil, man ønsker at skrive på, er åbnet (se OPEN FILE).
Alt, hvad man kan skrive i almindelige PRINT sætninger, kan også skrives med en PRINT FILE. Formatet er:

(linienummer) PRINT FILE(a), ønsket udskrift (se PRINT)

a er det selvvalgte navn til den aktuelle skrivefil. Eks.:

```
1Ø OPEN FILE(4,1),"$LPT"
2Ø PRINT FILE(4), USING"#.##  ER ET TILFAELDIGT TAL ",RND(Ø)*1Ø
```

I Et program, hvori der findes PRINT FILE til $LPT'en (linieskriveren), må ikke køres med RUNL. I så fald vil der fremkomme en fejludskrift.
Eks.:

```
1Ø OPEN FILE(1,1),"$LPT"
2Ø PRINT"DETTE KOMMER PAA SKAERMEN"
3Ø PRINT FILE(1),"OG DETTE PAA LINIESKRIVEREN."
```

Ønsker man at benytte beregnede el.lign. i et andet program kunne dette gøres ved at lade datamængden blive skrevet ud på linieskriveren (evt. ved en RUNL), og derefter taste den ind via traditionelle INPUT sætninger i det nye program. Men hvis der er tale om store mængder bliver dette dels for langsomt for datamaten, dels for unøjagtigt, og dels for trættende. Man kan da benytte PRINT FILE, og få dataene hullet ud på strimmel. Eks.:

```
1Ø OPEN FILE(1,1),"$PTP"
2Ø FOR I=Ø TO Ø.5 STEP Ø.Ø1
3Ø   PRINT FILE(1),SIN(I)
4Ø   PRINT FILE(1),COS(I)
5Ø NEXT I
```

Ved sådanne tilfælde må man ikke sammenskrive linie 3Ø og 4Ø til en enkelt linie 3Ø PRINT FILE(1),SIN(I),COS(I), ligesom man heller ikke må skrive
3Ø PRINT FILE(1),SIN(I),

Når man ønsker at indlæse disse værdier fra hulstrimmel gøres dette ved brug af INPUT FILE.

INPUT FILE

Man kan læse på følgende 2 filer: $PTR og $CDR, men på nuværende tidspunkt virker $CDR'en (hulkortlæseren) ikke tilfredsstillende. Når den kommer til det, vil principperne være det samme som for $PTR.
For at kunne læse på en af disse filer, er det nødvendigt at åbne dem. Eks.:

```
1Ø OPEN FILE(1,3),"$PTR"
2Ø INPUT FILE(1),A
```

INPUT FILE virker på samme måde som traditionel INPUT, bortset fra, at det ikke er muligt at skrive tekst i en INPUT FILE sætning. Formatet er:

(linienummer) INPUT FILE(a), variabel a, variabel b, ...

hvor a'et er det selvvalgte filnavn. Eks.:

```
1Ø OPEN FILE(5,3),"$PTR"
2Ø INPUT FILE(5),A,B
3Ø PRINT A,B,"TABEL"
4Ø GOTO 2Ø
```

(Eventuelt kan dette benyttes til det under PRINT FILE viste program.)

GOSUB - RETURN

Hvis et problem, der skal databehandles, kan nedbrydes i flere underproblemer, eller hvis flere af disse underproblemer stort set er ens; er det ofte en lettelse at opbygge et program med underrutiner (subrutiner, procedurer). Dette gør også programmet mere overskueligt og mere logisk at overskue. I COMAL II findes der følgende former for underrutiner:

1 GOSUB - RETURN
2 IF - GOSUB - RETURN
3 ON - GOSUB - RETURN
4 EXEC - PROC - ENDPROC
5 IF - EXEC - PROC - ENDPROC

I praksis taler man om de to proceduregrupper, nemlig GOSUB statementet og EXEC statementet. Her i det følgende beskæftiger vi os med den rene underrutine, GOSUB - RETURN. Formatet for dette er således:

```
linienummer GOSUB linienummera
            ==================
linienummera ...
             ==================
linienummer RETURN
```

Linienummer a er linienummeret på den første sætning i underrutinen. En underrutine må kun aktiveres ved kaldeordet (her: GOSUB), og kun forlades ved returneringsordet (her: RETURN). Man må ikke mødes med et returneringsord uden først at have mødt det tilsvarende kaldeord. Derfor er dette forkert:

```
1Ø INPUT A
2Ø GOSUB 3Ø
3Ø PRINT A
4Ø RETURN
```

RETURN bevirker, at maskinen vender tilbage til linien umidelbart efter GOSUB sætningen. Dette vil her bevirke, at den (datamaten) går videre til linie 3Ø, der vil blive udført normalt, men i linie 4Ø stopper maskinen med en fejludskrift. Skal programmet kunne køre, må det ændres, for eksempel sådan:

```
1Ø INPUT A
2Ø GOSUB 4Ø
3Ø GOTO 1Ø
4Ø PRINT A
5Ø RETURN
```

Det er tilladt for en underrutine - ligegyldigt i hvilken proceduregruppe, den er i - at kalde et ubegrænset antal andre underrutiner af begge grupper, dog ikke sig selv. Dette med den klausul, at alle de kaldte subrutiner skal have et returneringsord (skal være lovlige rutiner). Eks.:

```
1Ø INPUT A
2Ø GOSUB 4Ø
3Ø GOTO 1Ø
4Ø GOSUB 6Ø
5Ø RETURN
6Ø GOSUB 8Ø
7Ø RETURN
8Ø GOSUB 1ØØ
9Ø RETURN
1ØØ PRINT A
11Ø RETURN
```

Her er subrutinen: 4Ø - 5Ø, 6Ø - 7Ø, 8Ø - 9Ø, 1ØØ - 11Ø. Det er endda tilladt at have flere, datamaskinen skal nok finde rundt i dem!
Det er ikke nødvendigt at have underrutiner stående i rækkefølge, de må bare ikke kollidere med det egentlige program.

IF - GOSUB - RETURN

GOSUB kaldes lige som GOTO for en ubetinget hopordre. Hvis man i stedet for det ubetingede spring ønsker et betinget hop til en underrutine, kan man benytte en relationssætning, der minder meget om IF - GOTO, nemlig sætningen IF - GOSUB. Formatet er således:

linienummer IF relationssætning (THEN) GOSUB linienummera
==========

linienummera ...
==========

linienummer RETURN

Linienummer a er linienummeret på den første sætning i underrutinen. Underrutinen er nøjagtig af samme slags som under GOSUB - RETURN (s.d.).

Eks.:

```
10 DIM A$(30)
20 INPUT A
30 GOSUB 50
40 GOTO 20
50 IF A<=0 THEN GOSUB 90
60 LET Y=LOG(A)/3 + 1; X=SIN(Y) + A/2
70 PRINT"X=",X,"Y=",Y
80 RETURN
90 PRINT
100 INPUT"QNSKER DU AT FORTSAETTE?  ",A$
110 IF A$="JA" THEN GOTO 140
120 PRINT"SAA NEW'ER VI!"
130 NEW
140 LET A=ABS(A) + 1
150 RETURN
```

ON - GOSUB - RETURN

Dette er også en slags betinget hopordre. Her kan vi henvise til en liste af subrutiners linienumre på een gang. Formatet er:

linienummer ON udtryk/tal/variabel (THEN) GOSUB liste af linienumre
==========

linienummera ...
==========

linienummer RETURN
==========

linienummerb ...
==========

linienummer RETURN

osv.

Linienumrene a, b osv er linienumrene på de første sætninger i de respektive underrutiner, der henvises til i ON - GOSUB sætningen.

Da princippet ved dette statement er det samme som ved ON - GOTO henvises der hermed til denne for nærmere forklaring.

IF - GOTO

Programordren GOTO er en ubetinget hopordre, dvs. en hopordre, der træder i funktion spontant. Ønsker man at bygge sit program op på en sådan måde, at der kun springes til en bestemt del af programmet når en bestemt relation er opfyldt, så må man benytte IF - GOTO. Formatet ser sådan ud:

linienummer IF relation (THEN) GOTO linienummera

Linienummera er den sætning, hvortil der ønskes at springe hen, når relationen er opfyldt. I modsat fald går maskinen videre med linien efter relationssætningen.

