RCSL No: 42-i1578

Edition: December 1980.

Author: Erik Jeppesen
Stig Møllgaard

Title:

RC700 COMAL
Brugermanual

REGNECENTRALEN
af 1979

Keywords:
RC700 mikrodatamat, COMAL, ID-COMAL, Brugermanual.

Abstract:

This manual describes (in Danish) the use of the COMAL-system implemented on the RC700 microcomputer.

(printed pages 64)

Printed by A/S Regnecentralen af 1979, Copenhagen

424 1341

FORORD

Første udgave: RCSL: 42-i1339.
Systemet bygger på ID-COMAL, som er udviklet på instituttet for Datateknik, Danmarks Tekniske Højskole uder ledelse af Tom Østerby. Knud Henningsen og Erik Jeppesen har for Regnecentralen foretaget ændringer i COMAL og det underliggende system for dels at tilpasse det til RC700-mikrodatamaten og dels at tilnærme ID-COMAL til RC BASIC/COMAL, som kendes fra RC7000 - minidatamaten.

Indeværende manual er skrevet med udgangspunkt i Tom Østerby's beskrivelse af ID-COMAL, Dok.nr: ID-788, 1978-9-20.

Anden udgave: RCSL: 42-i1578.
Denne 2. udgave af manualen beskriver dels nye faciliteter, som er indført i COMAL og dels ændringer som følge af den nye udgave, RC702, af mikrodatamaten med tilhørende nyt tastatur. Ændringer i forhold til første udgave er markeret med en lodret streg i venstre margin.

Erik Jeppesen & Stig Møllgaard
A/S Regnecentralen af 1979, december 1980.

INDHOLDSFORTEGNELSE PAGE

INDHOLDSFORTEGNELSE (fortsat) PAGE

INDHOLDSFORTEGNELSE (fortsat) PAGE

1. INDLEDNING 1.

Denne manual beskriver den version af programmeringsproget COMAL, som er implementeret på Regnecentralen af 1979's mikrodatamatsystem, RC700. Manualen beskriver kun anvendelsen af COMAL. Med hensyn til installation og betjening af mikrodatamaten henvises til "RC700 Brugervejledning".

I manualen er COMAL-sprogets syntaks beskrevet på en slags Backus-Naur Form.
Denne beskrivelsesmetode kan bedst forklares ved et eksempel:

```
1: <strengudtryk>   ::= <strengoperand> ! <strengudtryk> + <strengoperand>

2: <strengoperand>  ::= <strengvariabel> ! <strengkonstant> !
                        CHR'(' <aritmetisk udtryk>')'

3: <strengkonstant> ::= "<vilkårlig ASCII-streng undtagen
                         CR, LF, og '"'>"
```

Symbolet '::=' kan læses som 'består af'.
Tegnet ' ! ' kan læses som 'eller'.

Linie 1 siger så, at strengudtryk består af strengoperand eller strengudtryk + strengoperand.
Linie 2 siger, at strengoperand består af strengvariabel eller strengkonstant eller CHR(aritmetiske udtryk).
Linie 3 siger, at strengkonstant består af en følge af ASCII-tegn omgivet af anførelsestegn (").

Generelt kan det siges, at
- store bogstaver angiver (nøgle-)ord, som brugeren skal skrive.
- små bogstaver omgivet af < og > angiver begreber, som er eller vil blive defineret, dvs. at <...> optræder én gang på venstresiden af symbolet '::='.
- paranteser angiver, at elementerne i paranteserne kan angives eller udelades. Hvis brugeren skal skrive et af tegnene (eller), er dette angivet ved '(' og ')'.

2. INDTASTNING 2.

Når der tændes for datamaten, og en systemdiskette indsættes, vil teksten 'RC700 COMAL REV. X.XX' normalt komme frem på skærmen. Sker dette ikke, se da brugervejledningen. Herefter udskrives en '*' som tegn på at systemet er klar til indtastning.
Der kan nu indtastes program-sætninger eller kommandoer.

2.1 Programsætninger 2.1

Programsætninger indledes af et linienummer, der er bestemmende for sætningens placering i programmet. Et linienummer er et heltal mellem 1 og 9999. Sætningerne kan indtastes i vilkårlig orden, idet systemet selv ordner dem efter stigende linienumre.
Indtastes en sætning med samme linienummer som en allerede eksisterende, vil den nye sætning erstatte den eksisterende.
Når en sætning er indtastet, undersøges det, om det er en korrekt COMAL-sætning (syntaksanalyse). Er dette tilfældet, lagres den i programmet i overenstemmelse med sit linienummer.
Er den ikke syntaktisk korrekt, udskrives den igen med cursor'en placeret på det sted, hvor fejlen blev detekteret. Herefter kan sætningen rettes og sendes til syntaksanalyse igen.
COMAL-sætningerne er beskrevet i kapitel 4.

2.2 Kommandoer 2.2

En kommando adskiller sig fra en programsætning ved ikke at starte med et linienummer.
Kommandoer udføres øjeblikkeligt og anvendes til listning, afvikling og indkøring af programmer samt til håndtering af filer på disketten.
Kommandoerne er beskrevet i kapitel 8.

2.3 Editering

2.3

Indtastning sker på skærmens nederste linie og afsluttes ved at trykke på tasten mærket '↵' ('RETURN').
Under indtastningen har brugeren forskellige editeringsfaciliteter til rådighed. Til de fleste af disse faciliteter er knyttet en speciel tast på tastaturet, mens enkelte faciliteter aktiveres ved hjælp af kontroltegn, som fremkommer ved, at der trykkes på tasten mærket 'CTRL' og en anden tast samtidigt. F.eks. betyder 'CTRL H': tryk samtidig på tasterne 'CTRL' og 'H'.

De specielle editeringstaster kontroltegn har følgende betydning i COMAL:

- '↵' (RETURN): afslutning på indtastning.
- '↓' (LINE FEED), = 'CTRL J'): som RETURN.
- '←' (CURSOR LEFT, = 'CTRL H'): Cursor'en flyttes en plads til venstre, uden at linien ændres.
- '→' (CURSOR RIGHT, = 'CTRL X'): Cursor'en flyttes en plads til højre, uden at linien ændres.
- '→|' (INSERT CHARACTER, = 'CTRL I'): alle tegn fra og med cursorens placering flyttes en plads til højre, og et blank tegn fremkommer på cursorens position.
- '|←' (DELETE CHARACTER, = 'CTRL E'): alle tegn fra cursorens placering flyttes en plads til venstre, og det oprindelige tegn på cursoren's position fjernes.
- 'CTRL K' (DELETE LINE): alle tegn placeret på cursoren's position og til højre for denne fjernes.
- 'ESC' (ESCAPE): den netop indtastede linie fjernes, én '*' udskrives, og en ny linie kan indtastes.

Ovennævnte taster kan bruges dels under indtastningen af programmer og dels, når systemets editor, som er beskrevet i kapitel 8, benyttes.

3. COMAL PROGRAMMER

3.

Et COMAL program består af et antal sætninger. Hver programlinie indledes af et linienummer, der efterfølges af et COMAL-nøgleord, som identificerer sætningstypen. Efter nøgleordet kan følge et antal argumenter, bestående af følgende grundelementer:

- variable
- nøgleord
- aritmetiske udtryk
- logiske udtryk

Aritmetiske og logiske udtryk bygges op af:

- talkonstanter
- identifikatorer
- operatorer

Disse grundlæggende sprog-elementer beskrives i det følgende.

3.1 Tal

3.1

Tal benyttes som operander i aritmetiske udtryk, samt i inddata. Reelle tal kan udtrykkes enten som heltal, decimaltal eller på eksponentiel form.
I den eksponentielle notation betyder 'E': 'gange 10 opløftet til potensen'.

Eksempler:

Heltal	decimaltal	eksponentiel form
13	13.	
150000		15E4
	12.5	
	0.25	2.5E-1
125		.125E+3
	.33	330E-3

3.2 Identifikatorer

Identifikatorer benyttes til at navngive forskellige størrelser i et COMAL-program. En identifikator består af op til 16 bogstaver og cifre og skal begynde med et bogstav. Blanktegn må ikke indgå i en identifikator, og den skal efterfølges af et tegn, som ikke er et bogstav eller et ciffer.

Eksempler:

```
I
X1
COMAL
H20
```

Identifikatorer kan skrives med små eller store bogstaver, men alle bogstaver omsættes til store bogstaver.

Identifikatorer kan betegne følgende størrelser i et program:

- simple reelle variable
- indicerede reelle variable
- simple strengvariable
- indicerede strengvariable
- brugerdefinerede funktioner
- procedurer
- formelle parametre
- filvariable

Samme identifikator kan ikke betegne forskellige størrelser i et program, og identifikatorer må ikke være de samme som de reserverede nøgleord.

3.3 Reserverede nøgleord

I COMAL findes en række reserverede nøgleord, der har en fast betydning, disse nøgleord må ikke benyttes i anden betydning.

De reserverede nøgleord er:

AND	CHAIN	CLOSE
CREATE	DATA	DEF
DELETE	DIM	ELSE
END	ENDWHILE	ENDIF
ENDLOOP	ENDPROC	ESC
EXEC	EXIT	FN
FOR	GET	GO
GOSUB	GOTO	IF
IN	INPUT	LET
LOOP	NEXT	NOT
ON	OPEN	OR
OUT	OUTPUT	PRINT
PROC	PUT	READ
REM	REPEAT	RESTORE
RETURN	STEP	STOP
THEN	TO	UNTIL
USING	WHILE	

Nøgleord kan skrives med store eller små bogstaver, men ved udskrift vil de blive konverteret til små bogstaver. Et nøgleord skal altid efterfølges af mindst et blanktegn.

3.4 Reelle variable

I COMAL findes to typer reelle variable, nemlig simple variable og talsæt.

Reelle variable kan tildeles reelle værdier og disse værdier lagres i 4 oktetter - 1 til eksponent og 3 til taldel.

Absolut - værdi af reelle værdier ligger i området

2.9E - 39 < x < 1.7E38

Hvis resultatet af en beregning bliver større end 1.7E38, sker der overløb, og programmet standses med en fejlmeddelelse. Sker der underløb (resultatet mindre end 2.9E - 39), sættes resultatet til nul, og beregningen fortsættes.

En simpel variabel refereres ved dens tilhørende identifikator. En simpel variabel skal ikke erklæres. Erklæringen foregår første gang den findes på venstre side af et '=' i en tildelingssætning (LET-sætning) eller i variabellisten i en INPUT- eller READ-sætning.

Et talsæt består af en samling indicerede variable organiseret i form af en vektor eller en matrix. En indiceret variabel har følgende opbygning:

```
<indicereret variabel> ::= <identifikator> '(' <index> ')'!
                           <identifikator> '(' <index> , <index> ')'
<index>                ::= <aritmetisk udtryk>
```

hvor 'identifikator' betegner navnet på talsættet. Dette skal være forskelligt fra navnene på de systemdefinerede funktioner.

Et talsæt skal erklæres, før det anvendes. Ved en sådan erklæring skal der angives dimension, og øvre indeks for hver enkel dimension. Nedre indeks er fast lig 1. Erklæringen foregår i en DIM-sætning. Ved udførelse af denne sætning afsættes plads til talsættet, og de indicerede variable initialiseres til nul.

Ved alle referencer til talsæt foretages indekscheck, dvs. det undersøges, om antallet af indices og deres værdier er tilladelige.

3.5 Strengvariable

3.5

COMAL er i stand til at arbejde med strenge bl.a ved hjælp af strengvariable og strengkonstanter.

En strengvariabel kan tildeles værdi i form af en streng af ASCII-tegn.

COMAL har to typer af strengvariable, nemlig simple strengvariable og vektorer af strengvariable (indicerede strengvariable). En strengvariabel har følgende opbygning:

```
<strengvariabel>          ::= <simpel strengvariabel>!
                              <indiceret strengvariabel>

<simpel strengvariabel>   ::= <strengidentifikator>!
                              <strengidentifikator> ( <selektor>)

<indiceret strengvariabel> ::= <strengidentifikator> '(' <index> ')'!
                               <strengidentifikator>
                                      '(' <index> , <selektor> ')'

<strengidentifikator>     ::= <identifikator>$

<selektor>                ::= <startposition> : <længde>

<startposition>           ::= <aritmetisk udtryk>

<længde>                  ::= <aritmetisk udtryk>
```

Strengvariable adskilles fra reelle variable ved, at der efter identifikatoren skal følge '$'.

Når en simpel strengvariabel kun angives ved den tilhørende strengidentifikator, betyder dette hele strengen af ASCII-tegn. Er der angivet en 'selektor', betyder dette en delstreng startende med 'startposition' og med et antal tegn lig 'længde'. Første tegn i en strengvariabel refereres med position = 1.

For en indiceret strengvariabel skal der angives et indeks, der angiver det ønskede element i vektoren. Herudover kan der eventuelt være en angivelse af selektor. Betydningen af denne er som under simple strengvariable.

Strenge og vektorer af strenge skal erklæres i en DIM-sætning. Ved udførelse af denne sætning afsættes lagerplads til strengvariablen, og den initialiseres med tegnværdier, hvis betydning er uinitialiseret (NULL). Strengens maksimale længde erklæres derfor i DIM-sætningen. Strengens aktuelle længde er derimod det antal tegn startende fra position 1 indtil det første tegn lig 'NULL'. Ved erklæring af en streng vil den aktuelle længde derfor være 0.

Ved programudførelsen foretages indekscheck, dvs. at indeks til strengvektor, startposition og længde er tilladelige.

3.6 Aritmetiske udtryk

3.6

Aritmetiske udtryk indgår i en række forskellige sætninger.

Aritmetiske udtryk har følgende opbygning:

```
<aritmetisk udtryk> ::= ( + ! - ) <a-udtryk>

<a-udtryk>          ::= <led> ! <a-udtryk> + <led> ! <a-udtryk> - <led>

<led>               ::= <faktor> ! <led>*<faktor> ! <led>/<faktor> !
                        <led> MOD <faktor> ! <led> DIV <faktor>

<faktor>            ::= <operand> ! <faktor>    <operand>

<operand>           ::= '(' <aritmetisk udtryk> ')' ! <reelt tal> !
                        <reel variabel> ! <systemfunktion> !
                        <brugerfunktion> ! (logisk udtryk) !
                        <filvariabel> ! <formel parameter>

<brugerfunktion>    ::= (FN) <identifikator>'('<aritmetisk udtryk')
```


Systemfunktioner er beskrevet i kapitel 6.

En brugerfunktion anvendt som operand skal være erklæret - i en DEF-sætning - inden den benyttes. Nøgleordet 'FN' kan udelades ved kald af en brugerdefineret funktion.

Reelle variable der benyttes som operander i aritmetiske udtryk skal være oprettede og have fået tildelt værdier - i LET-, READ- eller INPUT-sætninger.

Operatoren ↑ er potensopløftning, 'MOD' er modulus, 'DIV' er heltalsdivison.

3.6.1

3.6.1 Operatorers prioritet

Aritmetiske udtryk beregnes normalt fra venstre mod højre. Rækkefølgen kan dog ændres med paranteser og som følge af, at operatorerne har forskellig prioritet.
Følgende regler gælder:

1. Udtryk i parantes beregnes først. Er der paranteser i flere niveauer beregnes det inderste udtryk først.

2. Derefter udregnes funktioner.

3. De aritmetiske operatorers prioriteter er:

   Første: monadisk plus og monadisk minus
   Anden: potensopløftning
   Tredie: multiplikation, division, modulus regning
   Fjerde: addition og subtraktion

4. Har to operatorer samme prioritet beregnes de fra venste mod højre.

3.7 Strengudtryk

Strengudtryk benyttes ved tildeling af værdier til strengvariable (i LET-sætning), ved udskrivning (i PRINT-sætning) og i relationer.

Strengudtryk har følgende opbygning:

<strengudtryk> ::= <strengoperand> ! <strengudtryk> + <strengoperand>

<strengoperand> ::= <strengvariabel> ! <strengkonstant> !
CHR '(' <aritmetisk udtryk> ')'

<strengkonstant> ::= "<vilkårlig ASCII-streng undtagen CR, LF og ' " '>"

'+' er en strengoperator, der angiver konkatenering (sammenkædning).

Eksempel

Er A$= "RC" og B$= "700", da har A$+B$ værdien "RC700".

Tegnet ' " ' samt kontroltegn (tegn med ASCII-værdi <32) til f.eks cursor-styring kan indsættes i tegnstrenge ved hjælp af funktionen CHR (se afsnit 6.3).

3.8 Logiske udtryk

Logiske udtryk benyttes til at gøre sætningsudførelsen betinget. Logiske udtryk indgår i IF/THEN - sætning, WHILE - sætning og UNTIL - sætning.