Når ordet THEN står i parantes, betyder dette (og det gælder iøvrigt alle de steder, hvor jeg har sat THEN i en rød parantes), at det er valgfrit, om man vil skrive:

1) GOTO
2) THEN eller
3) THEN GOTO (aldrig GOTO THEN!)

Eks.:

```
1Ø IF A=3 GOTO 5Ø
2Ø IF A=5 GOTO 6Ø
3Ø IF A THEN GOTO 8Ø
```

I linie 3Ø står der kun IF A. Dette vil give spring til linie 8Ø, når A er forskellig fra nul. Dvs. A=Ø vil ikke give noget spring.

IF A GOTO ... kan benyttes som en slags erstatning for de ikke-eksisterende Boolske variable. Eks.:

```
IF A OR B THEN 3Ø
IF A AND B=Ø THEN GOTO 4Ø
```

IF - THEN

Ved hjælp af IF - GOTO kan man hoppe til en bestemt del af programmet, når en relation er sand. Men til tider kunne det være rart at have enordre, der slår to (eller flere) linier sammen. Lad os se på et eksempel:

```
1Ø INPUT A
2Ø IF A Ø GOTO 6Ø
3Ø IF A=Ø GOTO 8Ø
4Ø PRINT" A= ";A," OG LN(A)= "; LOG(A)
5Ø GOTO 1Ø
6Ø LET A=ABS(A)
7Ø GOTO 4Ø
8Ø LET A=1E-3Ø
9Ø GOTO 4Ø
```

Det bliver et ret så stort program. Men ved brug af IF - THEN kan det skæres ned til:

```
1Ø INPUT A
2Ø IF A Ø THEN LET A=ABS(A)
3Ø IF A=Ø THEN LET A=1E-3Ø
4Ø PRINT" A= ";A," OG LN(A)= "; LOG(A)
5Ø GOTO 1Ø
```

Altså 4 linier kortere! Formatet af en IF - THEN sætning er:

(linienummer) IF relation THEN COMAL II sætning

Som COMAL II sætning kan bruges PRINT, LET, INPUT, NEW, BYE osv., bare ikke MAT-statements. Iøvrigt, se under IF - GOTO.

ON - GOTO

Sætningen ON - GOTO er en speciel form for relationssætning, da det eneste relationstegn, der findes her, er lighedstegnet (=). En ON - GOTO's virkemåde er i princippet sådan:
I stedet for:

```
1Ø IF A=1 GOTO 2ØØ
2Ø IF A=2 GOTO 25Ø
3Ø IF A=3 GOTO 2ØØ
4Ø IF A=4 GOTO 3ØØ
5Ø IF A=5 GOTO 6ØØ
```

Kan man ved brug af ON - GOTO skrive:

```
1Ø ON A GOTO 2ØØ, 25Ø, 2ØØ, 3ØØ, 6ØØ
```

Hvis A er mindre end 1 vil programmet fortsætte med linien efter ON - GOTO
Er A et tal fra og med 1 og (her) mindre end 6 vil programmet - efter at der er blevet udført en INT(A) (dette sker automatisk) - fortsætte med at springe til nummer INT(A) linienummer i rækken.
Er A større eller lig med 6 fortsætter maskinen med linien efter ON - GOTO

Formatet ser således ud:

linienummer ON udtryk/tal/variabel (THEN) GOTO linienummera, linienummerb

Et eksempel (taget fra BASIC-lærebogen, og tilrettelagt efter COMAL II) viser praktisk brug af ON - GOTO. Opgaven lyder: For en række tal i DATA sætninger ønskes optalt, hvor mange der er hhv. negative, nul og positive.

```
1Ø DATA -17, -13, Ø,Ø, 7, 1, 3, Ø, -5, -1, 13, 1E3Ø
2Ø READ TAL
3Ø IF TAL=1E3Ø GOTO 11Ø
4Ø ON SGN(TAL)+2 THEN GOTO 5Ø, 7Ø, 9Ø
5Ø LET NEGATIV=NEGATIV+1
6Ø GOTO 2Ø
7Ø LET NUL=NUL+1
8Ø GOTO 2Ø
9Ø LET POSITIV=POSITIV+1
1ØØ GOTO 2Ø
11Ø PRINT"ANTAL NEGATIVE TAL: ",NEGATIV
12Ø PRINT"ANTAL NUL          : ",NUL
13Ø PRINT"ANTAL POSITIVE TAL: ",POSITIV
14Ø LET NEGATIV=Ø; NUL=Ø; POSITIV=Ø
15Ø END
```

Ønskes samme program udført med ON - GOSUB, skal følgende rettelser gøres:

Linierne 6Ø, 8Ø, 1ØØ skal hedde RETURN i stedet for GOTO 2Ø
Linie 4Ø: ON SGN(TAL)+2 GOSUB 5Ø, 7Ø, 9Ø
Ny linie: 41 GOTO 2Ø

Herefter virker programmet på basis af subrutiner, proceduregruppe 1.

Det bemærkes, at dette specielle tilfælde kan gøres kortere og mere håndgængeligt ved brug af IF - THEN sætninger. Prøv selv!

SELECT - CASE - ENDCASES

Et statement, der minder meget om ON - GOTO sætningen, og som har en del tilfældes med statementet ON - GOSUB (se disse) er multiforgreningsstatementet SELECT - CASE - ENDCASES. Da det ville være en alt for stor mundfuld at lave een enkelt Format, har jeg delt det op i tre, som jeg gennemgår efterhånden.

I SELECT-delen
Formatet er sådan:

linienummer SELECT tal/udtryk/variabel (THEN) DO

Som ved en ON - GOTO har man her den aktive del. Men i modsætningen til ON sætningen, der styrer programmet hen til programlinier (andre linienumre), så henviser SELECT sætningen programmet til rutiner; de såkaldte CASE's. Hvor en ON - GOTO har en indbygget INT-funktion, gælder dette ikke om SELECT - DO- delen. SELECT-delen virker skematisk således:
Sætningen: 1Ø SELECT A THEN DO

Er A mindre end eller lig med nul fortsættes med den næste sætning, der er efter ENDCASES.
Er A større end nul, og indeholdt i CASE'ne, fortsætter programmet med den respektive rutine (CASE). Ellers med næste sætning efter ENDCASES.
Det er tilladt at skrive:
1Ø SELECT o.ool THEN DO
der henviser til rutinen med navnet o.ool.

II CASE-delen
Formatet er sådan:

linienummer CASE tal (udtryk/variabel?)
==========
rutinedel
==========

Case indleder rutinedelen. Der intet i vejen for, at en CASE kan benytte andre SELECT - CASE - ENDCASES; det maximale antal af multiforgreninger er dog 7 inde i hinanden. En CASE minder meget om en underrutine af proceduregruppe 2; der er et kaldenavn (tallet, der skrives efter ordet CASE), og der er ikke et rigtigt returneringsord at tage hensyn til. Endvidere er der ingen grænser for, hvor stor en CASE må være i sin udstrækning. Man må ikke komme til en CASE på anden måde end ved SELECT sætningen (dette er endnu en ting, der minder om proceduregruppe 2). Af slutningen af en rutine gøres enten ved den næstfølgende CASE (Eks.:

```
5ØØ CASE 1
51Ø   PRINT"SLUT"
52Ø CASE 1.5
```

eller ved brug af ENDCASES. Tallene, der rent faktisk er kaldenavnene, skal være større end nul (såvidt vides på nuværende tidspunkt).