Et logisk udtryk har følgende opbygning:

<logisk udtryk> ::= (NOT) <l-udtryk>

<l-udtryk> ::= <l-led> ! <l-udtryk> OR <l-led>

<l-led> ::= <l-operand> ! <l-led> AND <l-operand>

<l-operand> ::= <relation> ! '('<logisk udtryk>')'

<relation> ::= <strengrelation> ! <aritmetisk relation>

<strengrelation> ::= <strengudtryk> <reloper> <strengudtryk>

<aritmetisk relation> ::= <aritmetisk udtryk> <reloper>
<aritmetisk udtryk>

<reloper> ::= < ! <= ! =< ! > ! >= ! => ! <> ! =

De logiske operatorer er defineret ved hjælp af følgende sandhedstabeller:

NOT:	A	NOT A
	FALSK	SAND
	SAND	FALSK

OR:	A	B	A OR B
	FALSK	FALSK	FALSK
	FALSK	SAND	SAND
	SAND	FALSK	SAND
	SAND	SAND	SAND

AND:	A	B	A OR B
	FALSK	FALSK	FALSK
	FALSK	SAND	FALSK
	SAND	FALSK	FALSK
	SAND	SAND	SAND

Betydningen af relationsoperatorerne er:

> : større end
>= : større end eller lig med
= : lig med
<> : ikke lig med
<= : mindre end eller lig med
< : mindre end

Hvis relationen mellem to udtryk er opfyldt, er værdien sand, ellers er den falsk.
I en strengrelation sammemlignes strengudtrykkene tegn for tegn fra venstre mod højre. Sammenligningen sker på grundlag af tegnenes ASCII-værdier (se App. A).
Hvis to strenge af forskellig længde sammenlignes, og de første tegn i den længste streng er lig med tegnene i den korteste, da vil den korteste streng være mindst. F.eks. er følgende relation sand:

"ABC" < "ABCD".

Hvis logiske udtryk indgår i aritmetiske udtryk, da vil værdien 'sand' svare til den aritmetiske værdi 1, og 'falsk' til 0.

Prioriteten af de logiske operatorer er:

Første: NOT
Anden : AND
Tredie: OR

Rækkefølgen af beregningerne kan ændres ved hjælp af paranteser.

4. COMAL SÆTNINGER 4.

4.1 REM - sætning 4.1

REM - sætningen benyttes til at kommentere et program.
REM - sætningen har følgende opbygning:
<rem - sætning> ::= REM <kommentar>
En REM - sætning har ingen virkning ved programudførelsen.

4.2 LET - sætning 4.2

LET - sætninger benyttes til at tildele værdier til reelle variable og strengvariable.

En LET - sætning har følgende opbygning:

<let-sætning> ::= (LET) <tildelingsliste>

<tildelingsliste> ::= <tildeling> |
<tildelingsliste> ; <tildeling>

<tildeling> ::= <aritmetisk tildeling> | <streng tildeling>

<aritmetisk tildeling ::= <variabelliste> = <aritmetisk udtryk>

<variabelliste> ::= <reel variabel> |
<variabelliste>, <reel variabel>

<streng tildeling> ::= <strengvariabel> = <streng udtryk>

Nøgleordet, LET, kan udelades.

I den aritmetiske tildelingssætning vil alle variable på venstre side af '=' få tildelt værdien af det aritmetiske udtryk.

Såfremt 'streng udtryk' er længere end 'strengvariabel' vil 'streng udtryk' blive afkortet, idet kun den første del vil blive benyttet. I 'streng tildeling' må 'strengvariable' gerne indgå i 'streng udtryk'. Følgende tildeling er ikke alene tilladelig, men også hensigtsmæssig:

```
LET string$ = string$ + "er"
```

der har den virkning at 'er' bliver placeret efter de tegn, der tidligere er blevet placeret i string$. Indeholder 'strengvariabel' ikke nogen selektor, vil 'strengvariabel' efter tildelingen indeholde 'streng udtryk' efterfulgt af null-tegn. Indeholder 'strengvariabel' en selektor tildeles kun de tegn, der er omfattet af selektor.

4.3 DIM - sætning 4.3

DIM - sætning benyttes til at erklære reelle talsæt (vektorer og matricer) samt strenge og vektorer af strenge.

DIM - sætningen har følgende opbygning:

```
<dim-sætning>        ::= DIM <erklæringsliste>

<erklæringsliste>    ::= <erklæring> !
                         <erklæringsliste> , <erklæring>

<erklæring>          ::= <talsæterklæring>! <strengerklæring>

<talsæterklæring>    ::= <identifikator> '('<aritmetisk udtryk>')' !
                         <identifikator>
                         '('<aritmetisk udtryk> , <aritmetisk udtryk>')'

<strengerklæring>    ::= <identifikator> $ '('<aritmetisk udtryk>')' !
                         <identifiaktor> $
                         '('<aritmetisk udtryk> , <aritmetisk udtryk>')'
```


Identifikatoren for et reelt talsæt skal være forskellig fra alle identifikatorer for simple reelle variable og bruger- og systemdefinede funktioner.

Identifikatoren for en strengvariabel skal være forskellig fra alle identifikatorer for simple reelle variable og brugerdefinerede funktioner.

Ved udførelsen af DIM - sætningen afsættes plads i dataområdet til de størrelser, der erklæres. Maksimumsstørrelsen for en indiceret variabel eller en strengvariabel er kun begrænset af arbejdslagerets størrelse.

4.4 DEF - sætning 4.4

En DEF - sætning benyttes til at erklære betydningen af en brugerdefineret funktion.

DEF - sætningen har følgende opbygning:

```
<def-sætning>        ::= DEF (FN) <funktionsnavn>'('<formel parameter>')'
                                                  =<aritmetisk udtryk>
<funktionsnavn>      ::= <identifikator>

<formel parameter>   ::= <identifiaktor>
```

Nøgleordet 'FN' kan udelades. Identifikatoren for funktionenen skal være forskellig fra alle identifikatorerne for alle andre størrelser. En brugerdefineret funktion kan kun have én parameter.

En brugerdefineret funktion skal erklæres, før den benyttes. Det er tilladt at kalde andre brugerdefinerede funktioner i det aritmetiske udtryk, der definerer funktionen. Rekursive kald er ikke tilladte.

Ved kald af en brugerdefineret funktion vil 'formel parameter' få tildelt værdien af aktuel parameter, hvorefter det aritmetiske udtryk udregnes. Værdien af dette udtryk indsættes på brugerfunktionens plads.

4.5 PRINT - sætning 4.5

PRINT - sætningen benyttes til udskrivning af resultater, enten tal eller tekstrenge.

PRINT - sætningen har følgende opbygning:

```
<print-sætning>     ::= PRINT !
                        PRINT <printlist> <printafslut> !
                        PRINT USING <strengudtryk> : <printlist>

<printlist>         ::= <printelement> !
                        <printlist> <printseparator> <printelement>

<printelement>      ::= <aritmetisk udtryk> ! <strengudtryk> ! <tabudtryk>

<tabudtryk>         ::= TAB '('<aritmetisk udtryk>')'

<printafslut>       ::= 'tom' ! <printseparator>

<printseparator>    ::= , ! ;
```

En PRINT - sætning uden 'printlist' bevirker, at den følgende udskrift starter i første position på næste linie.

Parametren 'printseparator' har indflydelse på udskriftens placering på linien. Anvendes ',' vil en linie være opdelt i 5 søjler, der starter i position 1, 17, 33, 49 og 65 og et printelement vil starte i næste søjle. Er separatoren ';' vil udskriften blive pakket, således at der udskrives et blanktegn foran og bagefter tal, mens tegnstrenge udskrives uden omgivende blanktegn.

Ved udskrift af relle tal gælder følgende regler:

- tal i området 1 <= x <= 9999999 udskrives på decimal form uden eksponent og med maksimalt 7 cifre.
- decimaltal, hvor alle cifre efter decimalpunktum er nul, ududskrives som heltal uden decimalpunkt.
- efterstillede nuller i decimaldel undertrykkes.
- eventuelt nul foran decimalpunktum udskrives
- tal i området x < 1 eller x >= 10000000 udskrives med eksponent og 7 cifre i mantissen, heraf 1 ciffer foran decimapunkt.

TAB er en tabulator-funktion, som bevirker, at næste printelement udskrives i den position, som argumentet angiver. Positionerne på linien nummereres fra 1 til 80. Er den aktuelle position større end argumentet for TAB-funktionen, ignoreres denne.

PRINT USING benyttes, når programmet selv ønsker at styre udskriften af resultater i 'printlist'. 'Strengudtryk' benyttes som formatstreng. Tegnene i dette udtryk fortolkes på følgende måde:

'#'	cifferposition og fortegn
'.'	decimalpunktum (kun omgivet af #)
	alle andre tegn overføres direkte til udskrift.

Kan formatet ikke opfyldes, udskrives '*'-er
Fortegnet kræver altid 1 position, også selvom tallet er positivt, i hvilket tilfælde der udskrives et blanktegn.

4.6 READ - sætning 4.6

READ - sætningen benyttes til at tildele startværdier til reelle variable og strengvariable.

READ - sætningen har følgende opbygning:

<read-sætning> ::= READ <variabel liste>

<variabel liste> ::= <variabel> ! <variabel liste> , <variabel>

<variabel> ::= <reel variabel> ! <strengvariabel>

Ved udførelsen af READ - sætningen vil variablene i variabellisten få tildelt værdier på følgende måde: Den første værdi udpeget ved 'Datapil' til 'Dataliste' vil blive tildelt den første variabel i 'variabel liste', samtidig med at Datapil vil blive justeret til at pege på næste værdi i Dataliste. Dataliste opbygges ved udførelse af DATA - sætningerne.

Stemmer den læste værdis type ikke med den angivne variables type eller er Dataliste udtømt stoppes udførelsen med fejludskrift.

Før programmet udføres, løbes det igennem, og alle DATA - sætninger findes og kædes sammen til 'Dataliste'.

4.7 INPUT - sætning 4.7

INPUT - sætningen benyttes til at indlæse værdier fra tastaturet og tildele disse til reelle variable og strengvariable.

INPUT - sætningen har følgende opbygning:

<input-sætning> ::= INPUT <input liste>

<input liste> ::= <input element> ! <input liste> , <inputelement>

<input element> ::= <variabel> ! <strengkonstant>

Udførelsen af INPUT - sætningen bevirker, at 'input liste' gennemgåes fra starten, idet strengkonstanter først udskrives, herefter kan brugeren indtaste en linie, der indeholder værdier i

form af relle tal og en tegnkonstant. En reel værdi kan indeholde fortegn, cifre, decimalpunktum og eksponent. Flere reelle værdier adskilles af blanktegn. En strengkonstant skrives som en tegnstreng uden '"'. Den indtastede linie gennemgåes sammen med input listen og værdierne tildeles.
Den indtastede linie gennemgåes sammen med input listen og værdierne tildeles. Indeholder input listen flere 'strengkonstanter', udskrives disse, når de mødes, og der vendes tilbage til indlæsning fra tastaturet.

Ved indtastning kan brugeren anvende de sædvanlige editeringsfaciliteter og indlæsningen afsluttes med 'RETURN'

4.8 DATA - sætning 4.8

DATA - sætningen benyttes til angivelse af værdier, der skal læses ved hjælp af READ - sætningen.

DATA - sætningen har følgende opbygning:

<data-sætning> ::= DATA <værdi liste>

<værdi liste> ::= <værdi> ! <værdi liste> , <værdi>

<værdi> ::= (+ ! -) <reelt tal> ! <strengkonstant>

Ved starten af programmets udførelse (run - kommando) gennemgåes programmet sekventielt, og alle DATA - sætningerne kædes sammen til Dataliste, og Datapil sættes til at pege på den første DATA - sætning i listen.

DATA - sætningerne kan placeres hvor som helst i programmet.

4.9 RESTORE - sætning

4.9

RESTORE - sætningen benyttes til at retablere Datapil til at pege på den første DATA - sætning i Dataliste.

RESTORE - sætningen har følgende opbygning:

```
<restore-sætning>   ::= RESTORE
```

4.10

4.10 Betingede sætninger

ID - COMAL har en række muligheder for at gøre udførelsen af en række sætninger betinget af udregnede værdier. Sproget indeholder den samme mulighed som standard-BASIC med if - then efterfulgt af et sætningsnummer. Det er også muligt at placere en vilkårlig sætning efter 'then'. Ønskes en række sætninger udført, når en betingelse er opfyldt, skrives disse efter 'then' og afsluttes med 'endif'. Endelig findes if - then - else, hvorved en række sætninger kan udføres afhængigt af, om betingelsen er opfyldt eller ikke opfyldt.

Betinget sætning har følgende opbygning:

```
<betinget sætning>      ::= <if-sætning> ! <if/then-sætning> !
                            <if/then/else-sætning>

<if-sætning>            ::= IF <logisk udtryk> THEN <sætningsnummer> !
                            IF <logisk udtryk> THEN <sætning>

<if/then-sætning>       ::= IF <logisk udtryk> THEN
                            <sætningsliste>
                            ENDIF <kommentar>

<if/then/else-sætning> ::= IF <logisk udtryk> THEN
                            <sætningsliste>
                            ELSE <kommentar>
                            <sætningsliste>
                            ENDIF <kommentar>
```


<sætningsliste> må indeholde vilkårlige sætninger, også betingede sætninger.

Ved en udførelse af en betinget sætning udregnes det logiske udtryk efter 'if'. Er værdien af dette 'sand' udføres sætningen/erne efter 'then' indtil det eventuelt tilhørende 'else', hvorefter programudførelsen fortsætter med den næste sætning efter det tilhørende 'endif'. Er værdien af det logiske udtryk 'falsk' afhænger det af den betingede sætnings form. Ved en 'if-sætning' fortsættes med næste sætning. For en 'if/then-sætning' fortsættes med den næste sætning efter det tilhørende 'endif'. Er det en if/then/else-sætning' fortsættes efter 'else'.

4.11 FOR/NEXT - sætning

4.11

FOR/NEXT - sætningen benyttes, når en række sætninger skal udføres et antal gange.

FOR/NEXT - sætningen har følgende opbygning:

```
<for/next-sætning   ::= <for-sætning>
                    ::= <sætningsliste>
                    ::= <next-sætning>

<for-sætning>       ::= FOR <simpel variabel> = <startværdi>
                        TO <slutværdi> (STEP <trinværdi> ) DO

<startværdi>        ::= <aritmetisk udtryk>

<slutværdi>         ::= <aritmetisk udtryk>

<trinværdi>         ::= <aritmetisk udtryk>

<next-sætning>      ::= NEXT <simpel variabel>
```

Er 'trinværdi' udeladt, vil dette betyde værdien 1.

<sætningsliste> må indeholde vilkårlige sætninger, også FOR/ NEXT - sætninger.

Ved udførelsen af en FOR - sætning vil den simple variabel (styrevariablen) få tildelt værdien, 'startværdi', og sætningsudførelsen vil fortsætte med den næste sætning i programmet. Ligeledes oprettes en indgang i programsætningsstakken med information om FOR - sætningen til brug ved udførelsen af den tilhørende NEXT - sætning. Derefter testes om området har nogen mening dvs.

<slutværdi - startværdi> * SGN(trinværdi) >= 0

Er dette ikke tilfældet overspringes FOR/NEXT - sætningen.

Når en NEXT - sætning udføres, testes først, om den simple variable angivet efter 'NEXT' er identisk med styrevariablen i den tilhørende FOR - sætning. Er dette ikke tilfældet, afsluttes kørslen med fejludskrift. Ellers udregnes 'trinværdi' og denne adderes til værdien af styrevariablen. Såfremt styrevariablens nye værdi ligger i området mellem 'startværdi' og 'slutværdi' fortsættes programudførelsen med første sætning efter FOR - sætningen, og ellers fortsættes med næste sætning efter NEXT - sætningen, idet den tilhørende indgang i programsætningsstakken afstakkes.

4.12 WHILE/ENDWHILE - sætning

4.12

WHILE/ENDWHILE - sætningen benyttes, når en række sætninger skal udføres, sålænge en betingelse er opfyldt.

WHILE/ENDWHILE - sætningen har følgende opbygning:

```
<while/endwhile-sætning> ::= <while-sætning>
                             <sætningsliste>
                             <endwhile-sætning>
```



```
<while-sætning>     ::= WHILE <logisk udtryk> DO

<endwhile-sætning>  ::= ENDWHILE <kommentar>
```

<sætningsliste> må indeholde vilkårlige sætninger også WHILE/ENDWHILE - sætninger.

Når en WHILE - sætning udføres, oprettes først en indgang i programsætningsstakken med information om WHILE - sætningen. Derefter udregnes værdien af det logiske udtryk efter 'WHILE'. Har dette værdien 'sand', fortsættes programudførelsen med den næste sætning. Er værdien 'falsk', fortsættes udførelsen med den næste sætning efter den tilhørende ENDWHILE - sætning, og indgangen i programsætningsstakken afstakkes.

Udførelsen af en ENDWHILE - sætning bevirker udregning af det logiske udtryk i den tilhørende WHILE - sætning, og programudførelsen fortsætter som ved WHILE - sætningen.

4.13

4.13 REPEAT/UNTIL - sætning

REPEAT/UNTIL - sætningen benyttes, når en række sætninger skal udføres, indtil en betingelse er opfyldt.

REPEAT/UNTIL - sætningen har følgende opbygning:

```
<repeat/until-sætning> ::= <repeat-sætning>
                           <sætningsliste>
                           <until-sætning>

<repeat-sætning>    ::= REPEAT <kommentar>

<until-sætning>     ::= UNTIL <logisk udtryk>
```

<sætningsliste> må indeholde vilkårlige sætninger, også REPEAT/UNTIL - sætninger.

En REPEAT - sætning bevirker oprettelse af en indgang i programsætningsstakken med information om REPEAT - sætningen. Derefter fortsætter programudførelsen med den næste sætning.

Når en UNTIL - sætning udføres, udregnes det logiske udtryk efter 'UNTIL'. Er værdien af dette 'sand' fortsætter udførelsen med den næste sætning efter UNTIL - sætningen, og den øverste indgang i programsætningsstakken fjernes. Er værdien af det logiske udtryk 'falsk', fortsættes med den næste sætning efter den tilhørende REPEAT - sætning.

4.14 LOOP/ENDLOOP - sætning 4.14

LOOP/ENDLOOP - sætningen benyttes til at udføre en række sætninger flere gange.