III ENDCASES
Formatet ser sådan ud:

Ikke nødvendigvis!
Det er ligeledes tilladt at benytte variable her i stedet for konstanter.

linienummer ENDCASES

ENDCASES afslutter konsekvent hele SELECT - CASE - ENDCASES statementet. Der må ikke henvises til ENDCASES. Eks.:

```
1Ø INPUT A
2Ø FOR I=Ø.1 TO Ø.5 STEP 0.1
3Ø   SELECT I THEN DO
4ø   CASE Ø.1
5Ø     PRINT A/I
6Ø   CASE Ø.3
7Ø     PRINT I/A
8Ø   ENDCASES
9Ø NEXT I
```

IF - THEN DO

IF - THEN DO er et statement, der på sin vis slår IF - THEN sammen med en slags rutiner, som man kender fra SELECT - CASE - ENDCASES. Hvis man ser på, hvordan et sådant statement virker, kan dette vises ved et eksempel:

```
10 IF A=3 GOTO 40
20 PRINT "A ER FORSKELLIG FRA 3"
30 GOTO 50
40 PRINT "A ER LIG MED 3"
50 REM HER SLUTTER RELATIONSSLQJFEN
```

Erstattes dette lille program med et tilsvarende IF - THEN DO, ser man tydelig ligheden:

```
10 IF A<>3 DO
20   PRINT"A ER FORSKELLIG FRA 3"
30 ELSE
40   PRINT"A ER LIG MED 3"
50 ENDIF HER SLUTTER RELATIONSSLQJFEN
```

Man ser, at linierne mellem IF sætningen og ELSE sætningen kun udføres, når relationen i IF sætningen er sand. Linierne mellem ELSE sætningen og ENDIF sætningen udføres kun når relationen i IF sætningen er falsk.
Når én af disse rutiner er udført, fortsætter programmet med linien efter ENDIF. Bemærk iøvrigt, at det er tilladt at skrive bemærkninger som tilsvarendende REM bemærkninger efter ENDIF.

En IF - THEN DO må gerne benytte flere IF - THEN DO i sine rutiner. Eks.:

```
10 INPUT B
20 IF SGN(B)=-1 THEN DO
30   PRINT"B ER MINDRE END NUL"
40 ELSE
50   IF SGN(B)=0 THEN DO
60     PRINT"B ER LIG NUL"
70   ELSE
80     PRINT"B ER STQRRE END NUL"
90   ENDIF
100 ENDIF
```

Et sådant eksempel som dette vil blive betydelig kortere ved at benytte IF - THEN sætninger i stedet (det færdige program vil da fylde 4 linier), men alligevel kan det siges, at IF - THEN DO er en af de mest brugte relationsstatements i COMAL II!

EXEC - PROC - ENDPROC
(Sammen med IF - THEN EXEC)

Dette er en underrutine, der tilhører proceduregruppe 2, dvs. en underrutine, der ikke kaldes ved linienummeret, men derimod ved subrutinens navn. Navnet på en sådan underrutine er selvvalgt, og tilhører variabelnavne. Det er tilladt at benytte en variabels navn, som man benytter i sit program, som navn for subrutinen. Sådanne navne kan f. eks være:
CHECK, HANSENS, KONTO, A, A1 (der kendes forskel på A og A1)(generelt eks.)

I modsætning til GOSUB-rutinen, der kaldes ved ordet GOSUB, efterfulgt af linienummeret på lokaliteten, hvor underrutinen befinder sig, kaldes en subrutine af proceduregruppe 2 her ved ordet EXEC, efterfulgt af subrutinens navn. Formatet til EXEC er sådan:

linienummer EXEC navn

Eks.:
5Ø EXEC KONTO
6Ø EXEC HANSES
7Ø IF A=5 THEN EXEC KONTO
Dette sidste var et eksempel på IF - THEN EXEC. Da det efterfølgende er generelt skrevet, vil jeg ikke gentage med hensyn til IF - THEN EXEC.
Da hver underrutine i proceduregruppe 2 har et navn og en slutning, er det ikke nødvendigt at tænke så meget over, hvor de skal anbringes. Jeg selv anbringer mine subrutiner i slutningen af programmet; men følger man den regel, at det er ikke tilladt at komme ind i subrutinen på anden måde end ved EXEC, og det er ikke tilladt at forlade den på anden måde end ved ENDPROC (ved henvisning til en anden subrutine i subrutinen vender man jo tilbage igen), kan man anbringe disse rutiner overalt i programmet.
Det, der indikerer navnet på en subrutine i gruppe 2, og samtidig starten på den, er ordet PROC (læs: PROCedure). Formatet er:

linienummer PROC navn

hvor navnet er det selvvalgte navn til underrutinen (ikke et COMAL II ord).
Eks.: 15Ø PROC KONTO
Herefter følger så underrutinen. Når den ikke er længere, vendes tilbage til linien umiddelbart under EXEC med ordet ENDPROC, hvis format er:

linienummer ENDPROC (tekst)
(tekst) angiver, at det - som ved END - er tilladt at skrive tekst efter ordet ENDPROC. Se END.
En underrutine må kalde et ubegrænset antal andre underrutiner, dog undtagen sig selv.

SYS(7)

SYS(7) er en BUILT-IN funktion, der giver oplysninger om den sidste fejl, der er registreret ved brug af datamaten.
Eks.: Indtaster man tegnet @, efterfulgt af car return, og derefter spørger om SYS(7), skal dette give svaret Ø1, hvilket er fejlnummert ved ERROR Ø1 - KAN IKKE GENKENDE ORD . Til tider kan man få et større tal, ved brug af SYS(7), end der findes i fortegnelsen over fejlnumre. Sker dette, er det helt normalt, og indikerer for ERROR Ø2. SYS(7) kan bygges - som andre funktioner - ind i relationssætninger.
*En fortegnelse over fejlnumre er ikke medtaget i denne refferencebog, men kan findes i enten den hvide (EXTENDEDBASIC) eller den orange (BATCHBASIC) manual. Hvis ikke, kan man forsøge sig frem...

* Se tillæg under Fejlmeddelelser

FORENKLINGER

Undervejs med beskrivelserne af de forskellige ord osv. i COMAL II, har der ikke ret tit været taget hensyn til udstyret. Dette være gjort nu. Det følgende skulle være en kort række eksempler på, hvordan man kan forenkle sin indtastning på det udstyr, der er til rådighed, uden at det går ud over meningen og ordvalget af de forskellige sætninger.

Ordet PRINT kan skrives kort som ; (semikolon), undtagen heraf er dog MAT PRINT.
Alle steder, hvor der forekommer THEN kan dette ord udelades (undtagelse, se efter LET).
Ordet LET kan udelades alle steder (undtagelse, sex herunder).
I en relationssætning skal der altid være enten THEN eller LET (eller begge), når det drejer sig om en IF - THEN LET sætning. Eks.:

```
IF A=2 THEN LET A=A + 1          eller
IF A=2 THEN A=A + 1              eller
IF A=2 LET A=A + 1
```

Det sidste anførselstegn (") kan udelades overalt. Eks.:

```
1Ø ; "PRQVE","NR. 1
2Ø OPEN FILE(1,1),"$LPT
```

Dette var en ultrakort liste over eksempler på, hvordan man kan forenkle sin indtastning. Flere sådanne eksempler kan man finde ved omgang med maskinen. Endvidere er visse deciderede COMAL II ord brugt til at forenkle arbejdet for programmøren: Tænk bare på ordet AUTO...