LOOP/ENDLOOP - sætningen har følgende udseende:

```
<loop/endloop-sætning> ::= <loop-sætning>
                           <sætningsliste>
                           <endloop-sætning>

<loop-sætning>         ::= LOOP <kommentar>

<endloop-sætning>      ::= ENDLOOP <kommentar>
```

<sætningsliste> må indeholde vilkårlige sætninger, også LOOP/ENDLOOP - sætninger.

En LOOP - sætning bevirker, at der oprettes en indgang i programsætningsstakken med information om LOOP - sætningen, og at programudførelsen fortsætter med den næste sætning.

Udførelse af en ENDLOOP - sætning bevirker, at programudførelsen fortsætter med næste sætning efter den tilhørende LOOP - sætning.

Udhop fra en LOOP/ENDLOOP - konstruktion kan ske ved en EXIT - sætning.

4.15 EXIT - sætning 4.15

En EXIT - sætning benyttes til udhop af en løkke eller til retur fra et underprogram.

En EXIT - sætning har følgende opbygning:

```
<exit-sætning>      ::= EXIT <kommentar>
```

EXIT - sætningen bevirker, at programudførelsen fortsætter med næste sætning efter afslutning af den sidst startede løkke eller underprogram dvs. efter den inderste NEXT -, ENDWHILE -, UNTIL-, eller GOSUB - sætning. Samtidigt fjernes den øverste indgang i programsætningsstakken. En EXIT - sætning vil ofte være placeret i en 'if-sætning'.

4.16 RETURN - sætning 4.16

RETURN - sætning benyttes til at returnere fra et underprogram.

RETURN - sætningen har følgende opbygning:

```
<return-sætning>    ::= RETURN <kommentar>
```

Ved udførelse af RETURN - sætningen fortsætter programudførelsen med den næste sætning efter den sætning, hvor kaldet af underprogrammet foregik, og samtidigt afstakkes den øverste indgang i programsætningsstakken.

4.17 PROCEDURE - sætning

PROCEDURE - sætningen benyttes til at definere et underprogram.

PROCEDURE - sætningen har følgende opbygning:

```
<procedure-sætning>     ::= <proc-sætning>
                            <sætningsliste>
                            <endproc-sætning

<proc-sætning>          ::= PROC <identifikator>
                                ( '(' <for.parameterliste> ')' )

<for.parameterliste>    ::= <formel parameter> !
                            <for.parameterliste> , <formel parameter>

<formel parameter>      ::= <identifikator>

<endproc>               ::= ENDPROC <kommentar>
```

PROCEDURE - sætningen definerer at 'sætningsliste' skal opfattes som et underprogram med navnet 'identifikator' efter 'PROC'. Navnet på underprogrammet skal være forskelligt fra alle identifikatorer på brugerdefinerede funktioner samt reelle og strengvariable.

En PROCEDURE - sætning kan placeres et vilkårligt sted i programmet. En eventuel 'for.parameterliste' indeholder en eller flere formelle parametre, der indgår som identifikatorer i procedurens sætningsliste. Formelle parametre kan indgå som simple eller indicerede reelle variable, simple eller indicerede strengvariable, brugerdefinerede funktioner, aktuelle parammtre, procedurer samt filvariable. Navnene på de formelle parametre skal være forskellige fra navnene på andre størrelser.

Ved udførelse af et underprogram vil de formelle parametre blive erstattet af de aktuelle parametre.

4.18 Procedurekald

Procedurekald benyttes til at få udført et underprogram.

Procedurekald har følgende opbygning:

```
<kalde-sætning>      ::= EXEC <identifikator>
                              ( '(' <akt.parameterliste> ')' )

<akt.parameterliste ::= <aktuel parameter> !
                        <akt.parameterliste> , <aktuel parameter>

<aktuel parameter>   ::= <aritmetisk udtryk> ! <strengvariabel> !
                         <identifikator>
```

Denne sætning bevirker, at underprogrammet betegnet ved 'identifikator' startes. Programudførelsen fortsætter i underprogrammet indtil RETURN sætning eller ENDPROC-sætning mødes. Derefter fortsætter programudførelsen med næste sætning efter kaldet.

Indeholder procedurekaldet en aktuel parameterliste, skal antallet af aktuelle parametre og deres type stemme med den formelle parameterliste i procedureerklæringen.

Reglerne for substitution af de formelle parametre med de aktuelle parametre er følgende:

- aktuel parameter er identifikator for
  - en simpel reel variabel
  - en datasætidentifikator
  - en formel parameter
  - en procedure
  - en filvariabel

  da vil formel parameter blive erstattet med en reference til aktuel parameter (call by reference).
- aktuel parameter er en strengvariabel, formel parameter erstattes af reference til strengvariabel samt længde

- aktuel parameter er et aritmetisk udtryk, værdien at udtrykket udregnes og der oprettes en lokal variabel, der initialiseres med værdien (call by value). Ønskes en aktuel variabel opfattet som 'call by value' omgives den af paranteser.

5. SPECIELLE SÆTNINGER 5.

Foruden de i kaptiel 4 nævnte COMAL - sætninger, indeholder COMAL systemet nogle sætninger, der anvendes i specielle tilfælde. Disse omfatter:

- BASIC - sætninger
- sætninger til styring af ydre enheder

BASIC - sætningerne GOTO, GOSUB og ON er medtaget for at gøre det muligt at udføre eksisterende BASIC-programmer.
IN og OUT sætningerne gør det muligt at styre specielle ydre enheder tilsluttet mikrodatamatsystemet (se brugervejledningen).
Diskette og skærm kan ikke styres med disse sætninger!

5.1 GOTO - sætning 5.1

GOTO - sætningen benyttes til at bryde den normale sekventielle programudførelse for at fortsætte programudførelsen med en specifik sætning.

GOTO - sætningen har følgende opbygning:

<goto-sætning> ::= GOTO <sætningsnummer> !
GO TO <sætningsnummer>

Programudførelsen vil fortsætte med sætningen angivet ved heltallet efter 'GOTO'. Findes ingen sætning med dette sætningsnummer, afbrydes programudførelsen med fejlskrift. Det frarådes, at anvende GOTO - sætningen ved udhop fra en indre løkke idet løkkestrukturen herved kan ødelægges, hvorved kørslen senere kan blive afsluttet med fejludskrift. Udhop fra en løkke bør ske med en EXIT - sætning.

5.2 ON - sætning 5.2

ON - sætningen benyttes til at fortsætte programudførelsen afhængig af værdien at et udtryk.

ON - sætningen har følgende opbygning:

<on-sætning> ::= ON <aritmetisk udtryk> GOTO <sætningsnummerliste>

<sætningsnummerliste>::= <sætningsnummer> !
<sætningsnummerliste> , <sætningsnummer>

Ved udførelsen af en ON - sætning udregnes det aritmetiske udtryk mellem 'ON' og 'GOTO'. Den udregnede værdi selekterer det sætningsnummer i listen, hvorfra udførelsen skal fortsætte. Er værdien lig 1, fortsættes med sætningen angivet ved første nummer i listen osv. Er udtrykkets værdi negativt, nul eller større end antallet af sætningsnumre fortsættes med den næste sætning efter ON - sætningen.

5.3 ON ESC - sætning 5.3

ON ESC-sætningen bruges til at angive, at der skal udføres en sætning (som er anført i forbindelse med ON ESC-sætningen), hvis brugeren trykker på 'ESC'-tasten, mens programmet udføres. Hvis en ON-ESC-sætning ikke er udført, vil et tryk på 'ESC'-tasten bevirke, at programudførelsen afbrydes.

ON ESC-sætningen har følgende opbygning:

<on esc-sætning> :: = ON ESC <sætningsnummer>!
ON ESC <sætning>

Bemærk, at udførelsen af selve ON ESC-sætningen i sig selv ikke har nogen virkning. Når sætningen er udført, vil en aktivering af 'ESC'-tasten imidlertid bevirke, at enten <sætning> udføres, eller at programudførelsen fortsætter ved sætningen med nummer <sætningsnummer>. Hvis 'ESC'-tasten aktiveres igen, vil

programudførelsen blive afbrudt, med mindre der er udført endnu en ON ESC-sætning.
<sætning> vil ofte være en EXEC-sætning, og den procedure, der kaldes, vil ofte blive indledt af en ny ON ESC-sætning.

5.4 GOSUB - sætning 5.4

GOSUB - sætningen benyttes til at kalde et underprogram.

GOSUB - sætningen har følgende opbygning:

<gosub-sætning> ::= GOSUB <sætningsnummer>

Når en GOSUB - sætning udføres, fortsættes programudførelsen med den sætning, der angives ved 'sætningsnummer', samtidigt stakkes information i programsætningsstakken, således at programudførelsen kan fortsætte når underprogrammet er udført (RETURN - sætning).

Eksisterer der ingen sætning med det givne sætningsnummer, afbrydes kørslen med fejludskrift.

Det er muligt i et underprogram at kalde andre underprogrammer.