==========

AFSLUTNING

Det var min hensigt at skrive en kort refferencebog over COMAL II til brug både for begyndere og for rutinerede brugere af datamaten. Jeg havde regnet med ikke over 25 sider. Men når man så ser tilbage, og ser, at der er over 45 sider, med 62 artikler, så kan man forstå, at bogen blev så stor, som den blev. Jeg beklager, at jeg ikke fik sat artiklerne i alfabetisk orden, men de mange sider blev skrevet når og kun når jeg havde tid eller lyst til det. Derfor er der ikke noget særligt godt system i det, blot ved jeg, at de mange artikler tilsammen dækker størstedelen af COMAL II. Dette være sagt ganske uden at rødme.
Det er muligt at sætte sig direkte hen til datamaten med bogen her til hjælp, men det vil nok være en fordel at følge med i de studiekurser, der sikkert kommer om BASIC. Dette være sagt til begynderen.
Og til rutinerede: Jeg har ikke set et eneste sted, bortset fra de forskellige stykker papire, som enkelte slæber frem og tilbage til EDB-rummet, hvor både maskinkoder og hulkoder er samlet. Endvidere tillader jeg mig ganske uden hensyntagen til eventuel cencur at medtage det noksom kendte Opladningsprogram - det er det, der altid skal køres, når maskinen blokerer. I sin tid var det kun en tre stykker af eleverne, der fik at vide hvordan. Siden er det vist nok set mere gennem fingre med hvem, der udfører denne opladning - der er i hvert tilfælde ikke blevet løftet nogen pegefinger mod dette. Skulle den side, hvor opladningsproceduren skulle stå på, være væk, så er grunden sikkert cencuren!
Som enhver anden ordentlig bog om et datamatsprog skal der være en litteraturliste. Jeg anbefaler følgende bøger:
BASIC-lærebogen, BASIC-håndbogen, Bogen om EDB, COMAL II (den grønne i EDB-rummet), BASIC for begyndere, BASIC og EDB-orbog. Andre bøger kan fås på biblioteket.

Med datamatisk hilsen
Normann Aaboe Nielsen
1977

TILLÆG TIL HURTIG REFERENCEBOG I

COMAL II

1977
by Bent Brun Kristen-
sen
& Normann Aa. Nielsen


IC DATA PROCESSING MACHINE
IBM
IBM
IBM
ELECTRONIC DATA PROCESSING MACHINE

BB. 2.Ms

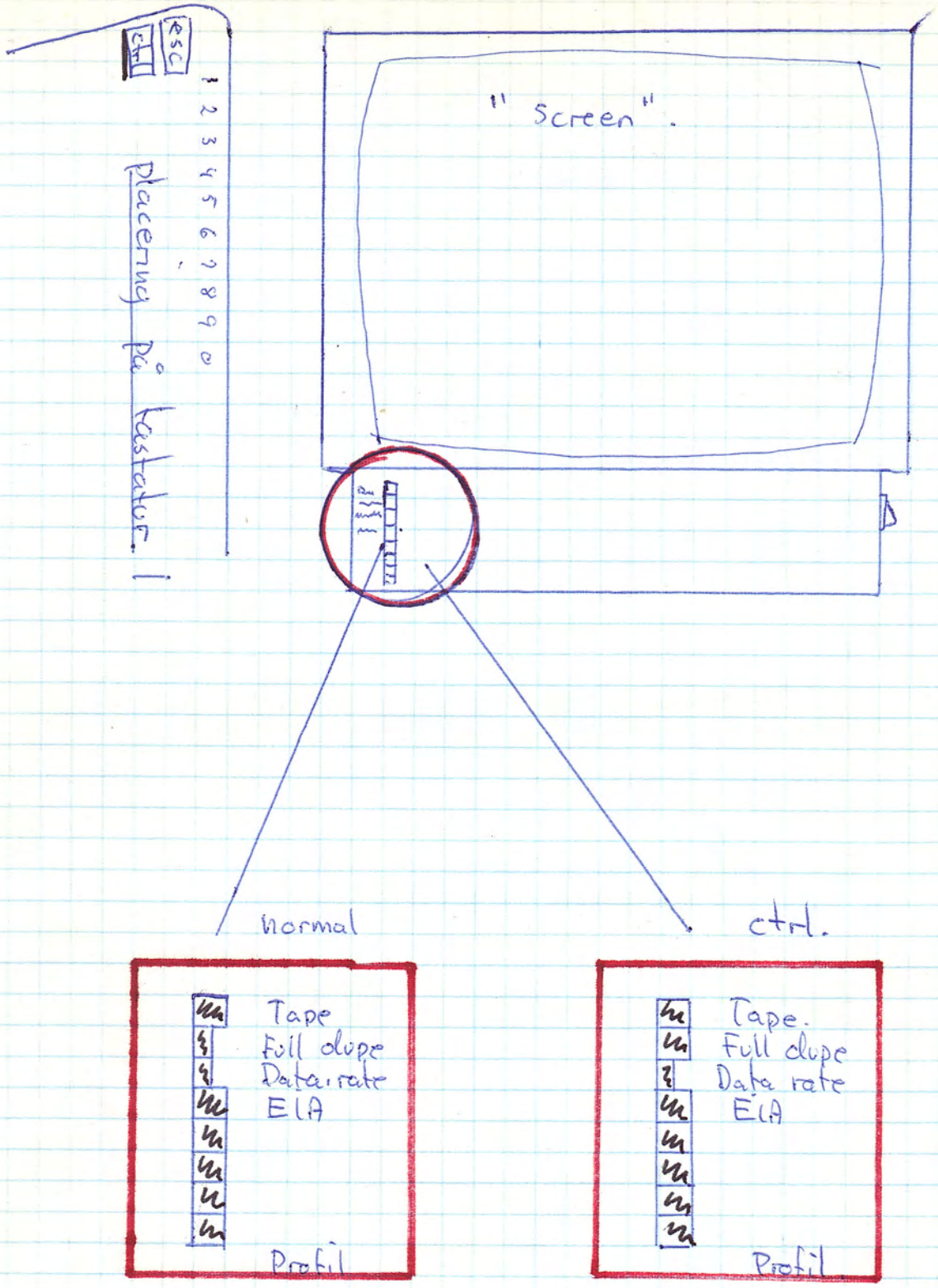
Tillæg til COMAL-instruktionsbogen w.b. Normann Aaboe Nielsen.

Controle.

Koden ctrl efterfulgt af et bogstav,tal,tegn giver visse muligheder m.h.t placering af blinkeren på skærmen.I det nedenstående er en oversigt over de forskellige bogstaver,tal og tegns inflydelse.

ctrl a -skifter linie med *
ctrl f -springer ud til venstre kant,hopper op og rykker en frem og tilbage.
ctrl g -virker som <7> ,hyler.
ctrl h -virker som <8> ,rykker 1 tilbage.
ctrl i -virker med 3 frem
ctrl j -hopper en linie ned.
ctrl l -virker som <12> sletter skærmen.
ctrl m -hopper ud i venstre side og 1 ned.
ctrl x -rykker 1 frem.
ctrl z -virker som <26> ,hopper en op.
ctrl ! -virker som ctrl a
ctrl 6 shift- hopper op til ¼ fra skærmens øverste kant ved 3 på hinanden følgende tryk.
ctrl 7 shift- virker som <7> ,hyler.
ctrl (-rykker 1 tilbage
ctrl 8 - rykker 1 frem.
ctrl 9 -hopper 3 frem.
ctrl * -hopper 1 op.
ctrl * shift -hopper 1 ned.
ctrl : -hopper 2 ned.
ctrl = -rykker helt op i venstre øverste hjørne.
ctrl , -virker som <12> .

Desuden kan nævnes at ctrl med rub-out sletter alt hvad der står efter blinkeren.Controle-tasten findes yderst til venstre på tastaturet lige under escape-tasten.Controle kan hverken bruges som kommando-eller sætningsord,den vil blive opfattet som en variabel.For at få controle til at virke skal knapperne på skærmen stå som vist på figuren næste side,FULL DUPX skal være ude og de andre som normalt. Den nye tekstudskrivninge kan godt virke forvirrende,men når du har placeret blinkeren hvor du vil,kan du slå tilbage til den normale.



Obs: Husk altid at stille tilbage til normal efter brug, for ikke at genere andre.

Normann Aalve Nielsen og Bent Brun 2.Ms.

5/9-84.