5.5 IN - sætning 5.5

IN - sætningen benyttes til læse fra ydre enheder tilsluttet mikrodatamatsystemet.

IN - sætningen har følgende opbygning:

<in-sætning> ::= IN <input port> , <variabel>

<input port> ::= <aritmetisk udtryk>

Fra den port, der udpeges ved 'input port' indlæses en værdi (0 - 255) og denne værdi tildeles 'variabel'.

5.6 OUT - sætning

OUT - sætningen benyttes til at udskrive en værdi til en ydre enhed tilsluttet mikrodatamatsystemet.

OUT - sætning har følgende opbygning:

<out-sætning> ::= OUT <output port> , <værdi>

<output port> ::= <arimetisk udtryk>

<værdi> ::= <aritmetisk udtryk>

Til den port, der udpeges ved 'output port', udskrives 'værdi'. Både 'output port' og 'værdi' tages modulo 256.

Ved hjælp af IN- og OUT-sætningerne er det muligt at skrive enhedsdrivere i COMAL. Anvendelsen af disse sætninger bør ske med stor forsigtighed på grund af sætningernes meget maskinnære status.

5.7 RANDOMIZE - sætning

RANDOMIZE sætningen benyttes, når man ønsker, at de tilfældige tal, genereret af RND - funktionen (se afsnit 6.4), skal starte et tilfældigt sted i sekvensen af (pseudo-) tilfældige tal.

RANDOMIZE - sætningen har følgende opbygning:

<randomize-sætning> ::= RANDOMIZE

Normalt vil RND-funktionen generere samme sekvens af tilfældige tal efter hver RUN-kommando. Dette kan være nyttigt under programafprøvning.

RANDOMIZE - sætningen genererer en tilfældig startværdi for sekvensen på basis af et internt register i maskinen.

5.8 OUTPUT - sætning

OUTPUT sætningen benyttes til at dirigere udskrifter til enten skærmen eller en tilsluttet enhed.

OUTPUT-sætningen har følgende opbygning:

<output-sætning> :: = OUTPUT <output-enhed>
<output-enhed> :: = P[RINTER] ! C[ONSOLE]

Det er tilstrækkeligt at skrive første bogstav af navnet.

Sætningen bevirker, at udskriften fra de efterfølgende PRINT og PRINT USING sætninger bliver dirigeret til den specificerede enhed.

6. STANDARDFUNKTIONER 6.

COMAL indeholder en række standardfunktioner, som kan indgå i aritmetiske udtryk og strengudtryk. De har alle et tre-bogstavs navn og ét argument.

6.1 Matematiske funktioner 6.1

SIN(X): sinus til X, hvor X angives i radianer.

COS(X): cosinus til X, hvor X angives i radianer.

TAN(X): tangens til X, hvor X angives i radianer.

ATN(X): arcus tangens til X, resultatet er i radianer.

LOG(X): den naturlige logaritme af X, X>0

EXP(X): eksponentialfunktionen af X,
-88 < X < 88

SQR(X): kvadratroden af X, X> = 0

Argumenterne kan være aritmetiske udtryk.

6.2 Andre aritmetiske funktioner 6.2

ABS(X): den absolutte værdi af X

INT(X): den hele del af X, dvs. det nærmeste heltal, som er mindre end eller lig med X.
INT (3.5) = 3 ; INT(-3.5) = -4

SGN(X): X < 0 : SGN(X) = -1
X = 0 : SGN(X) = 0
X > 0 : SGN(X) = 1

Argumenterne kan være aritmetiske udtryk.

6.3 Streng-orienterede funktioner 6.3

CHR(X): returnerer et tegn, hvis ASCII-værdi (se APP. A) er X modulo 256.
CHR - funktionen kan anvendes i strengudtryk.

LEN(<strengudtryk>): returnerer et reelt tal, som er lig den aktuelle længde af <strengudtryk>.

ORD(<strengudtryk>): returnerer ASCII-værdien af det første tegn i <strengudtryk>.

6.4 Diverse funktioner 6.4

RND(X): returnerer et pseudo-tilfældigt tal mellem 0 og 1. Argumentet benyttes ikke og ændres ikke. Se også afsnit 5.7.

TAB(X): benyttes i PRINT - sætninger (se afsnit 4.5).

7. FILSYSTEM

7.

COMAL besidder faciliteter for behandling af data lagret på disketter. Data på disketten er organiseret i form af filer, hvor en fil består af en række poster med fast længde.

En diskette kan højst indeholde 90 filer.

Før der læses eller skrives fra/til en fil, skal denne oprettes ved en create sætning eller åbnes ved en open-sætning. I de to initialiseringssætninger skal der som parametre angives filnavnet, som skal placeres i kataloget, eller som findes i kataloget. Desuden angives en identifikator, filreferencen, som benyttes ved alle senere referencer til filen.

Til brug som buffer ved overførsel til/fra filen skal der angives en vektor (reel eller streng), der er stor nok til at rumme en blok på 128 oketter, dvs. enten en strengvariabel med 128 tegn eller et reelt talsæt med 32 elementer. Denne buffer må ikke benyttes til andre formål, sålænge filen er åben.

I de to initialiseringsætinger skal postlængden angives. Ved createsætningen angives også antallet af poster i filen. Postlængden angiver antallet af oktetter i posten. Reelle tal fylder 4 oktetter, mens strengvariable fylder et antal oktetter svarende til antallet af tegn.

Læsning eller skrivning fra/til en fil sker ved angivelse af filreference og postnummer. De variable, hvortil/fra der skrives/læses angives ved en liste. Variablene kan omfatte reelle variable eller strengvariable. Angives identifikatorer for reelle datasæt eller strengvektorer, læses eller skrives alle elementerne.

Efter en læsning eller skrivning vil filreferencen tildeles værdien af returkoden ved operationen (se app. B). Filvariablen vil kunne indgå som operand i aritmetiske og logiske udtryk.

Når behandlingen af filen er afsluttet, udføres en close-sætning på filen, hvorved den tilhørende buffer og filvariabel kan benyttes til andre formål.

COMAL's filsystem omfatter følgende sætninger:

```
<fil-sætning>      ::= <create-sætning> !
                       <delete-sætning>
                       <open-sætning> !
                       <get-sætning> !
                       <put-sætning> !
                       <close-sætning>!
                       <chain-sætning>
```

I det følgende beskrives hver enkelt sætningstype.

7.1 CREATE - sætning

7.1

CREATE - sætning benyttes til at oprette en ny fil for lagring af data.

En CREATE-sætning har følgende opbygning:

```
<create-sætning>   ::= CREATE <filnavn> , <filvariabel> , <filbuffer> ,
                       <postlængde> , <postantal>

<filnavn>          ::= <strengudtryk>

<filvariabel>      ::= <identifikator>

<filbuffer>        ::= <identifikator> ! <strengidentifikator>

<postlængde>       ::= <aritmetisk udtryk>

<postantal>        ::= <aritmetisk udtryk>
```

Når denne sætning udføres, oprettes en fil med navn, som angivet i første parameter. Filens størrelse vil blive 'postlængde' * 'postantal' tegn, dog forhøjet til det nærmeste antal hele blokke.

Parametren 'filbuffer' skal være en identifikator for en reel indiceret variabel eller en strengvariabel, der er lang nok til at rumme en blok (128 oktetter) fra filen. Denne buffer må ikke anvendes til andre formål.

Efter oprettelse af filen vil indholdet være udefineret.

Der kan højst oprettes 90 filer på en diskette.

Udføres sætningen korrekt vil 'filvariabel' indeholde værdien '0' Kan sætningen ikke udføres korrekt, vil værdien være forskellig fra 0 (se app. B). Programudførelsen vil blive afbrudt med en fejlmelding, hvis filvariabel eller buffervariabel er brugt til andre formål, eller hvis filnavn eller postlængde ikke er lovlige.

7.2 DELETE - sætning 7.2

DELETE-sætning benyttes til at slette en fil på en diskette.

En DELETE-sætning har følgende opbygning:

<delete-sætning> :: = DELETE <filnavn>

Hvis filen ikke findes eller har forkert type afbrydes programudførelsen med en fejlmelding. Filen skal være lukket med en CLOSE-sætning før den slettes.

7.3 OPEN - sætning 7.3

OPEN - sætning benyttes til at åbne en allerede eksisterende fil for læsning eller skrivning.

En OPEN - sætning har følgende opbygning:

<open-sætning> ::= OPEN <filnavn> , <filvariabel> , <filbuffer> , <postlængde>

Når denne sætning udføres åbnes filen betegnet med 'filnavn' for læsning og skrivning. Filen skal være oprettet i et tidligere COMAL-job, og der kan eventuelt tidligere være skrevet poster i filen.

Parametren 'filbuffer' skal være identifikator for en reel indiceret eller en strengvariabel, der er lang nok til at rumme en blok (128 oktetter) fra filen. Denne buffer må ikke anvendes til andre formål.