Fejlfinding i programmer

Når et program ikke kører som det skal, er der som regel kun én, der har lavet fejlen: Brugeren. Et typisk teknikerord er GIGO - Gabbage In, Gabbage Out - eller på dansk, SISU: Skidt Ind, Skidt Ud. Det skal minde programmører om, at det ikke er maskinen, der klokker i det, men kun de dårlige programmer. Hvis det en sjælden gang ikke er brugeren, der har gjort fejlene er maskinen sikkert blokeret, og må lades op (se under Opstart), men ellers skal brugeren (programmøren) ud på den opgave, der hedder fejlfinding, eller aflusning.
I mange - og det er faktisk de fleste - tilfælde hjælper maskinen selv til ved aflusningen. Den har et righoldigt udvalg af fejludskrifter, som den skriver ud, når fejlen opstår. Denne udskrift sker samtidig med en standsning af det aktuelle program ved netop den linie, hvori fejlen ligger.
I selve fejlmeddelelsen bliver linienummeret skrevet ud, og man kan dels se at meddelelsen (der står på dansk), hvad fejlen er, og også se i hvilken linie, at fejlen optræder. Herved kan man straks LIST'e denne linie og rette fejlene. Man kan dog også fortsætte med CON, hvis fejlen er af underordnet betydning.

Eks.: Maskinen vil i programmet ved RUN stoppe ved linie 2Ø:

```
1Ø LET A=Ø
2Ø PRINT "1/Ø= "; 1/A
```

Idet man jo ikke må dividere med nul. Fejludskriften vil blive:

```
ERROR 16 AT ØØ2Ø - ARITMETISK FEJL
```

Eks.: Maskinen stopper ved linie 2Ø:

```
1Ø LET A=A+3
2Ø PRINT B
3Ø GOTO 1Ø
```

Idet maskinen ikke ved, hvad B er - med mindre B har fået en værdi uden for programmet, som så er startet med CON eller RUN 1Ø. Er dette ikke tilfældet, vil fejlmeddelelsen blive:

```
ERROR 17 AT ØØ2Ø - UKENDT VARIABEL
```

Men nogle gange ligger fejlen ikke i selve den linie, hvori maskinen standser. Dette viser følgende eksempel:

```
1Ø DEF FNA(X)=1/X
2Ø LET A=Ø
3Ø PRINT FNA(A)
4Ø LET A=A+1
5Ø GOTO 3Ø
```

Maskinen vil stoppe i linie 3Ø, da der divideres med nul. Men i dette program er det programmøren, der har gjort en fejl. Der skulle faktisk have stået (tænker vi os):

```
1Ø DEF FNA(X)=1/(X+1)
```

Man ser, at fejlen ikke ligger i linie 3Ø, men derimod i linie 1ø!

Sådanne fejl er de sværeste at finde, særligt i mere komplicerede udtryk.
Eks.:

```
1Ø DEF FNA(X)=SIN(X)+COS(X)
2Ø INPUT A,B
3Ø IF A=B AND FNB(A)=RND(C) THEN GOTO 7Ø
4Ø GOTO 2Ø
5Ø PRINT FNA(B)
6Ø GOTO 2Ø
```

Ved kørsel udfører maskinen følgende:

1) standser ved linie 3Ø, og skriver:
ERROR 17 AT ØØ3Ø - UKENDT VARIABEL
rettelse til: 3Ø IF A=B AND FNB(A)=RND(Ø) THEN GOTO 7Ø
RUN

2) standser ved linie 3Ø, og skriver:
ERROR 32 AT ØØ3Ø - UDEFFINERET FUNKTION
rettelse til: 3Ø IF A=B AND FNA(A)=RND(Ø) THEN GOTO 7Ø
RUN

3) standser ved linie 3Ø, og skriver:
ERROR 13 AT ØØ3Ø - IKKE EKSISTERENDE LINIENUMMER
rettelse til: 3Ø IF A=B AND FNA(A)=RND(Ø) THEN GOTO 5Ø
RUN Hvorefter maskinen skulle køre normalt

En typisk indtastningsfejl - som man plejer at opdage hurtigt - er at indtaste A2 i stedet for A$ (eller omvendt).

Ved indlæsningen af en hulstrimmel, der står i EXTENDED BASIC (der er et lavere BASIC end COMAL II), støder man til tider på, at maskinen af sig selv skriver en række ERROR'er ud, med samtidige henvisninger til linienumre. Lister man den angivne linie, kan man støde på en sådan sætning:

1Ø IF A=3THEN THEN GOTO 5Ø

Her skal det ene THEN selvfølgelig fjernes! Eks.:

1Ø FORI=2 TO-3 STEP-1

Sætningens mening er god nok, man skal bare rette den ind til:

1Ø FOR I= 2 TO -3 STEP -1

Altså med mellemrum efter FOR osv. Det er ikke tilladt at skrive ord tæt sammen i COMAL II, som det er i EXTENDED og BATCH (2 lavere sprog af BASIC).

Når man laver en RENUMBER på et program for eventuelt at få endnu et par linier presset ind i programmet, kan der ske det - særligt, hvis det aktuelle program ikke er færdigskrevet - at maskinen skriver en række linienumre ud i fejlmeddelelser som:
ERROR 13 AT linienr. - IKKE EKSISTERENDE LINIENUMMER
Hvis sætning linienr. LIST'es, kan man som eksempel få følgende:

linienr. GOSUB ØØØØ

Her skal man kende programmet for at vide, hvad der skal stå i stedet for nullerne efter GOSUB!

Ved brug af indicerede variable (i.v.), er den typiske fejl denne:

1Ø DIM A(1Ø)
2Ø LET A(11)=Ø

Her er dimensionen i linie 1Ø blevet overskredet i linie 2Ø. Fejludskrift:

ERROR 31 AT ØØ2Ø - INDEKS OVERSKRIDER DIMENSION

En anden fejl, man kan gøre, er denne:

1Ø DIM A(1Ø)
2Ø MAT A=ZER

Dette er ikke tilladt, selv om det kunne se sådan ud. En matrix er deffineret som en dobbelt i.v.!

Retter man så i programmet fra foregående side til en dobbelt i.v., risikerer man nye fejl:

Eks.: Programmet:

```
1Ø DIM A(1Ø)
2Ø MAT A=ZER
INPUT
XXXXX
3Ø INPUT A(1)
- - - - - - - -
```

rettes til:

```
1Ø DIM A(1,1Ø)
2Ø MAT A=ZER
3Ø INPUT A(1)
- - - - - - - - -
```

Man har glemt i linie 3Ø at rette inputtet til INPUT af dobbelt iv.:
INPUT A(1,1)

Dette sidste (dvs. analogt med foregående eksempel) er meget ubehageligt at skulle rette, hvis man vil have MAT-statements i programmet, evt. til udførelse af nogle MAT-beregninger andetsteds i programmet. Men er dette ikke tilladt, kan man nøjes med den almindelige nulstillingssløjfe:

```
1Ø DIM A(1Ø)
2Ø FOR I=Ø TO 1Ø
3Ø   A(I)=Ø
4Ø NEXT I
5Ø INPUT A(1)
- - - - - - - -
```

En fortegnelse over de mest almindelige fejlmeddelelser findes andetsteds i tillægget.

Fejlmeddelelser

Det følgende er en fortegnelse over de mest almindelige fejludskrifter. En fuldstændig liste kan findes i den engelske EXTENDED BASIC (hvid).

ERROR ØØ - FORMATFEJL
ERROR Ø1 - KAN IKKE GENKENDE ORD
ERROR Ø2 - SAETNINGEN ER IKKE RIGTIGT OPBYGGET
ERROR Ø5 - ULOVLIGT SAETNINGSNUMMER
ERROR Ø9 AT linienr. - ULOVLIGT FILNAVN
ERROR 1Ø AT linienr. - IND-UD-ENHED I BRUG
ERROR 11 - PARANTESFEJL
ERROR 12 - ULOVLIG KOMMANDO
ERROR 13 AT linienr. - IKKE EKSISTERENDE LINIENUMMER
ERROR 15 AT linienr. - IKKE FLERE DATA TIL READ
ERROR 16 AT linienr. - ARITMETISK FEJL
ERROR 17 AT linienr. - UKENDT VARIABEL
ERROR 19 AT linienr. - RETURN UDEN GOSUB
ERROR 2Ø AT linienr. - FOR MANGE FOR INDE I HINANDEN
ERROR 22 AT linienr. - NEXT UDEN FOR
ERROR 23 AT linienr. - INGEN LAGERPLADS TIL VARIABLE
~~ERROR 24 AT linienr. - UDTRYK FOR KOMPLICERET~~
ERROR 28 AT linienr. - FORSTE DIMENSIONERING OVERSKREDET
ERROR 31 AT linienr. - INDEKS OVERSKRIDER DIMENSION
ERROR 32 AT linienr. - UDEFFINERET FUNKTION
ERROR 34 AT linienr. - ULOVLIGT FUNKTIONSARGUMENT
ERROR 38 AT linienr. - STRENGVARIABEL IKKE DIMENSIONERET
ERROR 4Ø AT linienr. - MATRICER HAR FORSKELLIG STORRELSER
ERROR 41 AT linienr. - MATRIX HAR EN NULDIMENSION
ERROR 44 AT linienr. - FILEN IKKE AABNET
ERROR 46 AT linienr. - FEJL I INPUT(KOMMAER)
ERROR 6Ø AT linienr. - CASE UDEN ENDCASES

Når der står, at en fejlmeddelelse også kommer ud under programudførslen (dette er tilfældet med alle AT linienr. sætningerne), er dette ikke det samme som at sige, at alle sætninger kommer ud kun som fejlmeddelelser i programmer. De kan lige så vel blive udskrevet uden for programmet.

I/O fejl (ud-indskriftsfejl på software)

I/O ERROR 18 - LINIEN ER FOR LANG
I/O ERROR 2o - PARIETETSFEJL

Oplysende tekster:

INTET PROGRAM

PROGRAMEKSEMPLER.

På denne, og den følgende side, er konstrueret nogle små program-eksempler, der, med undtagelse af Program til beregning af Areal Under Kurver, ikke er særligt komplicerede. Det nævnte program bygger på en matematik, som man først lærer lidt af i 1.G.s fysikundervisning, og som bliver yderligere underbygget i 2.G.s matematiktimer: Integralregning.

Programmer, såsom fakultetsprogrammer og største - mindste tal er nyttige i mange andre sammenhænge, og disse programmer kan ved nogle få og små ændringer nemt benyttes til underrutiner i andre programmer.

```
10  ;"PROGRAMMET BEREGNER (X,Y) I 2 LIGNINGER M. 2 UBEKENDTE."
20  ;"LIGNINGSSYSTEMET ER PAA FORMEN:"
30  ;" A1*X+B1*Y=C1"
40  ;" A2*X+B2*Y=C2"
50  ;
60  INPUT"INDTAST A1,A2,B1,B2: ",A1,A2,B1,B2
70  INPUT"INDTAST C1,C2: ",C1,C2
80  ;
90  D=A1*B2-A2*B1
100 ;"DETERMINANT D= ";D
110 IF D=0 DO
120   ;"OG LIGNINGSSYSTEMET KAN SAALEDES IKKE LOESES."
130 ELSE
140   X=C1*B2-C2*B1 ; Y=A1*C2-A2*C1
150   ;"X= ";X;"/";D;" ELLER ";X/D
160   ;"Y= ";Y;"/";D;" ELLER ";Y/D
170 ENDIF
180 ;
190 INPUT"SKAL FLERE LIGNINGER BEHANDLES, TAST DA 1, ELLERS 0: ",A
200 IF A THEN GOTO 30
210 ;"PROGRAMSLUT."
220 END
```

```
10  ;"PROGRAMMET KASTER 6 TERNINGER PAA EEN GANG, OG GOR"
20  ;"DETTE IALT 10 GANGE."
30  DEF FNR(X)=INT(RND(RND(RND(RND(X))))*6+1)
40  FOR I=1 TO 10
50    ;FNR(1);FNR(2);FNR(3);FNR(4);FNR(5);FNR(6)
60  NEXT I
70  END
```

```
10  ;"PROGRAMMET SLAAR 1 TERNING, OG SPOERGER DIG"
20  ;"DEREFTER, HVAD DU TROR, DEN HAR SLAAET."
30  T=INT(RND(RND(RND(0)))*6+1)
40  INPUT"HVAD GAETTER DU? ",G
50  IF G=T ;"JO, DU HAVDE RET!"
60  IF G<>T ;"HA HA, DER DUMMEDE DU DIG! <7>"
70  INPUT"TAST 1, HVIS DU VIL PROVE IGEN, ELLERS 0: ",G
80  IF G GOTO 30
90  ;"PROGRAMSLUT"
100 END
```

```
10  ;"PROGRAMMET FINDER HHV. STØRSTE OG MINDSTE TAL I EN"
20  ;"SERIE. DET SIDSTE TAL SKAL VÆRE  1E50."
30  ;
40  INPUT" 1.TAL:    ",TAL
50  LET  MIN=TAL;MAX=TAL;N=2
60  ;N;
70  INPUT" .TAL:    ",TAL
80  IF  TAL=1E50  GOTO  130
90  IF  TAL<=MIN  THEN  LET  MIN=TAL
100 IF  TAL>=MAX  THEN  LET  MAX=TAL
110 N=N+1
120 GOTO  60
130 ;"MINDSTE TAL  =  ";MIN
140 ;"STØRSTE TAL  =  ";MAX
150 ;"ANTAL TAL    =  ";N-1
160 END
```

```
10  ;"PROGRAMMET FINDER FAKULTETET AF ET GIVENT TAL."
20  ;
30  INPUT"INDTAST N:    ",N
40  IF  N<0  OR  INT(N)<>N  DO
50    ;"ERROR!<7>"
60  ELSE
70    F=1
80    FOR  I=1  TO  N
90      F=F*I
100   NEXT  I
110   ;"FAKULTETET PÅ  N =  ";F
120 ENDIF
130 END
```

```
10  ;"PROGRAMMET FINDER AREAL UNDER KURVER, FASTLAGT VED"
20  ;"FUNKTIONEN FNF(X)."
30  ;"EFTER STOP DEFFINERES FUNKTIONEN SÅLEDES:"
40  ;"0060  DEF  FNF(X)=...(UDTRYK)."
50  STOP  TAST  CON  EFTER  DEFFINITIONEN.
60  DEF  FNF(X)=SIN(2*X)+3
70  ;
80  INPUT"AREAL ØNSKES FUNDET I INTERVALLET FRA:",A,"  TIL:  ",B
90  INPUT"HVOR MANGE DELINTERVALLER? (>100):",N
100 FOR  I=1  TO  2*N
110   S=S+ABS(FNF(A)+ABS(A-B)*I/(2*N))
120 NEXT  I
130 ;"AREALET UNDER KURVEN I INTERVALLET FRA";A;"TIL";B
140 ;"  =  ";S*ABS(A-B)/(2*N)
150 ;
160 GOTO  30
```

Kopiering af hulstrimler

Dette skal man have tilladelse fra en lærer, før man går i gang!