Postlængde angiver længden af posterne i oktetter benyttet i de efterfølgende GET - og PUT - sætninger.

Efter udførelse af sætningen vil 'filvariabel' indeholde returkoden, der kan testes i logiske udtryk. Returkode = 0 vil betyde korrekt åbning. Se iøvrigt app. B. og afsnit 7.1 om create-sætningen.

7.4 GET - sætning 7.4

GET - sætning benyttes til at indlæse reelle værdier og strengværdier til reelle variable og strengvariable fra en fil.

GET - sætningen har følgende opbygning:

<get-sætning> ::= GET <filvariabel> , <postnummer> : <variabelliste>

<postnummer> ::= <aritmetisk udtryk>

<variabelliste> ::= <variabelelement> !
<variabelliste> , <variabelelement>

<variabelelement ::= <reel variabel> ! <strengvariabel> !
<identifikator> ! <strengidentifikator>

Parametren, 'filvariabel' skal angive en idenfikator nævnt i en tidligere udført CREATE - eller OPEN - sætning. 'postnummer' angiver det sted i filen, hvorfra læsning skal starte.

De variable, hvortil der læses, angives ved 'variabelliste'. I denne liste kan findes simple og indicerede reelle variable, simple og indicerede strengvariable samt identifikatorer for datasæt og vektorer af strenge. I sidste tilfælde vil hele datasættet eller strengvektoren få tildelt en værdi. Det antal oktetter, der læses, kan være større end filens postlængde.

Ved læsning fra filen udføres ingen kontrol af om de læste værdier passer med variabellistens typer.

Efter udførelse af sætningen vil 'filvariabel' få tildelt værdien af returkoden. Returkode = 0 betyder korrekt læsning, mens <> 0 angiver fejl (see app. B). Hvis filen ikke er åbnet, afbrydes programudførelsen med en fejludskrift.

7.5 PUT - sætning 7.5

PUT - sætning benyttes til at udskrive reelle værdier og strengværdier fra reelle variable og strengvariable til en fil.

PUT - sætningen har følgende opbygning:

```
<put-sætning>       ::= PUT <filvariabel> , <postnummer> : <variabelliste>

<postnummer>        ::= <aritmetisk udtryk>

<variabelliste>     ::= <variabelelement> !
                        <variabelliste> , <variabelelement>

<variabelelement>   ::= <reel variabel> ! <strengvariabel> !
                        <identifikator> ! <strengidentifikator>
```

Parametren, 'filvariabel' skal angive en identifikator nævnt i en tidligere udført CREATE - eller OPEN - sætning. 'Postnummer' angiver det sted i filen, hvortil skrivning skal starte. De variable, hvorfra der læses, angives ved 'variabelliste'. I denne liste kan findes simple og indicerede strengvariable samt identifikatorer for datasæt og vektorer af stenge. I sidste tilfælde vil hele datasættet eller strengvektoren blive udskrevet. Det antal oktetter, der skrives, kan være større end filens postlængde. Det vil sige, at man (utilsigtet) kan komme til at overskrive data i den efterfølgende post. Efter udførelse af sætningen vil 'filvariabel' få tildelt værdien af returkoden. Returkode = 0 betyder korrekt skrivning, mens <> 0 angiver fejl (se app. B). Hvis filen ikke er åbnet, afbrydes programudførelsen med en fejludskrift.

7.6 CLOSE - sætning 7.6

CLOSE - sætningen benyttes til at afslutte behandlingen af en fil.

CLOSE - sætningen har følgende opbygning:

```
<close-sætning>     ::= CLOSE <filvariabel>
```

Den fil, der udpeges ved filvariabel, lukkes, hvorved den tilhørende filbuffer kan benyttes til andre formål.

7.7 CHAIN - sætning 7.7

CHAIN-sætningen benyttes, hvis man ønsker, at systemet, når et program er færdigt, automatisk skal indlæse et andet fra disketten og starte dette.

CHAIN-sætningen har følgende opbygning:

```
<chain-sætning> :: = CHAIN <filnavn>
```

Systemet fjerner det aktuelle program fra programområdet, indlæser et nyt program fra filen <filnavn> og starter dette.

Dette svarer til, at brugeren når det aktuelle program er afsluttet, indtaster de tre kommandoer NEW, LOAD <filnavn> og RUN (se kapitel 8).

Hvis filen ikke findes eller har forkert type, afbrydes program-udførelsen med en fejlmelding. I dette tilfælde vil det oprindelige program stadig være i lageret.

8. SYSTEMKOMMANDOER 8.

I RC700 COMAL findes en række kommandoer, der kan anvendes til:

- udvikling af programmer
- afvikling af programmer
- dynamisk fejlfinding
- kalkulator funktioner
- fil håndtering

En kommando udføres øjeblikkeligt og adskiller sig fra en programlinie ved ikke at begynde med et linienummer.
Kommandoen indledes med et navn, der evt. efterfølges af en eller flere parametre.

8.1 Sletning af Programsætninger 8.1

Denne kommando benyttes til at slette en eller flere linier i et program.

Kommandoen har følgende opbygning

<slette-kommando> ::= <linienummer> !
<start linienummer>,<slut linienummer>

Benyttes den sidstnævnte form slettes alle linier fra og med linie <start linienummer> til og med <slut linienummer>

8.2 AUTO 8.2

AUTO - kommandoen benyttes, når systemet automatisk skal generere voksende linienumre ved programindtastning.

AUTO - kommandoen har følgende opbygning:

<auto-kommando> ::= AUTO (<startlinie> (, <interval>))

Når denne kommando er indtastet, vil programindtastningen gå ind i en autolinieringstilstand, hvor systemet automatisk genererer sætningsnumre, før indtastning af en linie.

Den første parameter, 'startlinie' angiver sætningsnummer for første linie. Når denne programlinie er indtastet, adderes 'interval' til sætningsnumret, og dette vil blive udskrevet. Angives ingen parameter 'interval' vil denne være 10. Udelades 'startlinie' sættes denne til 10.

Autolinieringstilstanden forlades ved at trykke på 'ESC'.

8.3 BYE 8.3

BYE - kommandoen anvendes når man ønsker at forlade COMAL og udføre et andet af de programmer, som er indeholdt i RC700-systemet (se brugervejledningen).

BYE - kommandoen har følgende opbygning:

<bye-kommando> ::= BYE

BYE - kommandoen sletter program- og dataområdet.

8.4 CON 8.4

CONtinue - kommandoen benyttes til at genstarte udførelsen af et program, der er stoppet som følge af en fejl, med ESCape, eller efter udførelse af en END eller STOP sætning.

CONTINUE - kommandoen har følgende opbygning:

<continue-kommando> ::= CON

Kommandoen bevirker af program-udførelsen genoptages med sætningen umiddelbart efter den sætning, der forårsagede standsning af udførelsen.

Dataområdet, der rummer variables værdier, samt stakken med information om underprogramkald og løkke-sætninger berøres ikke. Man skal derfor være forsigtig med at ændre i programmet, før CON-kommandoen benyttes.

8.5 DELETE 8.5

DELETE - kommandoen benyttes til at slette en fil på disketten.

DELETE - kommandoen har følgende opbygning:

<delete-kommando> :: DELETE <filnavn>

Bemærk: Systemfiler kan ikke slettes. Angående filtyper se afsnit 8.11.

8.6 EDIT 8.6

EDIT - kommandoen benyttes til at rette i en eller flere allerede indtastede programlinier.

EDIT - kommandoen har følgende opbygning:

<edit-kommando> ::= EDIT <edit område>

<edit område> ::= 'tom' !
<start linienummer> !
<start linienummer> , <slut linienummer> !
<start linienummer>, *

Mangler 'edit område', betyder det hele programmet. Mangler 'slut linienummer' vil området kun bestå af den sætning, der udpeges ved 'start linienummer'. Parameteren '*' betyder til og med sidste sætning i programmet.

Ved udførelsen af EDIT - kommandoen vil første sætning i området blive udskrevet nederst på skærmen og markøren vil blive placeret efter linienumret. Brugeren kan nu ved hjælp af systemets editeringsfaciliter (se afsnit 2.3) rette den pågældende sætning. Når sætningen er rettet, tastes 'RETURN' og sætningen syntaksanalyseres og lagres. Derefter udskrives den næste sætning der så kan rettes.

EDIT - kommandoen afsluttes, når den sidste sætning i området er blevet rettet, eller brugeren trykker på ESCape.

Det er muligt at ændre linienummeret, virkningen af dette er, at linien placeres svarende til linienumret.

8.7 HELP

HELP - kommandoen udskriver navnene på de kommandoer, der er indeholdt i systemet.

HELP - kommandoen har følgende opbygning:

<help-kommando> ::= HELP

8.8 LET

LET - kommandoen benyttes til at tildele værdier direkte fra konsollen.

LET - kommandoen har følgende opbygning:

<let-kommando> ::= LET <tildelingsliste>

LET - kommandoen har samme virkning som LET - sætningen.