Filosofien bag dette er, at har man en datamat, kan man lige så godt benytte dens store lagerplads til "buffer"lager (midlertidigt) under kopiering, således at man eventuelt kan sammensætte flere strimler. Endvidere er dette også den eneste mulighed, hvis man skal have kopieret de såkaldte AB-strimler (Absolut Binær) eller Library-strimler. Det følgende giver en vejledning i kopieringsproceduren:

1. Nøgle på ON, alle adressekontakter ned, undtagen 12 og 14.
2. Binær Loader lægges i læser, og der tastes RESET, xxxxx på læseren i nævnte rækkefølge.
3. Kontakterne RESET, PROGRAM LOAD på datamaten aktiveres i rækkefgl.
4. Binær Loader indlæses nu, og adressekontakterne sættes derefter alle opad.
5. Kopieringsstrimlen lægges i læseren, og der tastes RESET på læser.
6. Herefter tastes på datamat RESET, START i nævnte rækkefølge, og kopieringsstrimlen læses nu.
7. På skærmen kommer nu teksten
   LOAD PTR - TYPE ANY KEY
   hvorefter datamaten venter. Brugeren skal nu påse, at der er tændt for hulstrimmelpuntcheren (se under Behandling Af Programmel Og Maskiner), samt efterse, om der er strimmel i maskinen.
8. Herefter lægges brugerens original i læseren, og man taster ligegyldigt hvilken tast, man vil, på tastaturet. Herefter starter kopieringen, idet datamaten både læser strimlen og udskriver samme indhold på hulstrimmelskriveren. Det spiller ingen rolle hvilket indhold, strimlen har, da dette ikke kontrolleres af datamaten. Man kan således kopiere en hulstrimmel baglæns, hvis det har interesse.
9. Er hulstrimlen lang, kan det ske, at læseren bliver før færdig end skriveren. Dette er ganske normalt, og kopieringen fortsætter stadig.
10. VIGTIGT: Læg ikke flere strimler i læseren, førend en kopiering er afsluttet, da datamaten ellers automatisk vil indlæse den nye strimmel, og på den måde lave en sammenhængende kopiering af to, måske fuldstændig forskellige strimler.
11. VIGTIGT: Indholdet af en strimmel forsvinder i samme øjeblik en kopiering er til ende. Skal der laves flere kopier af samme strimmel, må man altså indlæse den igen, som vist under 7-8.
12. Fortsat kopiering, eller sammensætning af strimler (IKKE AB - eller Librarystrimler): Første strimmels udføringsløb afrives, ligesom den næstfølgendes strimmels indløbsstrimmel også afrives. Disse kopieres nu efter 7-8 umiddelbart efter hinanden. Herefter vil de to - eller flere - strimler hænge sammen.
13. Efter kopieringsproceduren - dvs. efter 8 - stopper skriveren, og teksten LOAD PTR - TYPE ANY KEY kommer igen på skærmen. Datamaten er herefter klar til en ny kopiering.
14. Efter alle kopieringer er foretaget, skal maskinen lades op igen. Se under OPSTART.

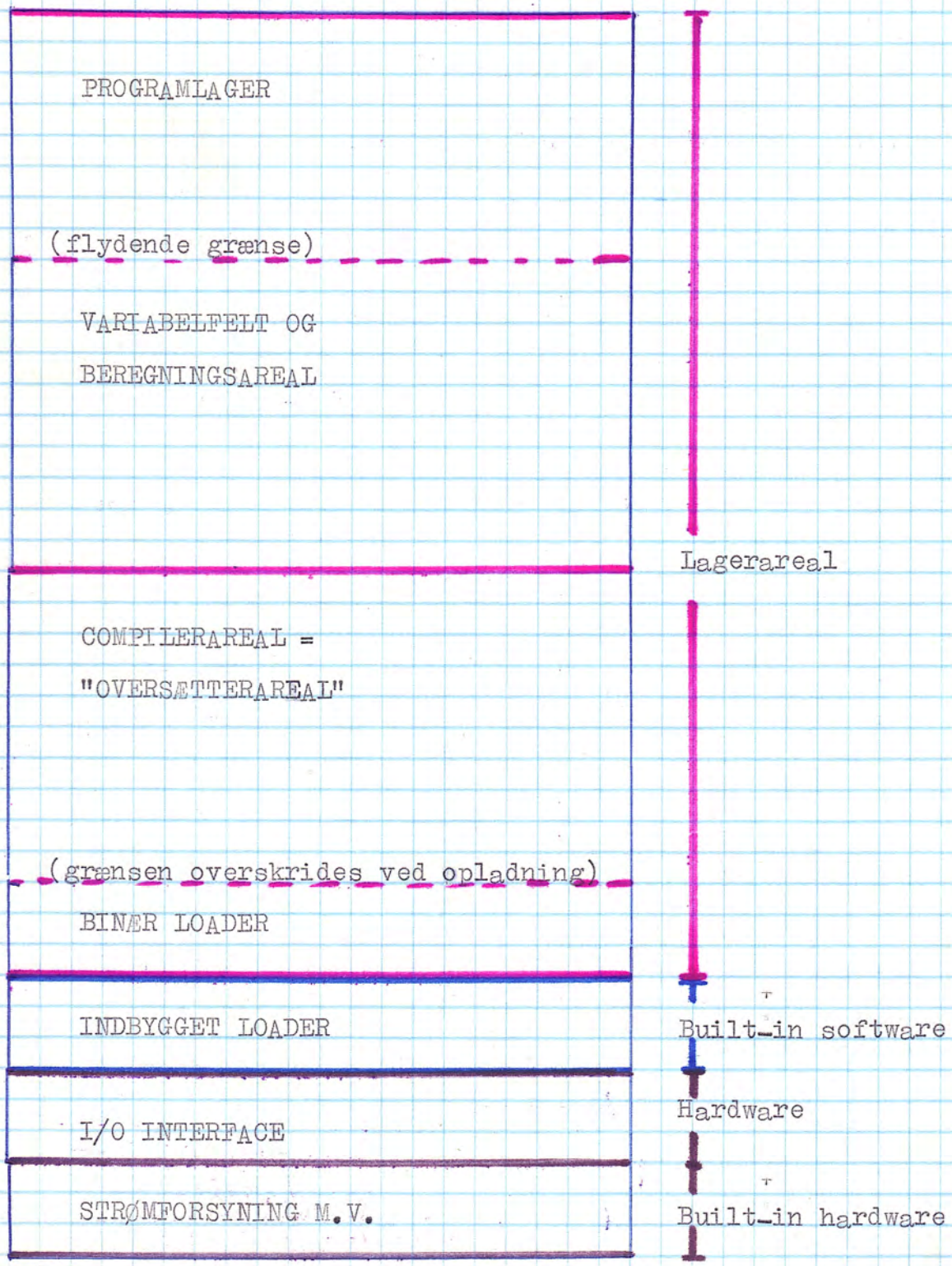
Grunden til, at man bør have tilladelse fra en lærer er, at kopiering kan bruge meget papir - hvis man evt. pjatter...

Bemærk: Taster man efter teksten LOAD PTR etc. på en eller anden tast, uden at der er strimmel i læseren, vil strimlen på hulstrimmelskriveren kun blive ført et lille stykke frem, og teksten vil blive udskrevet igen - man har kopieret en strimmel, der ikke var der...

Lidt om datamatens opbygning

Det følgende giver sig aldeles ikke ud for at være skrevet i lærebogsform, og jeg har heller ikke søgt at komme med svære fagudtryk med det samme. Grunden til, at jeg har medtaget dette, er at mange fejl lettere vil kunne forstås, hvis man blot så nogenlunde er klar over, hvordan datamaten er opbygget. Jeg vil ikke komme med indviklede strømdiagrammer, som hverken jeg, eller læseren, er klar over hvordan de fungere. Til gengæld prøver jeg med nogle få tegninger, at forklare princippet i den type maskine, som en datamat er. Jeg vil her henvise til speciallitteratur, hvis man er yderligere interesseret.

Det første, jeg vil fremdrage, er hvordan datamaten kunne tænkes opbygget af byggeklodser. Og det har jeg søgt at vise på tegningen herunder: (principtegningen kunne gælde alle datamater af digital-typen)



Som man kan se på tegningen er det alligevel lykkedes mig at rode mig ind i nogle fremmedord fra begyndelsen. Disse vil jeg gå igennem her, da de er temmelig vigtige for den videre forståelse:

COMPILER kan oversættes ved "oversætter", men dette er ikke særlig godt, da compileren ikke alene oversætter, men også sammenholder sætningens ord med en indbygget liste over tilladte ord i det givne sprog, lige som den også undersøger, om disse ord står rigtigt eller ej. Er dette tilfældet, sker oversættelsen til det, der kaldes for absolut binært, dvs. til koder, bestående af ordrene strøm/ikke strøm. Det er disse koder, der opbevares i programlageret og i variabelfeltet.

BINÆR LOADER er en række maskininstruktioner, der får datamaten til at indlæse hulstrimler med andre, og mere komplicerede, instruktioner. Her i referencebogen er der af de sidstnævnte omtalt: BATCH, RC-BASIC, COMAL II og kopieringsstrimlen, men mange andre ting kan indlæses ved BINÆR LOADER.

SOFTWARE er de ting, der kan lægges på strimmel, vises på en skærm, eller læses fra strimler, hulkort eller magnetbånd. Således er BINÆR LOADER også software, da den ligger på strimmel.

HARDWARE er maskiner og maskinelt materiel, dvs. datamat, puntcher, læser, linieskriver, hulkortlæser og opruller i vores tilfælde.

INTERFACE, eller som det fulde navn er, I/O INTERFACE DEVISE, er det tekniske sammenkoblingsudstyr til/fra datamat til det øvrige input/outputudstyr i hardwaregruppen.

Forklaring af tegning: Vi begynder nederst for en gangs skyld. Her finder vi alt det mere tekniske, nemlig strømforsyning, blæser, osv. Hertil vil jeg ikke sige så meget, ikke andet, end at maskinen kører på 5 V jævnspænding, og har lo A sikringer - dette for de mere teknisk orienterede. Videre i det tekniske kammer vi til den klods, der hedder I/O INTERFACE, som styrer det rent tekniske input/outputudstyr. Dette styres fra programmet, eller snarere fra det, der kaldes kontrolenheden (Controle Unit = CU). Går vi nu videre kommer vi til det indbyggede software, nemlig den indbyggede loader. Denne loader ligger direkte loddet ned på printplader, og kan således ikke ændres lige med det samme. I datamaten er den ret så vigtig, da det er denne enhed, der gør det muligt for datamaten at indlæse to tegn - de to første - fra en BINÆR LOADER. Den indbyggede loader startes ved kommandoen RESET, PROGRAM LOAD.
Herefter forlader vi den mere maskinelle del, og går til lagerarealet - det er lidt dumt at tale om arealer i en datamat, som er rumlig, og som gemmer informationer inde i ferritkerner på udeffinerbare steder, men vi fortsætter med abstraktionerne. Det første, vi møder her, er den omtalte BINÆR LOADER, der i de to første tegn har komandoen: Indlæs det følgende, og udfør det som ordre. Se i øvrigt under BINÆR LOADER i den ovenstående ordforklaring. Herefter kommer COMPILERAREALET - idet vi antager, at det er under normal kørsel, hvor man ikke ødelægger dette eller hint. Dette areal - se under COMPILER - er tvangsflydende, således at det ved indlæsning, bliver læst ovenfi den binære loader, og således fjerner den. Compileren reserverer herefter nu et areal til brug for det følgende: Variabelfelt og beregningsareal, samt programlager. Efter en compilering skal det færdige program lægges et eller andet sted hen, og det gøres i programlageret. Dette sker samtidig med, at der reserveres plads i et variabelfelt til alle de variable, der forekommer i programmet. Under kørsel af programmet forekommer der tit nogle beregninger, og disse lægges ned i et areal, der beregner alle disse udtryk. At der mellem programlageret og variabelfeltet er en flydende grænse, skal forstås således, at hvis programmet er stort, kan der ikke bruges så mange variable, og omvendt, er der mange variable (fx. en matrix med 75 x 75 pladser), er der ikke så meget plads til programmer.

Det er muligt at få datamaten til at blokere ved ordre ved tastaturet - og det oven i købet ved ordrer, som maskinen skulle producere en fejltekst til. Grunden til, at maskinen vil forsøge at udføre disse ordre, som den ikke kan, og derfor kommer "i konflikt" med sig selv - blokerer - ligger i, at under compilationen udfører maskinen ikke programmet, således at sætninger, der helt åbentlyst er forkerte, nemt kan kommes i maskinens lager. Således er det for eksempel med følgende ordre (bør skrives uden for

program, og medens der sidder et program i datamaten):
;"((
(((((",,,,,"((((((((((" car return

Maskinen vil skrive de første mange tegn, der næsten bruger al plads, der er i en enkelt linie, og derefter søge at flytte blinkeren de 5 x 14 pladser, som kommaerne angiver, for derefter at søge at skrive de sidste tegn. Men dette er umuligt, for derved bliver maksimalstørrelsen af mulige positioner i én linie - nemlig 12o positioner - overskredet. På den anden side er der intet forkert i at skrive en sådan linie, rent syntaksmæssigt, og maskinen skal udføre denne ordre. Maskinen skal gøre noget, den ikke kan gøre. Prøv herefter at LIST'e programmet, der stod i maskinen i forvejen - det vil antageligt se noget besynderligt ud. Mere besynderligt bliver det, hvis man søger at RUN'ne programmet. Prøv at se på datamatens lyspanel, når car return tastes efter ordet RUN... Efter den tur behøver maskinen en opladning, se da under OPSTART.

==========

Muligheder

Men hvad kan man så bruge en datamaskine til - og hvad kan man bruge især den datamaskine, som findes på VG, til? Dette er nok det spørgsmål, der rejser sig, når man har prøvet at køre et eller to simple programmer på maskinen - og det er nok det, der gør, at mange forlader maskinen efter et par gange. Man er klar over, at datamaskinen har mange muligheder, men man er ikke klar over hvor man skal begynde, eller hvad datamaten rent faktisk kan gøre for én. Mange vil også føle sig på gyngende grund efter de - desværre - alt for få og alt for koncentrerede studietimer, som der gives i programmeringsteknik. For der gives ingen vejledning i, hvad man også kan klare med sin nyerhvervede kunnen.
En datamaskine er en maskine, og som sådan yderst logisk. Den er fabelagtig til hurtig talbehandling, og kan benyttes til at løse komplicerede ligningssystemer - hvis der findes en slavisk metode til at gå frem efter. For også det skal man lægge mærke til: Et menneske kan ved at gå frem efter sin fantasi løse mange problemer, men en maskine har ingen fantasi, og har derfor en hel del begrænsninger.
Til tider kan man ved brug af iterative programmer - programmer, der ved brug af spørgende sætninger hele tiden ændrer på værdier i programmet (se f.eks. side 38) - få beregnet svaret på problemer, som man normalt skulle side i timevis for at få beregnet, men der findes også tilfælde, hvor det er ti gange hurtigere at regne tingene ud "i hovedet". Man må altså ikke stille for store krav til datamaten. Men til gengæld er en datamaskine en meget hurtig regnemaskine, en maskine, der har en virkelig stor hukommelse, og en maskine, der kan sammenligne to tal meget hurtigt og følge en efterfølgende ordre. En datamaskine er ligeledes i stand til at lave tekstbehandling, som f.eks. at redigere sider, lave sproganalyser ved at optælle vokaler og konsonanter, kommaer osv. i en given tekst - dette er altså igen slavearbejde.
I vores tilfælde kan vi svinge os helt op til tekstanalyse, men normalt benyttes RC 7ooo til to ting: Til at udføre beregninger, som en almindelig regnemaskine enten ikke vil kunne klare, eller i bedste fald ville køre for langsomt til at være økonomisk nok - eller til at lege med. Og det er nok i høj grad det sidste, man bruger maskinen til på VG. I tidens løb er der konstrueret mange morsomme programmer - programmer, der spiller skak (ikke et helt parti, der var kun et tårn og to konger på brædtet), programmer, der løser tipsopgaver, programmer, der leger bank (Vestfyns Gymnasiums Bank Inc.) - programmer til brug i samfundsfagstimerne, hvor de løser arbejdsløsheden, valutaunderskudet osv. - programmer til brug i fysik, der simulerer radioaktive stoffer - programmer til matematik, der beregner integraler - programmer til at kaste terninger, til at tegne kurver, til at lægge to tal sammen med, til at finde primtal osv, osv, osv.
Der er faktisk en hel masse muligheder - og den bedste måde, man kan finde ud af dem på, er ved at sætte sig ned foran skærmen i et frikvarter, en fritime eller i fritiden, og lege. Ved at følge de regler, der er angivet i bogen, vil man hurtigt kunne finde frem til et eller andet.

GOD FORNØJELSE!!