Det har kun mening at anvende LET - kommandoen før CON eller RUN <linienummer>, da data-arealet slettes ved udførelse af en RUN-kommando.

8.9 LIST

LIST - kommandoen benyttes til udskrivning af en eller flere linier i det indtastede program.

LIST - kommandoen har følgende opbygning:

<list-kommando> ::= LIST <list område>

<list område> ::= 'tom' |
<start linienummer> |
<start linienummer> , <slut linienummer> |
<start linienummer>, *

Mangler 'list område' betyder det hele programmet. Mangler 'slut linienummer' vil området kun bestå af den sætning, der udpeges ved 'start linienummer'. Parameteren '*' betyder til og med sidste sætning i programmet. Ved udførelsen af LIST - kommandoen udskrives sætningerne i det angivne område.

Ved udskriften vil variable og brugerdefinerede funktioner udskrives med store bogstaver, mens nøgleord og systemdefinerede funktioner udskrives med små bogstaver. For at lette læseligheden vil der ske indrykning af sætninger i løkker og if/then(/else)-sætninger.

Listningen kan ske enten på skærmen eller på en tilsluttet printer (se OUTPUT-kommandoen, afsnit 8.13).

8.10 LOAD

LOAD - kommando benyttes til indlæsning af et program lagret i en fil på disketten.

LOAD - kommandoen har følgende opbygning.

<load-kommando> ::= LOAD <filnavn>

Når programmet er indlæst, kan det startes, LIST'es eller modificeret på samme måde, som hvis det var indtastet fra tastaturet.
Før programmet indlæses, udfører systemet en NEW - kommando, så programområdet er tomt.

8.11 LOOKUP 8.11

LOOKUP - kommandoen benyttes til udskrivning af navnene på de filer, der findes på disketten.

LOOKUP - kommandoen har følgende opbygning:

<lookup-kommando> ::= LOOKUP

Filerne listes på følgende form:

<filnavn> <fil-type> <længde>

<fil-type> : en ét-bogstavs kode, der angiver filens type.
Følgende typer findes:
s systemfil
b fil, der indeholder et SAVE'd program
d datafil, oprettet af et COMAL-program

<længde> : filens længde i blokke a 128 oktetter.

Listningen kan fås enten på skærmen eller på en tilsluttet printer (se afsnit 8.13).

8.12 NEW 8.12

NEW - kommandoen benyttes til at slette et allerede eksisterende program og de tilhørende data, således at der kan ske indtastning af et nyt program.

NEW - kommandoen har følgende opbygning:

<new-kommando> ::= NEW

Der udføres automatisk en NEW - kommando ved opstart og før et program indlæses ved hjælp af LOAD - kommandoen.

8.13 OUTPUT 8.13

OUTPUT - kommandoen benyttes til at dirigere udskrifter til enten skærmen eller tilsluttet printer.

OUTPUT - kommandoen har følgende opbygning:

<output-kommando> ::= OUTPUT <output-enhed>

<output-enhed> ::= P[RINTER] ! C[ONSOLE]

Det er tilstrækkeligt at skrive første bogstav af navnet.

Kommandoen bevirker at udskrifter fra PRINT og PRINT USING sætninger samt fra LIST OG LOOKUP kommandoer bliver dirigeret til den specificerede enhed.
Fejludskrifter samt udskrifter fra INPUT - sætninger vil altid fremkomme på skærmen.
Efter udførelsen af en kommando udfører systemet automatisk en 'OUTPUT C' - kommando, således at udskrifter fremkommer på skærmen indtil en ny 'OUTPUT P' -kommando udføres.

8.14 PRINT 8.14

PRINT - kommandoen benyttes til at udskrive værdien af reelle variable og strengvariable.

PRINT - kommandoen har følgende opbygning:

<print-kommando> ::= PRINT <printlist>

PRINT - kommandoen har samme virkning som PRINT - sætningen.

8.15 RENUMBER 8.15

RENUMBER - kommandoen benyttes til at omnummerere sætningerne i et program.

RENUMBER - kommandoen har følgende opbygning:

```
<renumber-kommando>    ::= RENUMBER <område specifikation>

<område specifikation> ::= 'tom' !
                           <start linienummer> !
                           <start linienummer> , <trin> !
                           , <trin>
```

Programmet omnummereres således at første linie i programmet får 'start linienummer'. Såfremt denne parameter ikke er specificeret antages startværdien =10. Derefter adderes 'trin' til denne værdi, og dette bliver linienumret for næste sætning osv. Mangler 'trin' benyttes 10.

Ved omnummereringen behandles også sætningsnumrene i hop-sætninger (GOTO-, GOSUB-, IF-THEN- og ON-sætninger), således at disse bliver erstattet med de nye linienumre.

8.16 RUN 8.16

RUN - kommandoen benyttes til at starte udførelsen at et program.

RUN - kommandoen har følgende opbygning:

```
<run-kommando>      ::= RUN ! RUN <linienummer>
```

De to former af kommandoen har følgende virkninger:

RUN: Programmet, der er lagret i maskinens lager, udføres fra sætningen med det laveste linienummer.

Før programudførelsen slettes dataområdet, og stakken, der rummer information om underprogramkald og løkkesætninger, initialisers.

RUN <linienummer>: Programudførelse starter med sætningen, specificeret ved <linienummer>.
Alle data og informationer fra en tidligere udførelse bevares.
Denne version af kommandoen anvendes til at genoptage udførelsen efter at programmet er stoppet med ESCape, eller efter en fejl.

Udskrifter fra PRINT sætninger vil fremkomme på skærmen eller, hvis det er specificeret ved hjælp af OUTPUT - kommandoen (afsnit 8.13) på en tilsluttet printer.

8.17 SAVE 8.17

SAVE - kommandoen benyttes til lagring af et indtastet COMAL program i en fil på disketten.
SAVE - kommandoen har følgende opbygning:

```
<save-kommando>     ::= SAVE <filnavn>
```

En ny fil med navnet <filnavn> oprettes, og programmet skrives i filen. Eksisterer <filnavn> allerede, slettes denne, og en ny fil med samme navn oprettes.
Programmet kan senere indlæses til hovedlageret igen ved hjælp af LOAD - kommandoen eller CHAIN-sætningen (se kapitel 7).

8.15 RENUMBER

RENUMBER - kommandoen benyttes til at omnummerere sætningerne i et program.

RENUMBER - kommandoen har følgende opbygning:

<renumber-kommando> ::= RENUMBER <område specifikation>

<område specifikation> ::= 'tom' !
<start linienummer> !
<start linienummer> , <trin> !
, <trin>

Programmet omnummereres således at første linie i programmet får 'start linienummer'. Såfremt denne parameter ikke er specificeret antages startværdien =10. Derefter adderes 'trin' til denne værdi, og dette bliver linienumret for næste sætning osv. Mangler 'trin' benyttes 10.

Ved omnummereringen behandles også sætningsnumrene i hop-sætninger (GOTO-, GOSUB-, IF-THEN- og ON-sætninger), således at disse bliver erstattet med de nye linienumre.

8.16 RUN

RUN - kommandoen benyttes til at starte udførelsen at et program.

RUN - kommandoen har følgende opbygning:

<run-kommando> ::= RUN ! RUN <linienummer>

De to former af kommandoen har følgende virkninger:

RUN: Programmet, der er lagret i maskinens lager, udføres fra sætningen med det laveste linienummer.

Før programudførelsen slettes dataområdet, og stakken, der rummer information om underprogramkald og løkkesætninger, initialisers.

RUN <linienummer>: Programudførelse starter med sætningen, specificeret ved <linienummer>.
Alle data og informationer fra en tidligere udførelse bevares.
Denne version af kommandoen anvendes til at genoptage udførelsen efter at programmet er stoppet med ESCape, eller efter en fejl.

Udskrifter fra PRINT sætninger vil fremkomme på skærmen eller, hvis det er specificeret ved hjælp af OUTPUT - kommandoen (afsnit 8.13) på en tilsluttet printer.

8.17 SAVE

SAVE - kommandoen benyttes til lagring af et indtastet COMAL program i en fil på disketten.
SAVE - kommandoen har følgende opbygning:

<save-kommando> ::= SAVE <filnavn>

En ny fil med navnet <filnavn> oprettes, og programmet skrives i filen. Eksisterer <filnavn> allerede, slettes denne, og en ny fil med samme navn oprettes.
Programmet kan senere indlæses til hovedlageret igen ved hjælp af LOAD - kommandoen eller CHAIN-sætningen (se kapitel 7).

RETURN LETTER

Title: RC700 COMAL Brugervejledning RCSL No.: 42-11578

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

Do you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Name: _____ Title: _____

Company: _____

Address: _____

Date: _____

Thank you

42-i1298

..... Fold here .....

..... Do not tear - Fold here and staple .....

Affix postage here

REGNECENTRALEN af 1979
Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark