METANIC COMAL-80

COMAL-80
EDP
– the key to programming

METANIC COMAL-80 og den tilhørende dokumentation, er copyrighted af METANIC ApS.

Det er lovstridigt at kopiere nogen del af programmerne, eller den hertil hørende dokumentation, til kassettebånd, floppy disk eller et hvilketsomhelst andet medium, hvis denne kopiering har noget andet formål end personlig behagelighed.

Det er lovstridigt at forære bort eller sælge nogen del af programmerne eller den hertil hørende dokumentation. Enhver krænkelse heraf kan skade rettighedsindehaverne alvorligt, og vil blive søgt bragt til ophør med alle legale midler, herunder krav om erstatning.

Hvis der er spørgsmål til disse rettigheder, kontakt venligst:

METANIC ApS
BYVEJEN 11
3660 STENLØSE
TELF.: 02 172728

(TM) METANIC COMAL-80 er registreret varemærke for METANIC ApS

(R) CP/M er registreret varemærke for DIGITAL RESEARCH, INC.

(R) Z-80 er registreret varemærke for ZILOC, INC.

ET ER ET SØKORT AT FORSTÅ
ET ANDET ET SKIB AT FØRE

Denne sentens blev oprindelig sagt for mange år siden, længe før ord som byte, nanosekunder, datamater og fortolkere kom til verdenen.

På trods heraf, har vi ofte tænkt på disse ord i den tid, hvor vi har arbejdet med denne håndbog, fordi vi fandt, at det er en vanskelig opgave at beskrive med almindelige ord, hvordan noget så kompliceret som et højniveausprog arbejder.

Det foreliggende værk er imidlertid resultatet af vore samlede anstrengelser, og vi håber resultatet er nogenlunde dækkende. Vi håber desuden, at brugeren med positiv kritik vil udpege de svage områder og derved medvirke til, at næste udgave bliver endnu et skridt nærmere det perfekte resultat, vi ønsker.

Vi er derfor meget interesserede i at modtage alle rettelser, kommentarer, forslag og ønsker, der måtte være til det foreliggende materiale.

Håndbogens format er specielt tilrettelagt for at muliggøre let ajourføring. Der er derfor store chancer for, at det modtagne materiale kan påvirke næste udgave.

Vi har valgt at opstille alle nøgleord i alfabetisk orden, fordi det er en vigtig del af filosofien bag COMAL-80 at gøre anvendelsen af EDB så let som mulig, uden f. eks. at skulle kende de forskellige grupper, nøgleord kan opdeles i.

Vi håber, at det vil være en fornøjelse at arbejde med COMAL-80 og at denne håndbog vil hjælpe til hermed.

ARNE CHRISTENSEN MOGENS PELLE SUSANNE SØNDERSTRUP

Denne håndbog følger nøje specifikationerne for METANIC COMAL-80, der, foruden en fuldt udbygget BASIC, indeholder et stort antal strukturer hentet fra Pascal.

Den foreliggende håndbog er opdelt i to hovedafsnit og et antal tillæg. Afsnit 1 indeholder vejledning i opstart af de forskellige COMAL-80 versioner og en generel beskrivelse af faciliteter, som vedrører flere af eller alle COMAL-80 instruktionerne. Afsnit 2 beskriver syntaks og semantik for hver kommando, instruktion og funktion i alfabetisk orden. De forskellige tillæg indeholder en oversigt over fejlmeldinger, demonstrationsprogrammer, biblioteksprogrammer og en oversigt over ASCII koderne.

Dette værk er ikke tænkt som en lærebog i COMAL-80, men som en håndbog for de specielle faciliteter, som METANIC COMAL-80 indeholder.

Hver af de forskellige COMAL-80 udgaver indeholder to versioner af fortolkeren. De to versioner har identiske faciliteter bortset fra, at den delte version efterlader mere plads til brugeren, og bruger nogle få sekunder ved begyndelsen og afslutningen af hver program-udførelse til at indlæse den nu aktuelle del.

De forskellige filer hedder:

7 cifret nøjagtighed:	
Ikke delt version	COMAL-80.COM
Delt version	COMAL80S.COM
	COMAL-80.2
13 cifret nøjagtighed:	
Ikke delt version	COMAL80D.COM
Delt version	CMAL80DS.COM
	COMAL80D.2

Bemærk, at hver udgave kun indeholder filerne for den ene af de to mulige nøjagtigheder, og at CP/M operativsystemet ikke er inkluderet.

Det tilrådes, at COMAL-80 filerne kopieres til en ny diskette, som også indeholder CP/M operativsystemet. Fjern derefter den originale disk fra datamaten og opbevar den et sikkert sted. Husk, at kun denne diskette bærer garantien.

Skriv nu navnet på den ønskede version uden udvidelsen '.COM', og COMAL-80 læses ind.

Når COMAL-80 er initialiseret, spørges der på terminalen, om der ønskes tekster ved fejlmeldinger. Brugeren må hertil svare 'J' for ja eller 'N' for nej.

COMAL-80 er herefter klar til brug og viser dette med promtkarakteren '*'. Kommandoer og programsætninger kan nu indtastes.

Kommandoer kendes på, at de ikke begynder med et linienummer. Dette viser, at linien skal udføres straks efter, at 'RETURN' tasten nedtrykkes.

Som kommando kan dels anvendes de specielle systemkommandoer som 'RUN', 'LIST' osv., dels kan en stor del af COMAL-80 instruktionerne anvendes, så man f.eks. kan få resultatet af aritmetiske og logiske operationer udskrevet uden at skulle lave et program.

Programsætninger kendes på, at de begynder med et linienummer. Dette viser COMAL-80, at linien skal gemmes til senere udførelse.

Når der trykkes på 'RETURN', syntakskontrolleres linien først, og hvis der ikke er fejl, oversættes den til internt format og gemmes i datamatens arbejdslager. Er der fejl, vises linien igen på terminalen, og cursoren udpeger det punkt, hvori fejlen blev konstateret. Desuden vises et fejlnummer og (hvis ikke frakoblet) en fejlmelding, der nærmere beskriver fejlen.

Under anvendelse af editeringsfaciliteterne i COMAL-80 kan linien nu rettes, og når der igen trykkes 'RETURN', gentages ovennævnte forløb, indtil linien er rigtig.

Når brugeren taster 'RUN' udføres først et prepass, som fuldfører oversættelsen til internt format, hvorefter 'RUN' modulet udfører det aktuelle arbejde.

Instruktioner i COMAL-80 har følgende format:

```
nnnn COMAL-80 instruktion [// kommentar]
```

hvor nnnn er et linienummer i intervallet 1-9999.

Der kan kun være en instruktion på hver linie, dog kan der være flere tildelinger adskilt af semikolon. Se nærmere under 'LET' og 'MAT' instruktionerne.

En COMAL-80 instruktion begynder altid med et linienummer, slutter med en 'RETURN' og kan indeholde op til 159 karakterer. På terminaler med en linielængde mindre end dette, fortsættes automatisk på den næste fysiske linie, når den foregående er fyldt.

EDITERING

Hvis en fejl opdages, medens linien indtastes, kan den rettes blot ved at flytte cursoren tilbage til fejlpunktet og indtaste de korrekte karakterer. De nye karakterer vil erstatte de gamle. Karakteren, der udpeges af cursoren, kan fjernes ved at taste 'DEL' tasten. Samtidig rykker alle karakterer til højre herfor en plads til venstre.

Nye karakterer kan indsættes mellem allerede eksisterende karakterer ved at flytte cursoren hen til det aktuelle punkt og taste 'INS' tasten. Herved rykker den udpegede karakter og resten af linien en plads til højre efterladende en tom position under cursoren. Dette kan gentages til den ønskede plads er tilstede, eller den maksimale linielængde nået.

Når linien afsluttes ved at taste 'RETURN', gemmes hele linien som vist på skærmen, uanset cursorens aktuelle position.

En linie, som er ved at blive indtastet, kan afbrydes med 'ESC' tasten, men samtidig afsluttes også eventuel automatisk generering af linienumre.

Eksisterende programlinier, som er i hukommelsen, kan rettes ved at genskrive linien med samme linienummer eller ved anvendelse af 'EDIT' kommandoen.

'NEW' kommandoen fjerner hele det program, som befinder sig i datamatens hukommelse.

COMAL-80 karaktersættet er sammensat af alfabetiske karakterer, numeriske karakterer og specielle karakterer.

De alfabetiske karakterer er alfabetets store og små bogstaver samt understregning.

De numeriske karakterer er tallene 0-9.

Følgende specialkarakterer benyttes i COMAL-80:

KARAKTER	NAVN
	Mellemrum
=	Ligheds- eller tildelingstegn
+	Plustegn
-	Minustegn
*	Multiplikationstegn
/	Divisionstegn
^	Eksponenttegn
(	Venstreparantes
)	Højreparantes
#	Nummertegn
$	Dollartegn
!	Udråbstegn
,	Komma
.	Punktum eller decimalkomma
"	Anførselstegn
;	Semikolon
:	Kolon
<	Mindre end
>	Større end
_	Understregning
<ESC>	Escape. Stands og afvent input
<RETURN>	<vognretur>
<INS>	Indsæt
<DEL>	Slet

Konstanter er de aktuelle værdier, som COMAL-80 anvender under udførelsen af et program. Der er to typer af konstanter: strenge og aritmetiske.

En strengkonstant er en række af alfanumeriske karakterer omgivet af anførselstegn. Længden af en streng er kun begrænset af den i datamaten værende hukommelse.

Anførselstegn kan indgå som en del af en strengkonstant ved at skrive to anførselstegn ("") umiddelbart efter hinanden.

Karakterer, som ikke kan skrives på datamatens tastatur, kan indgå som del af en streng ved at skrive tegnets decimale ASCII værdi, omgivet af anførselstegn.

Eksempler på strengkonstanter:

```
"COMAL-80"
"KR. 10.000,00"
"LUK DEN DØR"
"TAST ""S"" FOR AT STANDSE"
"STOP"13""
```

Aritmetiske konstanter er positive og negative tal. Aritmetiske konstanter kan i COMAL-80 ikke indeholde komma. I stedet benyttes decimalpunktum. Der er to typer af aritmetiske konstanter:

1. Heltal — Hele tal i området -32767 til 32767. Heltal indeholder ikke decimalpunktum.

2. Reelle tal — Positive eller negative reelle tal, dvs. tal der indeholder decimalpunktum, samt positive og negative repræsenteret på eksponentialform. En reel konstant på eksponentialform består af et positivt eller negativt reelt eller heltal (mantissen) fulgt af bogstavet 'E' og et positivt eller negativt heltal (eksponenten). Desuden betragtes heltal uden for ovennævnte værdiområde som reelle konstanter.

Variable er navne brugt til at repræsentere værdier i et COMAL-80 program. Værdien af en variabel kan tildeles af programmøren, eller den kan tildeles som resultat af udregninger i programmet. Før en variabel er tildelt en værdi, er den udefineret.

VARIABELNAVNE OG DEKLARATIONSKARAKTERER

COMAL-80 variabelnavne kan have enhver længde op til 80 karakterer. De i et variabelnavn tilladte karakterer er bogstaver, tal og understregning. Første karakter skal være et bogstav. Specielle typedeklarationskarakterer er også tilladt. Se nedenfor.

Et variabelnavn må ikke være et reserveret ord, medmindre det reserverede ord er omgivet af andre karakterer. Reserverede ord er alle COMAL-80 kommandoer, instruktioner, funktionsnavne, operatornavne og navne defineret i en "EXTENSION".

Variable kan repræsentere enten en aritmetisk værdi eller en tekststreng. Strengvariabelnavne skrives med et dollartegn ($) som sidste karakter. Heltalsvariabelnavne skrives med et nummertegn (#) som sidste karakter. Dollar- og nummertegnenene kaldes typedeklarationskarakterer, dvs. de erklærer, at den pågældende variabel vil repræsentere en streng eller et heltal.

Eksempler på variabelnavne:

```
A
A8
DISKNAVN$
TÆLLER#
PRINT_DATA
VÆRDI_AF_LØBENDE_OPSPARING
```

En matrice er en gruppe eller tabel af værdier, som refereres med samme variabelnavn. Hvert element i en matrice refereres med et variabelnavn indiceret med et aritmetisk udtryk for hver dimension. Et matricevariabelnavn har lige så mange indices, som der er dimensioner i matricen. Når matricen bruges som parameter, kan den refereres som helhed, eller som en 'matrice af en matrice' ved at udelade nogle eller alle indices. Dette er nærmere beskrevet i afsnittet PARAMETERINDSÆTTELSE.

Alle matricer skal deklareres med en 'DIM' instruktion.

Når en aritmetisk matrice er dimensioneret, men før den er tildelt værdier, har alle dens elementer værdien 0 (nul).

Når en strengmatrice er dimensioneret, men før den er tildelt strenge, indeholder hvert element strengen "" (streng af nullængde, tom streng).

DELSTRENGE

Foruden at der kan refereres til en strengvariabel som helhed eller element for element, kan der refereres til en delstreng.

Dette gøres med en af følgende formater:

```
⟨navn⟩(I1,I2,...In,⟨start⟩,[⟨slut⟩])
⟨navn⟩(I1,I2,...In)(⟨start⟩[:⟨slut⟩])
```

I det første tilfælde undersøges det først, hvor mange dimensioner ⟨navn⟩ er dimensioneret til i den tilhørende 'DIM' instruktion. Derefter bruges de første n parametre i parantesen til at specificere det aktuelle element, hvor n er antallet af dimensioner. Parantesen kan derudover indeholde en eller to parametre, nemlig ⟨start⟩ og ⟨slut⟩. ⟨start⟩ specificerer i hvilken karakterposition delstrengen begynder, og ⟨slut⟩ i hvilken den slutter. Er ⟨slut⟩ udeladt, består delstrengen kun af den karakter, der befinder sig i den ved ⟨start⟩ angivne position.

I det sidste tilfælde indeholder første parantes det nødvendige antal indices, medens anden parantes indeholder ⟨start⟩ og ⟨slut⟩ som beskrevet for første tilfælde.

Hvis ⟨navn⟩ refererer til en simpel strengvariabel, betragtes antallet af dimensioner som værende 0 og paranteserne indeholder kun ⟨start⟩ og ⟨slut⟩. I det sidste tilfælde udelades den første parantes.

De aritmetiske operatorer er:

Prioritet	Operator	Operation	Eksempel
1	^	Potensopløftning	X^Y
2	/	Division	X/Y
2	*	Multiplikation	X*Y
2	DIV	Heltalsdivision	X DIV Y
2	MOD	Modulus	X MOD Y
3	-	Negation	-X
4	+	Addition	X+Y
4	-	Subtraktion	X-Y

At operatorerne har en prioritet betyder, at hvis et udtryk indeholder flere af disse, udføres først den operator, som står højest på ovenstående liste. Indeholder et udtryk flere operatorer af samme prioritet, beregnes udtrykket fra venstre mod højre.

Denne rækkefølge kan ændres ved i udtrykket at indsætte paranteser, idet operationer i paranteser udføres først. Er der flere operatorer inden i en parantes, gælder prioritetsrækkefølgen igen.

Bortset fra negering må de aritmetiske operatorer kun anvendes mellem udtryk, som antager aritmetiske værdier. Negering må kun anvendes foran et udtryk, som antager en aritmetisk værdi.

Den aritmetiske værdi af et sandt logisk udtryk er = 1, medens den aritmetiske værdi af et falsk logisk udtryk er 0.

Relationsoperatorer anvendes til at sammenligne to værdier. Resultatet af en sådan sammenligning kan enten være sand (= 1) eller falsk (= 0). Dette resultat kan derefter anvendes til at påvirke programafviklingen.

Når en aritmetisk værdi anvendes som logisk værdi, bliver værdien 0 regnet som falsk, og alle værdier forskellig fra 0 som sande.

Operator	Relation undersøgt	Eksempel
=	Lighed	X=Y
<>	Forskellig fra	X<>Y
>	Større end	X>Y
<	Mindre end	X<Y
>=	Større end eller lig med	X>=Y
<=	Mindre end eller lig med	X<=Y

(= anvendes også til at tildele en værdi til en variabel.)

Relationsoperatorer anvendes mellem to udtryk, som begge antager en aritmetisk værdi eller to udtryk, som begge antager en alfanumerisk værdi.

Relationsoperatorerne har prioritet lige under de aritmetiske operatorer således at, hvis et udtryk indeholder begge typer, udføres først alle de aritmetiske operationer og derefter relationsoperatoren.

I følgende eksempel:

```
X-2>T+3
```

beregnes først værdien af X-2 og T+3, hvorefter disse 2 værdier sammenlignes.

Ved sammenligning af to alfanumeriske udtryk, sker sammenligningen karakter for karakter under anvendelse af de enkelte karakterers ASCII kode. A er mindre end E, da A har ASCII koden 65 og E ASCII koden 69.

Er to strenge af forskellig længde, men ens således at den korte streng er lig med begyndelsen af den lange, er den korte streng mindst. 'ABE' er således mindre end 'ABEKAT'.

Ved sammenligning mellem strenge gælder alle karakterer inden for anførselstegnene, også mellemrum.

Filnavne følger i princippet CP/M specifikationerne. Da COMAL-80 imidlertid tillader op til 80 karakterer, er kun de første otte betydende og overføres til CP/M. I filnavne er både små og store bogstaver tilladt, men små bogstaver ændres til store.

Efter et punktum, kan en udvidelse på op til tre bogstaver specificeres. Denne udvidelse kan vælges frit, undtagen i forbindelse med 'SAVE' og 'LOAD' kommandoerne, hvor COMAL-80 systemet automatisk forsyner filnavnet med udvidelsen '.CSB'. Det er derfor ikke tilladt at specificere nogen udvidelse i forbindelse med disse kommandoer.

Hvis ingen udvidelse specificeres, bruges automatisk '.CML' i forbindelse med 'ENTER' og 'LIST' kommandoerne, '.DAT' i forbindelse med 'OPEN' kommandoen/instruktionen, '.CAT' ved 'CAT' kommandoen/instruktionen og '.RAN' for randomfiler.

Hele navnet bruges for at specificere en fil. Det betyder, at de to kommandoer:

```
ENTER PROGRAM
ENTER PROGRAM.CML
```

læser den samme fil ind i datamatens hukommelse, medens kommandoen:

```
ENTER PROGRAM.LST
```

indlæser en anden.

Diskdrevnavnet er valgfrit, men behandles som en integreret del af filnavnet. Hvis det udelades, benyttes den aktuelle standardenhed. Hvis det specificeres, skrives det umiddelbart foran filnavnet. Diskdrevnavnet er navnet på den enhed, der skal anvendes. Se senere.

Eksempel:

```
ENTER DK1:PROGRAM.CML
```

Bemærk, at diskdrevnavnene IKKE følger CP/M specifikationerne.

Diskdrevnavnene består af de to bogstaver 'DK' (som betyder disk), efterfulgt af et enhedsnummer og et kolon. Således betyder 'DK0:' det samme som under CP/M 'A:', 'DK1:' betyder det samme som CP/Ms 'B:' osv.

Et tilsvarende princip benyttes for de andre ydre enheder, hvilket betyder, at disse kan anvendes som filer, hvorfra data hentes eller udskrives, afhængig af deres natur.

De navne, der benyttes for de forskellige enheder, er:

```
'LP:' eller 'LP0:'     for linieprinteren
'LP1:'                 for papirperforatoren
'DS:' eller 'DS0:'     for dataskærmen
'KB:' eller 'KB0:'     for tastaturet
```

Eksempel:

```
10 OPEN FILE 0, "KB:", READ
20 OPEN FILE 1, "LP:", WRITE
30 DIM A$ OF 100
40 LOOP
50  INPUT FILE 0: A$
60  PRINT FILE 1: A$
70 ENDLOOP
80 // PROGRAMMET AFBRYDES VED TRYK PA 'ESC' TASTEN
```

Når 'INIT', 'RELEASE', 'DELETE', 'GETUNUT', 'RENAME', 'UNIT' og 'CAT' anvendes som instruktioner, betragtes filnavne som strengudtryk og skal omgives af anførselstegn. Dette er ikke tilladt i kommandoer. En følge af dette er, at et filnavn kan specificeres af ethvert strengudtryk, som i udførelsesøjeblikket har et legalt filnavn som værdi.

Eksempler:

```
100 DELETE "DK0:PROGRAM.CML"
100 OPEN FILE 0,"DK0:"+A$
100 DELETE "DK1:"+A$+".CML"
```

COMAL-80 benytter sit eget format i diskfiler. De normale CP/M formater kan specificeres ved at udvide filnavnet med '/C'. Ved yderligere at udvide filnavnet med '/B' specificeres CP/M's binære format.

Eksempler:

```
    ENTER TEST.BAK/C                  //  LÆS CP/M ASCII FIL
100 OPEN FILE 3,"TEST.XYZ/C/B",READ   // ABEN CP/M BINÆR FIL
100 OPEN FILE 2,"DATA/C",WRITE        // ABEN CP/M ASCII FIL
```

En af de specielle faciliteter i COMAL-80 er de ægte procedurer med parametre.

En procedure er et navngivet programområde, omgivet af nøgleordene 'PROC (navn)' og 'ENDPROC (navn)', og dette programområde kaldes ved brug af nøgleordet 'EXEC (navn)'.

Der er mange lighedspunkter mellem procedurer og subrutiner, som de normalt kendes fra BASIC. De kan altså kaldes fra et eller flere steder i et program, og når proceduren er færdigafviklet, fortsætter programudførelsen i linien efter den kaldende linie; men derudover har de andre fordele, som gør dem til et meget effektivt programmeringsværktøj.

For det første kaldes de ved navn, hvilket betyder at programmøren ikke skal huske i hvilken linie, proceduren er placeret.

For det andet er proceduren ikke-udførbar, indtil den kaldes. Dette betyder, at uanset hvor proceduren er placeret i programmet, overspringes linierne, indtil den kaldes med en 'EXEC (navn)' instruktion, og et sådant kald kan gå både frem og tilbage i programmet.

For det tredie kan parametre sendes med til proceduren, når den kaldes. Dette betyder, at en procedure kan reagere forskelligt og arbejde med forskellige data, hver gang den kaldes.

Der er to typer procedurer, kaldet åbne og lukkede procedurer. Forskellen mellem disse to typer er et spørgsmål om, hvordan proceduren ser de variable, der er brugt i resten af programmet.

Variable brugt i en åben procedure, har samme status som variable brugt i hovedprogrammet, hvilket betyder, at hvis den tildeles en ny værdi inden i proceduren, beholder den denne værdi efter at proceduren er afsluttet, og programudførelsen fortsætter i linien efter den kaldende linie.

De lukkede procedurer derimod, opfører sig på mange måder som separate programmer. Den lukkede procedure har sit eget sæt af variable, som kan dimensioneres og tildeles værdier inden i proceduren, men de er aldrig i stand til at påvirke de variable, som er brugt uden for proceduren, medmindre der foretages noget specielt (referenceparametre eller 'IMPORT' instruktionen). Dette gør det muligt at skrive biblioteksrutiner, som kan bruges som en del af ethvert program, uden at der opstår problemer med, at det samme variabelnavn er brugt både i proceduren og i resten af programmet.

Forskellen mellem de to typer af procedurer kan vises ved følgende to programmer:

```
10 A:=5
20 EXEC TEST
30 PRINT A
40 PROC TEST
50  A:=3
60  PRINT A
70 ENDPROC TEST
```

```
10 A:=5
20 EXEC TEST
30 PRINT A
40 PROC TEST CLOSED
50  A:=3
60  PRINT A
70 ENDPROC TEST
```

Afvikles disse to programmer, vil det første to gange skrive værdien '3', fordi tildelingen i linie 50 vinder over tildelingen i linie 10. Det andet eksempel vil skrive værdierne '3' og '5', fordi proceduren er lukket, og variablen i linie 50 derfor ikke er den samme som variablen i linie 10, selvom de har samme navn. Sagt mere teknisk, er variablen 'A' i første programeksempel global for proceduren, fordi hele programmet kan se og bruge den, men en variabel inden i en lukket procedure er lokal og kan kun bruges inden i proceduren.

En lokal variabel skal også tildeles (linie 50) eller dimensioneres inden i den lukkede procedure, før den bruges første gang. Dette betyder, at hvis linie 50 slettes i det andet eksempel, vil programafviklingen stoppe i linie 60 med en fejlmelding, der fortæller, at variablen er ukendt.

Selvom adskillelsen af variabelnavnene er den grundlæggende ide bag lukkede procedurer, er det ofte bekvemt at kunne gøre en variabel kendt både for hovedprogrammet og for proceduren.

Dette kan gøres med 'IMPORT' instruktionen som vist i det følgende eksempel:

```
10 A:=3
20 EXEC TEST
30 PRINT A
40 PROC TEST CLOSED
50  IMPORT A
60  A:=3*A
70  PRINT A
80 ENDPROC TEST
```

Dette program vil to gange skrive værdien '9'. Bemærk, at 'IMPORT' instruktionen skal placeres inden i den lukkede procedure og før den del af proceduren, som for første gang anvender variablen.

Lukkede procedurer kan kaldes inden i hinanden til enhver dybde, som datamatens hukommelse tillader (hvert niveau bruger mindst omkring 50 byte, afhængig af antallet af variable), men 'GLOBAL' instruktionen virker kun på det niveau, hvor den aktuelt er anbragt. Det følgende eksempel vil skrive værdien '3' (i linie 100) og derefter stoppe i linie 60 med en fejlmelding der fortæller, at variablen er ukendt:

```
 10 A:=3
 20 EXEC TEST1
 30 PRINT A
 40 PROC TEST1 CLOSED
 50  EXEC TEST2
 60  PRINT A
 70 ENDPROC TEST1
 80 PROC TEST2 CLOSED
 90  IMPORT A
100  PRINT A
110 ENDPROC TEST2
```

Variable kan også flyttes ind i og ud af lukkede procedurer ved hjælp af referenceparametre. Se nærmere herom i kapitlet PARAMETERINDSÆTTELSE.

Når en variabel dimensioneres eller tildeles en værdi i en lukket procedure, afsættes den nødvendige hukommelsesplads ikke før proceduren aktuelt kaldes, og denne del af hukommelsen frigives igen, når proceduren er færdigafviklet.

Derfor, uanset hvor mange gange en procedure kaldes, kan fejlmeldingen 'ikke mere hukommelsesplads' ikke komme, hvis ikke den fremkom under det første kald.

Dette 'visken tavlen ren' gør det også muligt at dimensionere en variabel inden i en procedure, som kaldes adskillige gange, uden at komme i konflikt med reglen om, at en variabel ikke kan redimensioneres, og det er muligt at lægge matricer og strengvariable, som bruges til mellemresultater, oven i hinanden og derved økonomisere med hukommelsespladsen blot ved at dimensionere og bruge disse inden i forskellige lukkede procedurer.

Enhver procedure kan kalde enhver procedure defineret hvorsomhelst i hovedprogrammet, og den kan endog kalde sig selv (rekursion). Bemærk, at også rekursion betyder at kalde et nyt niveau, hvorved der bruges hukommelsesplads. Disse kald må derfor nøje kontrolleres.

Reglerne for variable i lukkede procedurer gælder også for den anden lukkede struktur: Den brugerdefinerede funktion.

En vigtig del af COMAL-80 definitionen er procedurer (og brugerdefinerede funktioner) med parametre, som letter opdelingen af et program i mindre, navngivne rutiner. Disse kan være åbne (åbne procedurer) eller lukkede (lukkede procedurer eller brugerdefinerede funktioner).

Til at flytte data ind i og ud af en sådan rutine benyttes parametre, dvs. en liste af variabelnavne specificeret i den kaldende linie (de aktuelle parametre) og i første linie af rutinen (de formelle parametre). De aktuelle parametre indsættes så i de formelle, når rutinen kaldes.

Der er to typer af parametre, nemlig: 'kald med værdi' og 'kald med reference'.

'Kald med værdi' betyder, at den aktuelle værdi af den aktuelle parameter tildeles til den formelle parameter. Denne type kald kan kun føre data ind i en rutine, da ændringer i den formelle parameter ikke kan påvirke den aktuelle parameter.

'Kald med reference' betyder, at den formelle parameter erstattes med den aktuelle parameter. Denne type kan føre data både ind i og ud af en rutine, og specificeres ved at anvende nøgleordet 'REF' foran variabelnavnet i den formelle parameterliste. Denne erstatning sker dynamisk, altså når rutinen kaldes, og kan ikke ses i programudskrifter, som altid viser de formelle parametre.

Følgende eksempler viser forskellen:

```
10 A:=3                    10 A:=3
20 EXEC TEST(A)            20 EXEC TEST(A)
30 PRINT A                 30 PRINT A
40 PROC TEST(X)            40 PROC TEST(REF X)
50  X:=3*X                 50  X:=3*X
60  PRINT X                60  PRINT X
70 ENDPROC TEST            70 ENDPROC TEST
```

Her er 'A' i linie 20 den aktuelle parameter og 'X' i linie 40 den formelle parameter.

I det første eksempel tildeles værdien '3' til 'X' når proceduren 'TEST' kaldes i linie 20 og udskriver værdien '9' i linie 60. Efter at proceduren er afsluttet, udskrives værdien '3' i linie 30, fordi variablen 'A' ikke ændres.

Det andet eksempel vil to gange udskrive værdien '9', fordi den formelle parameter erstattes med den aktuelle, og ændringer derved føres tilbage.

Parametre er altid lokale, hvilket betyder at ændringer, der sker med 'kald med værdi' parametre i en rutine, ikke kan ændre en variabel med samme navn i resten af programmet. Dette vises af følgende eksempel:

```
10 A:=3
20 B:=2
30 EXEC TEST(A)
40 PRINT A,B
50 PROC TEST (A)
60  A:=3*A
70  B:=3*B
80  PRINT A,B
90 ENDPROC TEST
```

For 'A' vil dette program udskrive værdien '9' i linie 80 og derefter værdien '3' i linie 40. Begge linier vil udskrive værdien '6' for 'B'. Med andre ord, den formelle parameter 'A' er lokal for proceduren og en anden variabel end den brugt i linie 10 og 40, hvorimod 'B' ikke er en parameter (og proceduren ikke lukket), så denne variabel er global for proceduren og den samme i hele programmet.

Parameterlisterne kan indeholde lige så mange parametre, som den maksimale linielængde tillader (159 karakterer), adskildt af komma, men der skal være det samme antal parametre i begge lister og korresponderende parametre skal være af samme type og have samme dimension. Den eneste undtagelse er, at en aktuel parameter af heltalstypen kan tildeles til en reel formel parameter, når 'kald med værdi' benyttes.

Konstanter og udtryk kan bruges som aktuelle parametre ved 'kald med værdi'.

Eksempel:

```
10 EXEC TEST(3*5,"FEJL")
20 PROC TEST(A,B$)
30  PRINT A
40  PRINT B$
50 ENDPROC TEST
```

Bemærk, at en formel parameter ikke skal dimensioneres, da kaldet selv bærer den nødvendige information.

Matricer kan anvendes som parametre, enten som helhed, som en matrice af en matrice, eller et enkelt element. I de første to tilfælde kan dog kun 'kald med reference' anvendes.

Når et enkelt element anvendes, specificeres elementet i den aktuelle parameterliste med det nødvendige antal indices, og en variabel af samme type specificeres i den formelle parameterliste.

Eksempel:

```
 10 DIM A(3,5,2)
  .
  .
100 EXEC TEST(A(1,1,1))
  .
  .
200 PROC TEST(B)
  .
  .
300 ENDPROC TEST
```

Bemærk, at 'B' ikke behøver at være en referenceparameter, da kun et enkelt element anvendes.

En matrice af en matrice anvendes ved at udelade en eller flere indices fra højre i den aktuelle parameterliste og udvide det formelle parameternavn med en parantes indeholdende samme antal kommaer, som antallet af udeladte indices minus 1.

Eksempel:

```
 10 DIM A(3,5,2)
  .
  .
100 EXEC TEST(A(1,1))
  .
  .
200 PROC TEST(REF B())
  .
  .
300 ENDPROC TEST
```

Man skal i dette eksempel bemærke, at parantesen efter den formelle parameter 'B' er tom, fordi antallet af udeladte indices er 1.

De udeladte indices specificeres, når den formelle parameter anvendes i rutinen. Dette fremgår af eksemplet på næste side.

Eksempel:

```
 10 DIM ARRAY_OF_VECTORC(5,3)
 20 FOR I:=1 TO 5
 30  FOR J:=1 TO 3
 40   ARRAY_OF_VECTORS(I,J):=RND(1,5)
 50  NEXT J
 60 NEXT I
 70 EXEC CHANGE_SIGN(ARRAY_OF_VECTORS(4))
 80 PROC CHANGE_SIGN(REF VECTOR()) CLOSED
 90  FOR I:1 TO 3
100   VECTOR(I):=-VECTOR(I)
110  NEXT I
120 ENDPROC CHANGE_SIGN
130 FOR I:=1 TO 5
140  FOR J:=1 TO 3
150   PRINT ARRAY_OF_VECTORS(I,J);
160  NEXT J
170 NEXT I
```

En hel matrice kan også anvendes som parameter. Dette gøres ved helt at undlade at specificere indices i den aktuelle parameterliste og udvide den formelle parameterangivelse med en parantes, som indeholder det samme antal kommaer, som matricens dimension minus 1.

Eksempel:

```
 10 DIM A$(5,3,2) OF 25
  .
  .
100 EXEC TEST(A$)
  .
  .
200 PROC TEST(REF B$(,,))
  .
  .
300 ENDPROC TEST
```

COMAL-80 er internt opbygget af tre hovedmoduler kaldet:

Inputmodul
Prepassmodul
Runmodul

Hvert modul har sine egne rutiner til at håndtere fejlsituationer, så de forskellige fejltyper kan håndteres så effektivt som muligt.

Disse rutiner har til deres rådighed et bibliotek af fejlmeldinger, som kan give en kort beskrivelse af mere end 200 forskellige typer af fejl.

Sammen med en fejlmeddelelse vises altid et fejlnummer, og i de fleste tilfælde vises også den fejlramte linie, hvori cursoren udpeger fejlpunktet.

For at give øjeblikkelige fejlmeddelelser, er fejlmeddelelserne en integreret del af COMAL-80. Da dette bibliotek bruger omkring 3K af datamatens hukommelse, kan det meste af det bortkastes, når COMAL-80 initialiseres, hvilket giver brugeren omkring 2.5K ekstra hukommelsesplads.

Bortset fra at selve fejlmeddelelsen mangler, virker resten af fejlmeldesystemet på normal måde og fejlnummeret gør det muligt at finde den manglende tekst i tillæg A i denne håndbog.

SYNTAKSFEJL

Inputmodulet består i virkeligheden af to undermoduler: editoren og syntakskontrollen.

Editoren er en linieorienteret editor, som tillader brugeren at indtaste en linie og ændre den efter ønske. Når en linie afsluttes ved at taste 'RETURN', overføres den til syntakskontrollen og sammenlignes med COMAL-80 specifikationerne.

Hvis der ikke konstateres syntaksfejl, udføres linien, hvis det er en kommando, medens instruktioner oversættes til internt format og gemmes i datamatens hukommelse.

Hvis linien indeholder en syntaksfejl, udskrives et fejlnummer og (hvis ikke frakoblet) en fejltekst samt den fejlramte linie med cursoren pegende på fejlen, og kontrollen går tilbage til editoren. Brugeren skal nu korrigere linien, og ovennævnte forløb gentages, til linien kan accepteres.

Når en ASCII fil indlæses ved hjælp af 'ENTER' kommandoen, syntakskontrolleres hver linie på samme måde. Hvis en fejl konstateres, udskrives den fejlramte linie på skærmen sammen med et fejlnummer og, hvis muligt, en fejltekst samt en pil der viser fejlen. Den fejlramte linie gemmes ikke.

Det er umuligt at gemme en linie som indeholder en syntaksfejl.

PREPASSFEJL

Når brugeren ønsker at afvikle et program og derfor indtaster 'RUN', går prepassmodulet, som er usynligt for brugeren, først i arbejde. Dette modul udvider programmets interne format med bl.a. absolutte hukommelsesadresser og undersøger, om alle strukturer er korrekt afsluttet, og at referencepunkter eksisterer.

Hvis der ikke konstateres fejl, overgives kontrollen til runmodulet.

Hvis en af instruktionerne i en struktur mangler (FOR...NEXT, REPEAT...UNTIL, WHILE...ENDWHILE osv.), udskrives den korresponderende linie, et fejlnummer og muligvis en fejlmelding. Linienumre med kald til ikke-eksisterende 'LABEL' instruktioner vises på samme måde.

Hvis en linie indeholder 'EXIT' instruktionen uden at være omgivet af 'LOOP' og 'ENDLOOP' instruktioner, udskrives linienummeret for 'EXIT' instruktionen.

Alle fejl i hele programmet udskrives samtidig, og kontrollen returneres derefter til inputmodulet. Bemærk, at det ikke er muligt at udføre nogen del af et program, som indeholder en prepassfejl.

UDFØRELSESFEJL

Når runmodulet går i arbejde, kan programmet kun indeholde fejl af dynamisk art, dvs. fejl som findes, når linien udføres. En fejl af denne type standser normalt programafviklingen og udskriver den fejlramte linie med cursoren pegende på fejlen. Desuden udskrives fejlnummeret og muligvis en fejlmelding. Kontrollen overføres derefter til editoren i inputmodulet, så fejlen let kan rettes. Der findes imidlertid et antal fejl, som er ikke-fatale, fordi de kan forbigås på veldefineret måde. Et eksempel herpå er division med nul, hvor det ofte er praktisk at sætte resultatet lig den maksimale værdi, COMAL-80 kan repræsentere.

For at undgå programstop for ikke-fatale fejl, findes to specielle instruktioner: 'TRAP ERR-' og 'TRAP ERR+'.

Hvis en 'TRAP ERR-' instruktion er blevet udført, standser en ikke-fatal fejl ikke programafviklingen, men tildeler sit fejlnummer til systemvariablen 'ERR'. Ved at teste denne variabel kan programafviklingen derefter påvirkes. Denne operationsmåde fortsætter til en 'TRAP ERR+' instruktion udføres, hvorefter systemet går tilbage til normal programafvikling.

Fatale fejl standser altid programafviklingen.

Bemærk, at denne måde at håndtere ikke-fatale fejl på er et spørgsmål om at have udført en 'TRAP ERR-' instruktion. Dens aktuelle linienummer er uden betydning.

'RUN' kommandoen bevirker altid, at COMAL-80 går tilbage til normal fejlhåndtering.

COMAL-80 tillader, at den samme streng af karakterer bruges som navn for forskellige variabeltyper, for procedurer og for funktioner. Da dette ikke må kunne misforstås, gælder følgende regler:

Statisk binding anvendes overalt. Det skal være klart udfra programteksten, hvad hvert navn betyder. Dette afspejler sig i adskillige af de følgende regler.

Alle variable, som første gang anvendes indeni en lukket procedure eller funktion, er lokale for denne procedure eller funktion og der kan ikke refereres til deres værdi indefra nogen anden procedure eller funktion eller fra hovedprogrammet. Dette gælder også for eventuelle parametre til en lukket procedure/funktion, da disse blot er specialtilfælde af lokale variable.

Indefra en lukket procedure/funktion kan der refereres til følgende variable: De lokale variable til proceduren/funktionen og til alle variable som udtrykkeligt er importeret fra hovedprogrammet under anvendelse af 'IMPORT' instruktionen. Der kan ikke refereres til andre variable indefra en lukket procedure/funktion.

Parametre til en åben procedure/funktion er lokale til proceduren/funktionen og der kan ikke refereres til dem indefra nogen anden procedure/funktion (heller ikke indefra en anden åben procedure/funktion) eller fra hovedprogrammet. En åben procedure/funktion kan ikke have andre lokale variable end eventuelle parametre.

Indefra en åben procedure/funktion kan der refereres til følgende variable: procedurens/funktionens parametre samt hovedprogrammets variable. Der kan ikke refereres til andre variable indefra en åben procedure/funktion. Hvis der er variable, som bliver 'DIM'ensioneret indeni den åbne procedure, behandles de i alle måder, som om de var 'DIM'ensioneret i hovedprogrammet. Variable, som bruges indeni en åben procedure/funktion, men ikke i hovedprogrammet, betragtes som hørende til hovedprogrammet.

'IMPORT' instruktionen kan kun anvendes indeni en lukket procedure/funktion. Den kan ikke anvendes indeni en åben procedure/funktion eller i hovedprogrammet.

Der kan refereres til system-variable og -funktioner overalt i et program.

Procedurer og funktioner kan ikke anbringes indeni hinanden, hvadenten de er åbne eller lukkede.

Procedurer og funktioner kan kalde hinanden eller sig selv (rekursion), hvadenten de er åbne eller lukkede.

Procedurer kan kaldes overalt i programmet. Procedurenavne kan ikke importeres.

Funktioner kan kaldes overalt i programmet, undtagen fra punkter hvor en variabel med det samme navn, skjuler funktionen. Dette kan kun være tilfældet, hvis funktionen har parametre. Regelen er: hvis der kan refereres til en 'DIM'ensioneret variabel med samme navn i et bestemt punkt, så kan funktionen ikke kaldes fra dette punkt. Ethvert forsøg på at kalde funktionen fra dette punkt, vil blive betragtet som en reference til variablen, hvor parametrene betragtes som indices.

Et kald af en funktion uden parametre udføres ved at skrive funktionsnavnet efterfulgt af tomme paranteser, f.eks. 'FUNKTION()'. Dette udelukker muligheden for misforståelser, selvom der kan refereres til en variabel med samme navn i det kaldende punkt.

Ovenstående regler er indført for at tillade tilfældig navngivning af variable i biblioteksrutiner (som altid bør være lukkede).

Funktionsnavne kan ikke importeres.

Labels, der er defineret i hovedprogrammmet, kan der kun refereres til fra hovedprogrammet. Labels, der er defineret i en procedure eller funktion og uanset om den er åben eller lukket, kan der kun refereres til i den pågældende procedure eller funktion. Labels kan ikke importeres.

Når COMAL-80 er indlæst i datamatens arbejdslager, og er begyndt at arbejde (men før promttegnet vises), er det ofte bekvemt at kunne ændre startværdier eller indlæse og starte afviklingen af et program automatisk.

Initialiseringsfilen giver denne mulighed. Før promttegnet vises, undersøges CP/M's standarddisk for en bestemt fil, som, hvis den eksisterer, indlæses og udføres.

Filen er en normal COMAL-80 tekstfil gemt på disketten under navnet COMAL80I.NIT (for 7 cifret versionen) og CMAL80DI.NIT (for 13 cifrer versionen). At filen er en tekstfil betyder, at hver enkelt linie (dog i dette tilfælde undtagen den første) følger syntaksen:

    (linie nr.) // (COMAL-80 instruktion eller kommando)

Filens første linie følger syntaksen:

    (linie nr.) // (fejltekst mode) [højeste hukommelse position)]

(linie nr.) er et normalt linienummer, som kun anvendes, når filen editeres.

Kommentartegnet ('//') gør det muligt at skrive hvadsomhelst på resten af linien.

(COMAL-80 instruktion eller kommando) er en normal instruktion eller kommando, som nærmere beskrevet i afsnit 2 i denne manual. De eneste undtagelser er, at 'AUTO' og 'EDIT' kommandoerne ikke kan anvendes.

(fejltekst mode) angiver, hvorvidt fejltekster ønskes eller ikke. Mulige svar er her 'Y' for ja, 'N' for nej og 'A' for spørg. Når 'A' benyttes, spørger COMAL-80 inden opstarten på skærmen om fejltekster ønskes.

(højeste hukommelse position) angiver i decimal den højeste hukommelseadresse, som COMAL-80 systemet må anvende. Denne specification er valgfri, og er den ikke angivet anvender COMAL-80 hele arbejdslageret under CP/M.

Når initialiseringsfilen udføres, indlæses den linie for linie fra disken. (linie nr.) og kommentartegnet bortkastes og resten af linien udføres nøjagtig som en normal COMAL-80 linie.

EKSEMPEL:

```
0010 // Y 50000
0020 // ZONE:=8; PAGEWIDTH:=132; PAGELENGTH:=0
0030 // CLEAR
0040 // PRINT "VELKOMMEN TIL COMAL-80"
```

Denne fil indeholder kun kommandoer. I linie 0010 får COMAL-80 systemet besked om, at fejltekster ønskes og at ingen hukommelsesposition højere end adresse 50000 decimalt må anvendes.

Resten af linierne indlæses og udføres en for en.

Initialiseringsfilen kan også indeholde instruktioner som vist i det følgende eksempel:

```
0010 // N
0020 // 10 // SKRIV 80 STJERNER
0030 // 20 FOR I=1 TO 80
0040 // 30 PRINT "*"
0050 // 40 NEXT I
0060 // RUN
```

I dette eksempel gemmes 'FOR...NEXT' sløjfen først i hukommelsen, styret af linienumrene 20, 30 og 40, og udføres så, når 'RUN' kommndoen mødes i linie 0060. Bemærk at instruktionerne virkelig gemmes i hukommelsen, og de vil stadig være der, når initialiseringsfilen er færdigafviklet, medmindre en 'NEW' kommando inkluderes.

Det er også muligt at indlæse og udføre en eller flere filer:

```
0010 // N
0020 // LOAD DK1:PROG1
0030 // RUN
0040 // NEW
0050 // ENTER PROG2
0060 // RUN
```

Dette eksempel indlæser først filen 'PROG1' fra drev 1, udfører programmet, sletter hukommelsen, indlæser derefter 'PROG2', som også udføres.

For at sikre, at hele initialisationsfilen udføres er 'ESC' tasten uvirksom indtil sidste linie nås. Dette betyder, at medens 'PROG1' i foranstående eksempel udføres, kan det ikke stoppes ved at trykke på 'ESC' tasten, hvorimod denne tast reagerer normalt, medens 'PROG2' udføres, fordi 'RUN' kommandoen er på sidste linie i initialisationsfilen.

Hver enkelt linie undersøges som normalt for fejl, men da 'ESC' tasten er frakoblet, er det kun muligt at rette den linie, der i fejltilfælde vises på skærmen. Dette betyder, at der er situationer, hvor en fejl kun kan forbigås ved at slette hele linien.

Extensions i COMAL-80 er en enestående ny facilitet, som gør det ligt for brugeren at udvide sproget ved at tilføje nye instruktioner, standardfunktioner og operatorer.

Nye nøgleord og den nødvendige Z-80 maskinkode opbevares i en diskfil, som dannes ved hjælp af en relokerende assembler. En eller flere af disse filer kan derefter aktiveres med 'EXTENSION' kommandoen, og de nye nøgleord bliver derved, i alle henseender, ligestillet med de originale nøgleord.

Tillæg E viser et fuldt arbejdende eksempel på denne mulighed og bør studeres i sammenhæng med denne beskrivelse.

Et assemblerprogram, der definerer extensions, skal begynde med

```
NAME('⟨navn⟩')
```

hvor ⟨navn⟩ er programmets navn.
Derefter skal indholdet af filen EXTDEFS.MAC komme efterfulgt af specifikationer for de extensions, der er defineret i filen.
Disse skal følge lige efter hinanden. Koden for dem kommer efter den sidste af dem. De ser hver sådan ud:

```
EXTENSION  ⟨navn⟩[,⟨lokalt navn⟩]
⟨grænsefladeangivelse⟩
ENDEXT     ⟨navn⟩[,⟨lokalt navn⟩]
```

Her skal angives samme ⟨navn⟩ og evt. ⟨lokalt navn⟩ begge steder. I den sidste specifikation skal ordet ENDEXT erstattes med ENDALLEXT. ⟨navn⟩ er det navn, extension'en skal kaldes ved i COMAL-programmer excl. eventuelt $-tegn. ⟨lokalt navn⟩ er det navn, extension'en har i assemblerfilen. Hvis det ikke er angivet, bruges ⟨navn⟩ i stedet. Angivelse af ⟨lokalt navn⟩ kan være nyttig, hvis flere extensions har navne, der falder sammen på de første 5 tegn, indeholder underscore eller falder sammen med Z80 opcodes.

⟨grænsefladeangivelse⟩ er forskellige for funktioner, sætninger og operatorer.

For funktioner:

```
FUNCTION  ⟨returtype⟩
⟨liste af parametre⟩
```

hvor ⟨returtype⟩ (funktionens type) er INT, REAL eller STR.

For sætninger:

STATEMENT
⟨liste af parametre⟩

⟨liste af parametre⟩ er nul eller flere linier, hvor hver linie specificerer een parameter på følgende måde:

PARAMETER [⟨dimension⟩,]⟨type⟩

Hvis ⟨dimension⟩, udelades, har man angivet en værdioverført parameter. Ellers referenceoverføres parameteren, og ⟨dimension⟩ angiver parameterens dimension. 0 = simpel variabel, 1=vektor osv. Her kan angives ANYDIM for ⟨dimension⟩. Det betyder, at den aktuelle parameter kan have en hvilkensomhelst dimension, og at den referenceoverføres. Det er da op til extension'en at finde ud af, hvilken dimension, parameteren har, og udnytte dette.

⟨type⟩ angiver parameterens type. Her kan angives INT, REAL, STR med den indlysende betydning. Desuden kan angives ANYTYPE eller INTREAL. ANYTYPE betyder at den aktuelle parameter kan have en hvilkensomhelst type (altså INT, REAL eller STR), og INTREAL betyder, at den aktuelle parameter kan have type INT eller REAL. Der gør sig nogle særlige forhold gældende i disse to tilfælde vedrørende den måde, hvorpå parameteren overføres til extension'en - se nedenfor.

For operatorer er ⟨grænsefladeangivelsen⟩ enten

OPERATOR ⟨returtype⟩,⟨venstre operands type⟩,⟨højre operands type⟩,⟨prioritet⟩

eller

OPERATOR ⟨returtype⟩,⟨operandens type⟩,⟨prioritet⟩

hvor det første benyttes for dyadiske operatorer og det andet benyttes for monadiske operatorer. ⟨returtype⟩ er som for funktioner enten INT, REAL eller STR . ⟨venstre operands type⟩, ⟨højre operands type⟩ og ⟨operandens type⟩ kan være INT, REAL, STR, ANYTYPE eller INTREAL med samme betydning, som ovenfor beskrevet ved funktioner og sætninger.
Parametre til operatorer kan kun være værdioverførte. ⟨prioritet⟩ angiver hvilken prioritet i beregningen, operatoren skal have i forhold til andre operatorer, herunder standardoperatorer. Det angives som navnet på en standardoperator, hvor det navn, som også benyttes ved kald af matematikpakken, skal bruges. Operatoren vil da få samme prioritet som denne standardoperator. Følgende navne kan bruges:

```
POWER,
TIMES, SLASH, DIV, MOD,
PLUS, MINUS, CHS,
LEQ, LSS, GEQ, GTR, EQL, NEQ, IN,
B.NOT,
B.AND,
B.OR
```

Efter linien med ENDALLEXT ..... kommer koden for de extensions, der er specificeret. Hver rutine begynder med en label, som er det navn (eller lokalt navn), der er angivet i den tilsvarende EXTENSION linie.

Når rutinen starter, er alle registre, som benyttes af COMAL, gemt og kan benyttes frit. Parametrene ligger på en stak, som udpeges af IX og som vokser nedad ligesom SP-stakken. Den sidste parameter ligger øverst.
For referenceoverførte parametre ligger et såkaldt referenceelement for parameteren på stakken. Det beskrives nedenfor. Bemærk, at hvis den formelle parameter er af type ANYTYPE eller REALINT, må extension'en selv finde ud af, hvad typen på den aktuelle parameter er, og indrette sig derefter. Typen fremgår af referenceelementet, se beskrivelsen nedenfor.
For værdioverførte parametre ligger værdien på stakken. Hvordan værdien er struktureret, beskrives nedenfor. Hvis den formelle parameter er af type ANYTYPE eller REALINT, er der stakket endnu en ekstra byte ovenpå værdien, og denne byte angiver værdiens type på følgende måde: For ANYTYPE kan byten have værdien INT, REAL eller STR, svarende til den aktuelle parameters type. For INTREAL vil en aktuel parameter af type REAL altid blive konverteret (med afrunding) til INT før extension'en kaldes, men den ekstra byte vil angive om konverteringen gav anledning til overløb (tallet for stort) Hvis dette var tilfældet, vil byten være <> 0; ellers er den =0. Hvis den aktuelle parameter er af type INT, vil byten være =0. I tilfælde af overløb, vil der stadig ligge et heltal på stakken lige under den omtalte byte, men dets værdi er uinteressant.

Desuden indeholder A- hhv. B-registeret den kørende COMAL-versions versionsnummer hhv. underversionsnummer. For version 2.0 er underversionsnummeret 0, og versionsnummeret er 8+DB+OV, hvor DB=0 for 7-ciffer-udgaven og DB=2 for 13-cifferudgaven, mens OV=0 for den hele udgave og OV=1 for overlayudgaven.

Inden rutinen returnerer, skal alle parametrene afstakkes fra IX-stakken. Hvis rutinen svarer til en funktion eller en operator, skal returværdien stakkes på IX-stakken inden returnhoppet.
Hvis der returneres med C-bitten sat (se nedenfor), er det dog ikke nødvendigt at afstakke parametre eller stakke nogen returværdi.
Der returneres v.hj.a. RET. Der skal ikke pilles ved SP-stakkens højde, men den kan godt benyttes, da der er ca. 100 bytes fri plads

Afhængigt af indholdet af AF-registret vil COMAL afgive en fejlmeldding når rutinen har returneret. Der er mulighed for at få både fatale og ikke-fatale fejlmeldinger.

Hvis Z-bitten er sat, vil der ikke blive givet nogen fejlmelding. Ellers vil der blive givet fejlmeldingen med det nummer, der ligger i A-registret, dog således at der skal fratrækkes 50 fra numre, der er større en 200. Hvis C-bitten er sat, vil fejlmeldingen være fatal; ellers vil den være ikke-fatal, dvs. den vil ikke komme, hvis man kører med TRAP ERR-, men i stedet kan man få fejlnummeret v.hj.a. ERR() .

Struktur af parametre

Nedenfor gennemgås den måde, hvorpå parametre, dvs. værdier og referenceelementer, repræsenteres af COMAL:

Heltal repræsenteres som datamaten gør det. Bitmønsteret 8000H betyder "udefineret" og vil ikke blive leveret som parameter. Det bør ej heller returneres som returværdi eller tildeles nogen parameter. Hvis en referenceoverført parameter har denne værdi, betyder det at variablen aldrig er blevet tildelt nogen værdi. Dette bør undersøges først hvis værdien benyttes. Dette gøres implicit af matematikpakkens funktion LDVAL.

Reelle tal fylder 4 bytes for 7-cifferversionen, og 8 bytes for 13-cifferversionen. Første byte = 80H og sidste byte lig 00H betyder "udefineret". Bemærkningerne ovenfor gælder også her.

Strenge består af to dele: først 2 bytes (et heltal) der fortæller hvor lang strengen er, og dernæst, i rækkefølge mod højere adresser, det således angivne antal bytes, som hver indeholder et tegn, således, at den byte, der følger lige efter længden, indeholder det første tegn, den næste det andet tegn osv.

Referenceelementer angiver variable af alle slags, både simple variable, herunder strengvariable, arrays og delstrenge. Formatet for referencelementer for delstrenge er specielt og beskrives senere. Alle andre referenceelementer fylder 4 bytes, og består af to pointere: først en pointer til en beskrivelse af variablen, og derefter en pointer til variablens dataområde.

Variabelbeskrivelsen er af variabel størrelse og vokser mod lavere adresser (NB!). For simple variable består den af en byte med værdi -INT, -REAL eller -STR, svarende til de tre typer. Hvis det er -STR følges denne byte af strengvariablens maksimale længde (et heltal i 2 bytes).

For arrays er der først et antal index-felter, svarende til antallet af indices. Hvert indexfelt består af følgende:

antal indexfelter efter dette	(1 byte)
lav grænse for dette index	(heltal 2 bytes)
høj grænse for dette index	(heltal 2 bytes)
størrelse af hver delarrays eller elements data	(heltal 2 bytes)

Eksempel:
Variabelbeskrivelsen for en array dimensioneret med

```
DIM A$(-2:5, 1:10) OF 20
```

er:

```
1     (1 byte)
-2    (2 bytes)
5     (2 bytes)
220   (2 bytes)

0     (1 byte)
1     (2 bytes)
10    (2 bytes)
22    (2 bytes)  (2 for aktuel længde + 20 til selve tegnene)

-STR  (1 byte)
20    (2 bytes)
```

Referenceelementer for delstrenge begynder også med en pointer til en variabelbeskrivelse, der i dette tilfælde er en enkelt byte med værdien enten '-STR-1' eller '-STR-2'.

Derefter følger en pointer til datafeltet for den strengvariabel, delstrengen er en del af. Efter denne følger:

den maksimale (dimensionerede) længde for den pågældende strengvariabel,
2. index (til-værdi), og
1. index (fra-værdi),

som er anvendt ved specifikationen af delstrengen.

Eksempler på 2. og 1. index: ved A$(8:17) er 2.index = 17 og 1.INDEX = 8. Ved A$(10) er både 2.index og 1.index = 10. Alle tre værdier er heltal (2 bytes).

Datafeltet for en simple variabel er struktureret som en værdi, se ovenfor. Datafeltet for en array er blot en række simple datafelter efter hinanden, et for hvert af elementerne. For strengarrays er der afsat plads til at hvert element kan vokse til sin maksimale længde.

Matematikpakken

I EXTDEFS.MAC er defineret alt, hvad der skal bruges for at man kan benytte matematikpakken. Man kalder matematikpakken ved at benytte makroen

```
EXPR
```

hvorefter følger kommandoer til matematikpakken. Der afsluttes med

```
EXPREND
```

Kommandoerne udføres en efter en i den rækkefølge de står. De tager (for det meste) deres argumenter fra IX-stakken og gemmer deres resultater på denne stak. Man kan opfatte kommandoerne som et program til en stakorienteret maskine. Kommandoerne er i virkeligheden makroer, som er defineret i EXTDEFS.MAC. Hver makro expanderer til en eller flere bytes med parametre til kaldet af matematikpakken, som fortolker disse parametre og opererer på IX-stakken.

Man kan benytte følgende kommandoer:

De følgende kommandoer svarer til COMALs standardfunktioner, men alle kommandoer kræver argumenter af en bestemt type, og leverer et et resultat af en bestemt type, som angivet. Hvor der er angivet INTREAL, betyder dette, at argumentet skal være af heltalstype, men at det er i orden, hvis dette er fremkommet ved at kald af REALINT umiddelbart før kommandoen.

navn	argument(ers) type	resultats type	
ATN	REAL	REAL	
COS	REAL	REAL	
SIN	REAL	REAL	
TAN	REAL	REAL	
LOG	REAL	REAL	
EXP	REAL	REAL	
SQR	REAL	REAL	
ESC	-	INT	
ERR	-	INT	
EOD	-	INT	
EOF	REALINT	INT	
LEN	ref. elem.	INT	(ref.elem. skal være streng)
ORD	STR	INT	
IVAL	STR	INT	
VAL	STR	REAL	
INT	REAL	REAL	
FRAC	REAL	REAL	
TRUNC	REAL	INT	
ROUND	REAL	INT	
POS	STR, STR	INT	
BVAL	STR	INT	
CHR	REALINT	STR	
STR	REAL	STR	(svarer begge til STR$, men
I.STR	INT	STR	taget af reelt tal hhv. heltal)

ERRTEXT	REALINT	STR	
SGN	REAL	INT	(svarer begge til SGN, men
I.SGN	INT	INT	taget af reelt tal hhv. heltal)
ABS	REAL	REAL	(svarer begge til ABS, men
I.ABS	INT	INT	taget af reelt tal hhv. heltal)
RND0	-	REAL	(RND uden parametre)
RND2	REALINT, REALINT	INT	(RND med 2 parametre)
SPC	REALINT	STR	
PEEK	REAL	INT	
INP	REALINT	INT	
BSTR	REALINT	STR	
VARPTR	ref.elem.	REAL	
FREEST	-	INT	(FREESTORE)

Diverse konverteringer:

CONV konverterer det øverste stakelement fra INT til REAL.

CONV1 konverterer det næstøverste stakelement fra INT til REAL. Det forudsættes at det øverste stakelement er af type REAL.

REALINT konverterer det øverste stakelement fra REAL til INT med afrunding. Hvis det tallet er udenfor heltalsområdet, bliver det øverste stakelement udefineret, men der sættes desuden et særligt overløbsflag, som de standardfunktioner, der ovenfor er angivet REALINT ved, reagerer passende på.

RLBL konverterer det øverste stakelement fra REAL til INT, således, at hvis tallet er <> 0 bliver resultatet 1, ellers 0. Benyttes i forbindelse med de boolske operatorer.

RLBL1 konverterer det næstøverste stakelement fra REAL til INT på samme måde som RLBL. Det forudsættes at det øverste stakelement er af type INT.

COMALs standardoperatorer har følgende navne i denne sammenhæng:

navn	COMALs navn	argument(er)s type	resultats type
CHS	- (monadisk)	REAL	REAL
POWER	^	)	
TIMES	*	)	
SLASH	/	)	
DIV	DIV	)-) REAL, REAL	REAL
MOD	MOD	)	
PLUS	+	)	
MINUS	- (dyadisk)	)	

LEQ	<=	)	
LSS	<	)	
GEQ	>=	)->REAL, REAL	INT
GTR	>	)	(værdi 0 eller 1)
EQL	=	)	
NEQ	<>	)	
I.CHS	-	INT	INT
I.TIMES	*	)	
I.DIV	DIV	)	
I.MOD	MOD	)->INT, INT	INT
I.PLUS	+	)	
I.MINUS	-	)	
I.LEQ	<=	)	
I.LSS	<	)	
I.GEQ	>=	)->INT, INT	INT
I.GTR	>	)	(værdi 0 eller 1)
I.EQL	=	)	
I.NEQ	<>	)	
S.PLUS	+	STR, STR	STR
S.LEQ	<=	)	
S.LSS	<	)	
S.GEQ	>=	)->STR, STR	INT
S.GTR	>	)	(værdi 0 eller 1)
S.EQL	=	)	
S.NEQ	<>	)	
IN	IN	STR, STR	)
B.AND	AND	INT, INT	)->INT
B.OR	OR	INT, INT	) (værdi 0 eller 1)
B.NOT	NOT	INT	)

Herudover er der følgende kommandoer:

INX forventer et tal og dernæst et referenceelement for en array eller en strengvariabel på stakkens top. Foretager indicering, dvs. afstakker disse to og stakker i stedet et referenceelement for det valgte element i den angivne array eller den angivne delstreng.
Tallet kan være et heltal eller et reelt tal. I det sidste tilfælde konverteres tallet automatisk til heltal med afrunding. Tallets type (INT eller REAL) skal angives efter INX f.eks. sådan:

```
INX  REAL
```

LDVAL forventer et referenceelement på stakkens top. Afstakker dette og stakker i stedet værdien af den tilsvarende variabel. Kan også bruges efter INX. Det checkes at variabelen ikke er "udefineret", dvs. har den specielle værdi, der betyder, at der ikke er blevet tildelt nogen værdi til den.

STVAL forventer et referenceelement og dernæst en værdi på stakkens top. Begge afstakkes og værdien gemmes i den variabel, der er angivet af referenceelementet. Værdiens type og typen af den variabel, der angives af referenceelementet, skal stemme overens. Kan benyttes efter INX.

LOAD forventer en adresse (2 bytes) på stakkens top. Afstakker denne og stakker i stedet den værdi/det referenceelement, der ligger på den adresse. Det skal angives, hvilken type, værdien har, eller at det er et referenceelement. Dette gøres på samme linie efter LOAD, f.eks. således:

```
LOAD  INT
```

STORE forventer en adresse (2 bytes) og dernæst en værdi eller et referenceelement på stakkens top. Begge afstakkes og værdien/referencelementet gemmes på den angivne adresse. Det skal, ligesom ved LOAD, angives hvilken type, værdien har, eller at det er et referenceelement. Dette gøres ligesom ved LOAD, f.eks.:

```
STORE  REF
```

INTCON stakker et heltal eller en adresse (2 bytes) på stakken. Heltallet/adressen skal angives på den samme linie efter INTCON. f.eks. således:

```
INTCON   47
```

STRCON stakker en strengværdi på stakken. Strengværdien skal angives på den samme linie efter STRCON, f.eks. således:

```
STRCON  'Strengen skal i enkelt anførselstegn'
```

UROUND forventer et reelt tal på stakken. Dette afstakkes, konverteres til et fortegnsløst heltal i området 0 til 65535 og stakkes.

SYSVAR stakker værdien af en af systemvariablene som et heltal. Det skal angives hvilken af systemvariablene, man er interesseret i; dette gøres på samme linie efter SYSVAR, f.eks. således:

```
SYSVAR  PAGEWI
```

Systemvariablenes navne er lidt forkortede her:

COMALs navn	Her kaldes den
ZONE	ZONE
INDENTION	INDENT
PAGEWIDTH	PAGEWI
PAGELENGTH	PAGELE
KEYWORDLOWER	KWLOWER
IDENTIFIERLOWER	IDLOWER

TRUE stakker et heltal 1

FALSE stakker et heltal 0

Der er naturligvis mulighed for at der opstår fejl ved brug af matematikpakken: overløb, indexfejl, forskellige funktioner får argumenter, der er ude af definitionsområdet osv. Der skelnes mellem fatale og ikke-fatale fejl. Fejl, der er fatale under normal COMAL-afvikling, er fatale her; fejl, der ikke er fatale under normal COMAL-afvikling, dvs. kan slås fra v.hj.a. TRAP ERR-, er ikke fatale her. Hvis en ikke-fatal fejl opstår, gemmes dens fejlnummer blot og der fortsættes med udførelse af kommandoer. Fejlnummeret vil da stå i A-registret, når der vendes tilbage til extension'en efter ENDEXPR, og carry-bitten vil være 0, mens Z=1. Hvis der opstår en fatal fejl, vil resten af kommandoerne blive sprunget over, IX sættes tilbage til dens værdi ved indgangen til matematikpakken, og der vendes tilbage til extension'en efter ENDEXPR med fejlnummeret i A-registeret, carry=1 og Z=1. Hvis der ikke opstår nogen fejl overhovedet, er A=0 og Z=0 når der vendes tilbage til extension'en efter ENDEXPR. Ønsker man således at checke om der opstår en fejl ved udførelsen af en bestemt kommando, kan man blot indføje

```
ENDEXPR
JP      NZ,FEJLBEHANDLING
EXPR
```

efter denne kommando.

Bemærk, at disse konventioner for indholdet af A, CY, Z svarer til konventionerne for rapportering af fejl tilbage til det kaldende COMAL-program.

Format for .EXT -filer

Den fil, der læses op af COMAL når man bruger EXTENSION-kommandoen, skal have et bestemt format. Dette er et særlig simpelt relokerbart format. Assemblerprogrammet, hvis form er beskrevet i den første afsnit kan oversættes af Microsofts M80 makroassembler til eet relokerbart format. Andre assemblere benytter andre relokerbare formater (og nogle af dem vil også kræve at definitioner i EXTDEFS.MAC omskrives. COMALen definerer sit eget relokerbare format, som er som følger:

Filen starter med et heltal på 2 bytes, som fortæller, hvor meget kode, filen indeholder, talt i bytes. Dette er ikke det samme som filens længde, da filen udover koden indeholder information om, hvilke dele af koden, der skal relokeres.

Resten af filen består af et antal blokke med samme format. Hver blok består af 9 bytes, hvoraf de 8 indeholder kode, og den niende indeholder relokeringsinformation for de 8 bytes. Hvis det tal, filen starter med, er L, vil der være (L+7) DIV 8 blokke. Hvis L ikke er et multiplum af 8, er der overskydende plads i den sidste blok. Denne overskydende plads indeholder blot noget nonsens.

Koden, som er indeholdt i blokkene, udgør logisk een lang følge af kodebytes. Koden for to blokke efter hinanden vil altså ligge lige efter hinanden når filen er blevet læst op af COMALen. Byten med relokeringsinformation indeholder een bit for hver af de 8 kodebytes, således at bit 0 svarer til den første byte, osv. op til bit 7, der svarer til den ottende og sidste kodebyte. Hvis en sådan bit er 1, betyder det, at den adresse, som ligger i den pågældende byte og den foregående, skal relokeres. Hvis bitten er 0, skal de to byte ikke relokeres.

Programmet CONVERT kan lave en relokerbar fil i Microsoft format om til den tilsvarende fil i COMALs format. Benytter man en assembler, der bruger et andet format, må man selv lave et program, der kan lave den relokerbare fil om til COMALs format.

Alle COMAL-80 kommandoer, instruktioner og funktioner er beskrevet i dette kapitel. Hver beskrivelse er opstillet således:

Type:	Angiver, om det er en kommando, instruktion eller funktion.
Formål:	Beskriver, hvortil nøgleordet anvendes.
Syntaks:	Viser den korrekte syntaks i forbindelse med nøgleordet. Den anvendte notation er beskrevet nedenfor.
Udførelse:	Beskriver, hvorledes nøgleordet udføres.
Eksempel:	Programmer eller programdele, som viser, hvorledes nøgleordet anvendes.
Kommentarer:	Beskriver detaljer for nøgleordets anvendelse.

Syntaks notation.

Overalt hvor syntaksen for en kommando, instruktion eller funktion vises, gælder følgende regler:

Ord, skrevet med store bogstaver, skal indtastes som vist; men både store og små bogstaver kan anvendes.

Ord eller begreber, skrevet med små bogstaver, og omgivet af vinkler (< >), indsættes af brugeren.

Ord i firkantparanteser ([ ]) er valgfrie.

Alle tegn, undtagen vinkler og firkantparanteser, (dvs. komma, parenteser, semikolon, kolon, udråbstegn, divisionstegn, nummertegn, plus tegn, minus tegn eller lighedstegn) skal indtastes hvor de er vist.

Alle nøgleord skal omgives af mellemrum, hvor dette er nødvendigt for at opnå entydig oversættelse.

Type:
Aritmetisk funktion

Formål:
At beregne den absolutte værdi af et aritmetisk udtryk.

Syntaks:
ABS((udtryk))

Udførelse:
Den absolutte værdi af (udtryk) beregnes.

Eksempel:
```
10 PRINT ABS(3*(-5))
```

Kommentarer:
1. (udtryk) skal være aritmetisk og af reel eller heltallig type. Resultatet bliver samme type.

Type:

Logisk operator

Formål:

At danne logisk 'og' mellem to udtryk.

Syntaks:

(udtryk1) AND (udtryk2)

Udførelse:

(udtryk1) og (udtryk2) beregnes, hvorefter logisk 'og' dannes.

Eksempel:

```
10 INPUT A#
20 INPUT B#
30 IF A#=5 AND B#=7 THEN
40  PRINT "PRODUKTET ER 35"
50 ELSE
60  PRINT "PRODUKTET ER MASKE IKKE 35"
70 ENDIF
```

Kommentarer:

1. Operatoren har følgende sandhedstabel:

(udtryk1)	(udtryk2)	resultat
sand	sand	sand
sand	falsk	falsk
falsk	sand	falsk
falsk	falsk	falsk

Type:

Aritmetisk funktion

Formål:

At beregne arctangens af et numerisk udtryk.

Syntaks:

ATN(⟨udtryk⟩)

Udførelse:

Arctangens af ⟨udtryk⟩, som er i radianer, beregnes.

Eksempel:

```
10 INPUT A
20 PRINT ATN(A)
```

Kommentarer:

1. ⟨udtryk⟩ er et aritmetisk udtryk af reel eller heltallig type. Resultatet er altid reelt og beliggende i intervallet -pi/2 til pi/2.

Type:

Kommando

Formål:

Under indtastningen af et program, beregner 'AUTO' automatisk et nyt linienummer efter hver vognretur.

Syntaks:

AUTO [<start>[,<spring>]]

Udførelse:

Efter hver vognretur udregnes et nyt linienummer som summen af det sidst benyttede linienummer (eller værdien angivet i start) og det angivne spring. Den fundne værdi anbringes i inputbufferen og vises på skærmen.
Cursoren anbringes i position 6 plus den gældende linieindrykning (som er programafhængig) klar til indtastning af programlinien.

Eksempler:

AUTO
AUTO 15
AUTO 10,5

Kommentarer:

1. Udelades værdien for <start>, benyttes værdien 10.

2. Udelades værdien for <spring>, benyttes værdien 10.

3. Hvis der genereres et linienummer, som allerede findes, erstatter den nye linie den gamle.

4. Den automatiske linienummergenerering kan til enhver tid afbrydes ved tryk på 'ESC' tasten.
Den linie, hvori dette sker, gemmes ikke.

Type:

Strengfunktion

Formål:

At oversætte et aritmetisk udtryk til binær repræsentation.

Syntaks:

BSTR$((udtryk))

Udførelse:

(udtryk), som skal være aritmetisk, beregnes og afrundes om nødvendigt. Værdien oversættes derefter til en binær tegnstreng på nøjagtig 8 pladser.

Eksempel:

```
10 DIM A$ OF 8
20 INPUT B
30 A$:=BSTR$(B)
40 PRINT A$
```

Kommentarer:

1. (udtryk) skal have en værdi i det lukkede interval 0 til 255.

Type:

Aritmetisk funktion

Formål:

At omsætte et binært tal, som forefindes som en tegnstreng, til et heltal.

Syntaks:

BVAL((strengudtryk))

Udførelse:

Det binære tal, som forefindes som en tegnstreng på nøjagtig 8 karakterer, oversættes til et heltal.

Eksempel:

```
10 DIM A$ OF 8
20 INPUT "SKRIV ET BINÆRT TAL PÅ 8 CIFRE: ": A$
30 PRINT BVAL(A$)
```

Kommentarer:

1. Hvis tegnstrengen indeholder mere end eller mindre end 8 tegn, eller hvis den indeholder andet end binære cifre (0 eller 1), standses programafviklingen med en fejlmelding.

Type:

Instruktion, kommando

Formål:

Med 'CALL' kan maskinsprogsprogrammer for Z-80 mikroprocessoren knyttes til et COMAL-80 program.

Syntaks:

CALL (udtryk)

Udførelse:

(udtryk), som skal være aritmetisk, beregnes og afrundes om nødvendigt. CPUen gemmer derefter sine registre og fortsætter programafviklingen i den beregnede adresse.

Eksempler:

```
      CALL 256
240 CALL 53248
```

Kommentarer:

1. For en nærmere beskrivelse af Z-80 mikroprocessoren og dens maskinsprog henvises til producenternes manualer.

2. Brugeren kan frit benytte CPUens registre, dog skal stackpointeren være retableret inden returnering til COMAL-80.

3. COMAL-80 benytter ikke CPUens interruptfaciliteter. Disse kan derfor frit udnyttes af brugeren, også efter at man er vendt tilbage til COMAL-80.

4. Returnering til COMAL-80 sker ved at afslutte maskinsprogsprogrammet med en 'RET' instruktion.

Type:

Instruktion

Formål:

'CASE' strukturen anvendes, når man, på grundlag af et udtryks værdi, skal vælge mellem forskellige programdele.

Syntaks:

```
CASE (udtryk) OF
WHEN (liste af muligheder)
.
.
WHEN (liste af muligheder)
.
.
WHEN (liste af muligheder)
.
[OTHERWISE
.
.]
ENDCASE
```

Udførelse:

(udtryk) beregnes og 'WHEN' instruktionerne gennemløbes en for en for at undersøge, om en af de her angivne muligheder passer med den beregnede værdi. Er dette tilfældet, udføres de umiddelbart herefter følgende linier til næste 'WHEN', 'OTHERWISE' eller 'ENDCASE' instruktion, hvorefter programafviklingen fortsætter efter 'ENDCASE' instruktionen, forudsat at ingen af de udførte linier har overført programafviklingen til en anden del af programmet.
Passer ingen af de angivne muligheder med værdien for (udtryk), udføres i stedet linierne umiddelbart efter 'OTHERWISE' instruktionen.
Hvis 'OTHERWISE' er udeladt standser programafviklingen i dette tilfælde med en fejlmelding.

Eksempel:

```
 10 DIM A$ OF 1
 20 INPUT "TRYK PÅ + ELLER - TASTEN": A$
 30 CASE A$ OF
 40 WHEN "+"
 50  PRINT "DU HAR TRYKKET PÅ + TASTEN"
 60 WHEN "-"
 70  PRINT "DU HAR TRYKKET PÅ - TASTEN"
 80 OTHERWISE
 90  GOTO 20
100 ENDCASE
```

Kommentarer:

1. Udtrykkene indeholdt i 'WHEN' instruktionerne skal være af samme type som (udtryk). Dog er et heltalsudtryk tilladt i 'WHEN' instruktionerne, hvis (udtryk) er af reel type.

2. Hvis flere 'WHEN' instruktioner indeholder muligheder, som passer med (udtryk), udføres kun den programdel, som svarer til den første af disse.

Type:

Kommando

Formål:

At udskrive katalogblokken for et tilsluttet baggrundslager.

Syntaks:

```
CAT [<filnavn1>[,<filnavn2>]]
CAT <filnavn2>
```

Udførelse:

Datamatens operativsystem kaldes med angivelse af, for hvilken enhed katalogblokken ønskes udskrevet.
De aktuelle filer udskrives derefter på den ved <filnavn2> angivne enhed.

Eksempler:

```
CAT
CAT DK1:
CAT DK1:K
CAT DK1:,DK0:ABC.DEF
CAT *.CML,LP:
CAT DK1:C??????Y.*,LP:
CAT LP:
```

Kommentarer:

1. <filnavn2> er navnet på den fil, hvori katalogblokken ønskes udskrevet.

2. <filnavn1> angiver, helt eller delvist, de filnavne fra kataloget, som skal indeholdes i udskriften.
   En delvis angivelse kan bestå af kun et enhedsnavn, i hvilket tilfælde alle filnavne på pågældende enhed udskrives, eller af en filangivelse, hvor karaktererne '*' og '?' er anvendt i overensstemmelse med specifikationerne for CP/M.

3. Hvis <filnavn2> udelades, udskrives katalogblokken på den tilsluttede terminal.

4. Udelades <filnavn1> udskrives hele kataloget for den aktuelle standardenhed.

Type:

Instruktion

Formål:

At udskrive et baggrundslagers katalogblok i en fil.

Syntaks:

CAT (filnavn), FILE (filnummer)

Udførelse:

Datamatens operativsystem kaldes med oplysninger om, til hvilken enhed, hvilke filnavne skal udskrives.
Katalogets indhold udskrives derefter på ASCII form på det angivne (filnummer).

Eksempler:

```
100 CAT "DK1:", FILE 3
100 CAT "DK1:*.CML", FILE 2
```

Kommentarer:

1. (filnavn) er et strengudtryk.

2. (filnavn) angiver de ønskede filer fra et katalog.

3. (filnavn) angiver, helt eller delvist, de filnavne fra kataloget som skal indeholdes i udskriften.
   En delvis angivelse kan bestå af kun et ennedsnavn, i hvilket tilfælde alle filnavne på pågældende enned udskrives, eller af en filangivelse, hvor karaktererne '*' og '?' er anvendt i overensstemmelse med specifikationerne for CP/M.

4. Udelades (filnavn) udskrives hele katalogblokken for den aktuelle standardenhed.

5. Inden 'CAT' instruktionen mødes, skal en fil med det pågældende (filnummer) være åbnet med en 'OPEN' instruktion.

6. Enheden, hvorpå kataloget skal udskrives, angives i den tilhørende 'OPEN' instruktion.

7. Efter lukning og genåbning kan den dannede fil læses med 'INPUT FILE' instruktionen.

8. Hvis der til udskriften anvendes en linieprinter med 'PAGEWITH' = 0, eller en diskfil, skrives et filnavn på hver linie.

9. '#' og 'FILE' har samme betydning og kan anvendes i flæng, men i programlistninger konverteres '#' til 'FILE'.

Type:

Instruktion

Formål:

Fra et program at indlæse og igangsætte udførelsen af et andet program, som er gemt i binært format på baggrundslageret.

Syntaks:

CHAIN ⟨filnavn⟩ [,⟨liste af variable⟩]

Udførelse:

Datamatens arbejdslager slettes, det i ⟨filnavn⟩ angivne program indlæses, og programafviklingen fortsætter i dette programs laveste linienummer.

Eksempel:

```
 10 //HOVEDPROGRAM
 20 DIM PROGRAM$ OF 10
 30 REPEAT
 40  INPUT "HVILKET PROGRAM ØNSKES? ":PROGRAM$
 50 UNTIL PROGRAM$="UDSKRIFT" OR "INDLÆS"
 60 CASE PROGRAM$ OF
 70 WHEN "UDSKRIFT"
 80  CHAIN "DK0:UDSKRIFT"
 90 WHEN "INDLÆS"
100  CHAIN "DK1:INDLÆS"
110 ENDCASE
```

Kommentarer:

1. ⟨filnavn⟩ er et strengudtryk.

2. Hvis 'CHAIN' instruktionen inkluderer en ⟨liste af variable⟩, skal det nye program indeholde en 'RECEIVE ⟨liste af variable⟩' instruktion.

3. Sætningen anvendes typisk til at opdele et stort program i mindre, uafhængige dele, som indlæses og udføres på grundlag af brugerkommandoer.

4. Programmet ⟨filnavn⟩ skal være af binær type, dvs. være lagret ved hjælp af 'SAVE' instruktionen.

5. Se også 'RECEIVE' instruktionen.

Type:

Strengfunktion

Formål:

At omsætte et aritmetisk udtryk til en enkelt ASCII karakter.

Syntaks:

CHR$((udtryk))

Udførelse:

(udtryk), som skal være aritmetisk, beregnes og afrundes om nødvendigt. Værdien oversættes derefter til den hertil svarende karakter.

Eksempel:

```
10 INPUT A
20 PRINT CHR$(A)
```

Kommentarer:

1. (udtryk) skal have en værdi i det lukkede interval 0 til 255.

Type:

Instruktion, kommando

Formål:

At slette udskriften på dataskærmen og stille cursoren i øverste venstre hjørne.

Syntaks:

CLEAR

Udførelse:

Skærmteksten slettes og cursoren flyttes op i øverste venstre hjørne.

Eksempler:

```
10 CLEAR
   CLEAR
```

Kommentarer:

1. Denne instruktion/kommando påvirker kun skærmen.

Type:

Instruktion, kommando

Formål:

At lukke en eller flere datafiler efter benyttelsen.

Syntaks:

CLOSE [FILE ⟨filnummer⟩]

Udførelse:

⟨filnummer⟩, som skal være aritmetisk, beregnes og afrundes om nødvendigt. Filen med dette nummer lukkes derefter.

Eksempler:

```
200 CLOSE
390 CLOSE FILE 3
540 CLOSE FILE A*B
    CLOSE
```

Kommentarer:

1. Udelades ordet 'FILE' og ⟨filnummer⟩ lukkes alle åbne datafiler.

2. Efter at 'CLOSE' er udført, er den ved 'OPEN' instruktionen angivne sammenhæng mellem ⟨filnavn⟩ og ⟨filnummer⟩ opløst. Filen kan derefter genåbnes med det samme eller et andet ⟨filnummer⟩.

3. Man bør sikre sig, at 'CLOSE' instruktionen er udført inden arbejdet med et program afsluttes, idet man ellers risikerer, at der ligger data i systemets buffere. Kommandoen 'RELEASE', der også kan bruges som instruktion, fortæller, hvorvidt dette er tilfældet.

4. '#' og 'FILE' har sammme betydning og kan anvendes i flæng, men i programlistninger konverteres '#' til 'FILE'.

Type:

Kommando

Formål:

At genoptage programafviklingen efter stop.

Syntaks:

CON [<linienummer>]

Udførelse:

Programafviklingen fortsættes, enten i det angivne linienummer, eller, hvis dette mangler, efter den sidst udførte instruktion.

Eksempler:

```
CON
CON 220
```

Kommentarer:

1. Variable kan tildeles en ny værdi inden programafviklingen genoptages.

2. Programafviklingen kan genoptages efter stop forårsaget af en 'STOP' eller 'END' instruktion, efter tryk på 'ESC' tasten eller efter en ikke-fatal fejl.

3. Hvis der editeres i programmet, kan programafviklingen i visse tilfælde ikke genoptages.

4. Hvis programafviklingen stoppes ved tryk på 'ESC' tasten, medens datamaten venter i en 'INPUT' instruktion, får den pågældende variabel ikke tildelt nogen værdi. Programafviklingen bør derfor i dette tilfælde genoptages ved CON <linienummer>, hvor <linienummer> vises på skærmen, umiddelbart efter at 'ESC' tasten er nedtrykket

Type:

Trigonometrisk funktion

Formål:

At beregne cosinus af et aritmetisk udtryk.

Syntaks:

COS((udtryk))

Udførelse:

Cosinus til (udtryk), som er i radianer, beregnes.

Eksempel:

```
10 INPUT A
20 PRINT COS(A)
```

Kommentarer:

1. (udtryk) er aritmetisk og af reel eller heltallig type. Resultatet er altid reelt.

Type:

Instruktion, kommando

Formål:

At styre cursorens placering på dataskærmen.

Syntaks:

CURSOR ⟨udtryk1⟩, ⟨udtryk2⟩

Udførelse:

⟨udtryk1⟩ og ⟨udtryk2⟩, som begge skal være aritmetiske, beregnes og afrundes til heltal. Cursoren flyttes derefter til den karakterposition, som er angivet ved ⟨udtryk1⟩ og det linienummer, som er angivet ved ⟨udtryk2⟩.

Eksempler:

```
100 CURSOR 8,12
220 CURSOR KARAKTER#, LINIE
300 CURSOR 3*2, 5+4
    CURSOR 10, 15
```

Kommentarer:

1. ⟨udtryk1⟩ regnes positiv fra venstre mod højre, og ⟨udtryk2⟩ regnes positiv fra oven og nedefter.
   Øverste venstre hjørne har derfor betegnelsen 1,1.

Type:

Instruktion

Formål:

At definere konstanter i form af en dataliste, som indlæses af en 'READ' instruktion.

Syntaks:

DATA (konstant1),(konstant2),....,(konstantn)

Udførelse:

Ved programmets start gennemsøges dette for data instruktioner, som kædes sammen i en dataliste. Under programafviklingen udpeger en intern pointer hele tiden den næste konstant i denne liste.

Eksempel:

```
10 DIM FORNAVN$ OF 10
20 DIM EFTERNAVN$ OF 15
30 DATA "OLE", "JENSEN"
40 READ FORNAVN$
50 PRINT FORNAVN$,
60 DATA 35
70 READ EFTERNAVN$, ALDER
80 PRINT EFTERNAVN$, "ALDER ", ALDER
```

Kommentarer:

1. 'DATA' instruktioner er ikke udførbare og overspringes under programafviklingen.

2. Et vilkårligt antal 'DATA' instruktioner kan placeres hvorsomhelst i programmet.

3. En 'DATA' instruktion kan indeholde lige så mange konstanter (adskildt af komma), som den maksimale inputlinielængde (=159 karakterer) tillader.

4. 'READ' instruktionen læser 'DATA' instruktionerne i linienummerorden.

5. Konstanternes type kan blandes efter ønske, men skal svare til typerne anvendt i tilhørende 'READ' instruktioner, idet programafviklingen ellers standser med en fejlmelding.
   Udtryk er ikke tilladt blandt konstanterne, og strengkonstanter skal være omgivet af anførselstegn.

6. Konstanterne kan genlæses, helt eller delvist, ved hjælp af 'RESTORE', 'RESTORE (linienummer)' eller 'RESTORE' (navn)'.

7. Når sidste konstant er læst, tildeles systemvariablen 'EOD' værdien sand (= 1).

Type:

Kommando

Formål:

At fjerne en eller flere linier fra et program.

Syntaks:

DEL ⟨startlinie⟩[,⟨slutlinie⟩]
DEL ,⟨slutlinie⟩
DEL ⟨startlinie⟩

Udførelse:

Det angivne område slettes i programmet.

Eksempler:

```
DEL 25,100
DEL ,220
DEL 95,
DEL 40
```

Kommentarer:

1. Hvis kun ⟨startlinie⟩ angives, slettes kun denne linie.

2. Hvis ⟨startlinie⟩ efterfulgt af komma angives, slettes denne linie og resten af programmet.

3. Hvis kun komma efterfulgt af ⟨slutlinie⟩ angives, slettes begyndelsen af programmet til og med den angivne linie.

4. Hvis både ⟨startlinie⟩ og ⟨slutlinie⟩ angives, fjernes fra og med ⟨startlinie⟩ til og med ⟨slutlinie⟩.

Type:

Instruktion, kommando

Formål:

At slette filer i baggrundslageret.

Syntaks:

DELETE ‹filnavn›

Udførelse:

Datamatens operativsystem kaldes med information om, hvilken fil, der skal slettes.

Eksempel:

```
100 DELETE "TEST.CML"
220 DELETE "DK1:DATA.DAT"
300 DELETE "DK0:D??????P.*"
    DELETE PROGRAM.CML
    DELETE C*.CML
```

Kommentarer:

1. I instruktioner er ‹filnavn› et strengudtryk.

2. ‹filnavn› angiver helt eller delvist den fil/de filer, der skal slettes, idet karaktererne '*' og '?' kan anvendes i overensstemmelse med specifikationerne for CP/M

3. Hele filnavnet, inklusive eventuel ekstension, skal angives.

4. Hvis ‹filnavn› ikke eksisterer, gives for kommandoer en fejlmelding, medens dette ikke sker for instruktioner.

Type:

Instruktion

Formål:

At reservere hukommelsesplads for matricer samt at angive grænserne for indices.

Syntaks:

DIM (liste af indicerede variable)

Udførelse:

Under hensyn til variablens type udregnes og afsættes det nødvendige hukommelsesareal.

Eksempler:

```
10 DIM ABE(5)
10 DIM ANTAL(7,3), NUMMER(7)     // Se note 5
10 DIM BILER#(-5:15,3:8)
10 DIM A$(3:2), B(5)             // Se note 6
```

Kommentarer:

1. Matricer skal altid dimensioneres.

2. Matricer kan have vilkårligt mange dimensioner, kun begrænset af den til rådighed værende hukommelse og inputliniens maksimale længde på 159 karakterer.

3. Hver af elementerne i (liste af indicerede variable) specificeres under anvendelse af syntaksen:
   (variabelnavn)((liste af indexgrænser))
   hvor (variabelnavn) eventuelt indeholder typeerklæringskarakteren '#'.
   De enkelte elementer adskilles af komma.
   (liste af indexgrænser) angiver for hver dimension nedre og øvre grænse for pågældende dimension efter syntaksen:
   [(nedre grænse):](øvre grænse)
   De enkelte dimensioner adskilles af komma.
   Hvis der ikke angives nogen nedre grænse, sættes denne til 1.

4. 'DIM' instruktionen tildeler værdien nul til de enkelte elementer.

5. Flere variable kan dimensioneres på den samme linie.

6. Aritmetiske- og strengvariable kan dimensioneres på den samme linie.

Type:

Instruktion

Formål:

At reservere hukommelsesplads for strengvariable og matricer, samt at angive grænserne for indices.

Syntaks:

DIM ⟨indiceret variabel⟩ OF ⟨længde⟩

Udførelse:

Under hensyn til variablens dimension og længde udregnes og afsættes det nødvendige hukommelsesareal.

Eksempler:

```
10 DIM A$ OF 80                          // Se note 9
10 DIM A$(3) OF 10                       // Se note 7
10 DIM B$(0:1,3) OF 25                   // Se note 8
10 DIM A$(3:2) OF 10, B$(5) OF 25        // Se note 5
10 DIM A$(5) OF 15, C(5)                 // Se note 6
```

Kommentarer:

1. Matricer og strengvariable skal altid dimensioneres.

2. Matricer kan have vilkårligt mange dimensioner, kun begrænset af den til rådighed værende hukommelse og inputliniens maksimale længde på 159 karakterer.

3. Hver af elementerne i ⟨liste af indicerede variable⟩ specificeres under anvendelse af syntaksen:
   ⟨variabelnavn⟩(⟨liste af indexgrænser⟩) OF ⟨længde⟩
   hvor ⟨variabelnavn⟩ indeholder typeerklæringskarakteren '#'.
   De enkelte elementer adskilles af komma.
   ⟨liste af indexgrænser⟩ angiver for hver dimension nedre og øvre grænse for pågældende dimension efter syntaksen:
   [⟨nedre grænse⟩:]⟨øvre grænse⟩
   De enkelte dimensioner adskilles af komma.
   Hvis der ikke angives nogen nedre grænse, sættes denne til 1.
   ⟨længde⟩ angiver den maksimale længde af strengvariablen eller af hvert element i strengmatricen.
   Den aktuelle længde af en streng variabel/element kan være fra nul karakterer (den tomme streng) op til og inklusive den angivne længde.

4. 'DIM' instruktionen tildeler værdien "" (tom streng) til de enkelte elementer.

5. Flere variable kan dimensioneres på den samme linie.

6. Aritmetiske- og strengvariable kan dimensioneres i den samme linie.

7. Denne matrice indeholder elementerne A$(1), A$(2) og A$(3), som hver har en maksimal længde på 10 karakterer.

8. Denne matrice indeholder elementerne B$(0,1), B$(0,2), B$(0,3), B$(1,1), B$(1,2) og B$(1,3) som hver har en maksimal længde på 25 karakterer.

9. En strengvariabel behøver ikke at være en matrice.

Type:

Aritmetisk operator

Formål:

At udføre heltalsdivision mellem to aritmetiske udtryk.

Syntaks:

⟨udtryk1⟩ DIV ⟨udtryk2⟩

Udførelse:

⟨udtryk1⟩ divideres med ⟨udtryk2⟩ og resultatet afrundes til heltal.

Eksempel:

```
100 A#:=B DIV C
100 NUMBER:=17 DIV NUM
```

Kommentarer:

1. Resultatet N er defineret ved den heltallige værdi af N som får udtrykket:

   ⟨udtryk1⟩ − N * ⟨udtryk2⟩

   til at antage sin mindste ikke-negative værdi.

2. Beregningen udføres, ved først at udføre en normal reel division, hvorefter resultatet omformes til et heltal. Resultatets type afhænger af ⟨udtryk1⟩ og ⟨udtryk2⟩ på følgende måde:

⟨udtryk1⟩	DIV	⟨udtryk2⟩	resultat
reel		reel	reel
reel		heltal	reel
heltal		reel	reel
heltal		heltal	heltal

3. Se også 'MOD' operatoren.

Type:

Kommando

Formål:

At lette indførelsen af rettelser i programmer, som allerede findes i datamatens arbejdslager.

Syntaks:

EDIT [⟨nedre grænse⟩][,⟨øvre grænse⟩]
EDIT [⟨nedre grænse,]

Udførelse:

Det angivne område hentes, linie for linie, frem fra datamatens arbejdslager og udskrives på skærmen.
Cursoren placeres umiddelbart efter sidste tegn og kan flyttes frem og tilbage over teksten ved hjælp kontroltaster.
Cursoren anbringes over det tegn, som skal ændres, det nye indtastes og cursoren rykker en plads til højre.
Når de ønskede rettelser er indført, trykkes på 'RETURN' tasten, hvorefter linien syntakskontrolleres og lægges, hvis den er korrekt, på plads i arbejdslageret.
Næste linie vises og hele forløbet gentages indtil ⟨øvre grænse⟩ nås.

Eksempler:

EDIT
EDIT 100
EDIT 100,
EDIT ,100
EDIT 100,200

Kommentarer:

1. Hvis ⟨nedre grænse⟩ udelades, begyndes med programmets første linie.

2. Hvis ⟨øvre grænse⟩ udelades, fortsættes til programmets sidste linie (eller indtil 'ESC' tasten nedtrykkes).

3. Hvis begge grænser udelades, begyndes med programmets første linie og fortsættes til sidste linie (eller indtil 'ESC' tasten nedtrykkes).

4. Hvis kun ⟨nedre grænse⟩ angives (ikke efterfulgt af komma) editeres kun den pågældende linie.

5. Alle de editeringsfaciliteter, som er nævnt i kapitlet 'Input editering' i afsnit 1, kan anvendes.

6. Linienummeret kan også editeres, hvilket medfører, at linien lægges på den plads i arbejdslageret, som det nye linienummer angiver. Hvis der i forvejen findes en linie med dette nummer, slettes denne.
   Den originale linie fjernes i dette tilfælde ikke fra programmet. (Brug kommandoen DEL XXXX).

7. Når 'RETURN' tasten nedtrykkes, lægges linien på plads i arbejdslageret, nøjagtig som den står på skærmen og uden hensyn til, hvor cursoren står.

8. 'EDIT' kommandoen kan når som helst afbrydes ved tryk på 'ESC' tasten, men ændringer indføres kun i den aktuelle linie, når 'RETURN' tasten nedtrykkes.

Type:

Instruktion

Formål:

At standse udførelsen af et program.

Syntaks:

END

Udførelse:

Programafviklingen standses, og promtkarakteren '*' udskrives for at vise, at COMAL-80 er klar til at modtage inddata

Eksempel:

```
 10 K:=0
 20 IF K)100 THEN
 30  END
 40 ELSE
 50  GOTO JENS
 60 ENDIF
 70 LABEL JENS
 80 PRINT K," ",
 90 K:+1
100 GOTO 20
```

Kommentarer:

1. 'END' instruktionen giver, i modsætning til 'STOP' instruktionen, ikke nogen udskrift om, hvor programafviklingen er standset.

2. 'END' instruktionen er valgfri som sidste linie i et program, idet COMAL-80 selv indsætter en sådan, for brugeren usynlig, instruktion sidst i ethvert program. Når denne 'END' instruktion mødes, udskrives meddelelsen:

   PROGRAMUDFØRELSE AFSLUTTET

Type:

Kommando

Formål:

At hente en fil, som en streng af ASCII karakterer, fra baggrundslageret og lægge den på plads i arbejdslageret.

Syntaks:

ENTER (filnavn)

Udførelse:

Fra baggrundslageret hentes den angivne fil, karakter for karakter. Efter hver 'RETURN' syntakskontrolleres den derved dannede linie som, hvis den er korrekt, lægges på plads i arbejdslageret. Hvis der er fejl, standses indlæsningen, og linien, med tilhørende fejlmeddelelse, vises på dataskærmen. Indlæsningen fortsætter derefter, men linier med syntaksfejl gemmes ikke i arbejdslageret.

Eksempler:

```
ENTER DKO:PROGRAM
ENTER POLYNO
```

Kommetarer:

1. 'ENTER' kommandoen kan kun indlæse filer, som er uoskreskrevet på ASCII format, f.eks. ved hjælp af 'LIST' kommandoen.

2. Arbejdslageret slettes ikke inden indlæsningen, men nyoplæste linier med et linienummer, som allerede findes i arbejdslageret, erstatter den gamle linie. Denne overskrivning sker på liniebasis og altså uden hensyn til eventuelle forskelle i linielængde. Dette kan anvendes til at blande to eller flere filer ved i forvejen at sørge for, at der ikke er sammenfaldende linienumre. Bortset fra denne anvendelse bør man altid slette arbejdslageret med 'NEW' kommandoen, inden en fil oplæses med 'ENTER' kommandoen.

3. ASCII filer kan læses af alle versioner af COMAL-80, hvorfor dette format tilrådes brugt ved langtidsopbevaring af filer.

Type:

Systemfunktion

Formål:

At angive om alle data fra de i programmet indgående 'DATA' instruktioner er læst.

Syntaks:

EOD()

Udførelse:

'EOD()' har værdien falsk (= 0), sålænge der i programmets 'DATA' instruktioner er data, som ikke er blevet læst. Når sidste sæt data læses, tildeles 'EOD()' værdien sand (= 1).
Udføres der derefter en 'RESTORE' instruktion, tildeles 'EOD()' igen værdien falsk.

Eksempel:

```
10 WHILE NOT EOD() DO
20  READ A
30  PRINT A
40 ENDWHILE
50 DATA 55, 2, -15, 35
```

Kommentarer:

1. Under programmering kan 'EOD' og 'EOD()' valgfrit anvendes, men i programlistninger anvendes 'EOD()'.

Type:

Systemfunktion

Formål:

At angive, om alle data i en datafil er læst.

Syntaks:

EOF ((filnummer))

Udførelse:

Ved udførelse af en 'OPEN FILE' instruktion eller kommando af type 'READ', tildeles den hertil hørende EOF ((filnummer)) systemfunktion værdien 0 (eller falsk). Når sidste værdi i filen er læst, tildeles den værdien 1 (eller sand).

Eksempel:

```
10 OPEN FILE 0,"TEST",READ
20 REPEAT
30  READ FILE 0: A
40 UNTIL EOF(0)
```

Kommentarer:

1. (filnummer) er et aritmetisk udtryk.

Type:

Systemfunktion

Formål:

At huske, om der under programafviklingen har været en ikke-fatal fejl.

Syntaks:

ERR()

Udførelse:

Under normal programafvikling vil enhver fejl standse programmet og give fejludskrift. Der findes imidlertid en række fejl, som det er muligt at omgå på en veldefineret måde. I sådanne fejltilfælde kan man undgå programafbrydelser, ved, inden de opstår, at udføre en 'TRAP ERR-' instruktion. Når den er udført, vil ikke-fatale fejl i stedet medføre, at systemfunktionen 'ERR()' tildeles værdien sand, fejlen omgås og programafviklingen fortsætter.

Eksempel:

```
 10 INIT ""
 20 TRAP ERR-
 30 OPEN FILE 0, "XPLOCOMM", READ
 40 TRAP ERR+
 50 IF NOT ERR() THEN
 60  INPUT FILE 0: DEFAULT_FILNAVN$
 70 ELSE
 80  DEFAULT_FILNAVN$:="XPLOPROG"
 90 ENDIF
100 CLOSE
```

Kommentarer:

1. Når udførelsen af et program starter, tildeles systemvariablen 'ERR()' værdien falsk. Når en ikke-fatal fejl opstår, tildeles 'ERR()' værdien sand og beholder denne værdi, til der spørges på den. Umiddelbart efter at der er spurgt på værdien, eller den iøvrigt er anvendt, tildeles 'ERR()' værdien falsk.
   Normalt tildeler COMAL-80 værdien sand til en variabel ved at sætte variablen lig med 1, men i dette tilfælde tildeles 'ERR' fejlens nummer som værdi. Da enhver værdi forskellig fra 0 behandles som sand i tests, vil konstruktioner som 'IF ERR() THEN ...' arbejde normalt.
   Fejlnumrene er nærmere beskrevet i appendix A.

2. Ved at udføre en 'TRAP ERR+' instruktion går systemet tilbage til den normale fejlbehandling.

3. Under programmering kan 'ERR' og 'ERR()' frit anvendes, men i listninger anvendes 'ERR()'.

Type:

Tegnfunktion

Formål:

At give adgang til COMAL-80 systemets fejlmeldinger.

Syntaks:

ERRTEXT$((udtryk))

Udførelse:

(udtryk), som skal være aritmetisk, beregnes og afrundes om nødvendigt. Den hertil svarende fejlmelding returneres derefter.

Eksempel:

```
10 DIM A$ OF 30
20 FOR I=1 TO 295
30  A$:=ERRTEXT$(I)
40  PRINT A$
50 NEXT I
```

Kommentarer:

1. Denne funktion virker kun, hvis fejlmeldingerne ikke er frakoblet ved COMAL-80 systemets indlæsning. Er de det, giver funktionen en tom streng som resultat.

Type:

Systemfunktion

Formål:

At huske om datamatens 'ESC' tast har været nedtrykket.

Syntaks:

ESC()

Udførelse:

Under normal programafvikling undersøges det, før hver instruktion udføres, om tastaturets 'ESC' tast har været nedtrykket. Er dette tilfældet, standser programafviklingen. Hvis der tidligere i programmet er udført en 'TRAP ESC-' instruktion, blokeres denne funktion, og systemfunktionen 'ESC()' tildeles i stedet værdien sand.

Eksempel:

```
10 TRAP ESC-
20 REPEAT
30  PRINT "TRYK PÅ ",
40  FOR I=1 TO 1000
50  NEXT I
60  PRINT "ESC-TASTEN!"
70 UNTIL ESC()
80 STOP
```

Kommentarer:

1. Når programafviklingen starter, tildeles 'ESC()' systemfunktionen værdien falsk (= 0).
   Når 'ESC' tasten nedtrykkes, tildeles den værdien sand (= 1) og beholder denne værdi, til der spørges på dens værdi. Umiddelbart efter at værdien er anvendt, tildeles 'ESC()' igen værdien falsk.

2. Systemet går tilbage til normal behandling af 'ESC' tasten, når der udføres en 'TRAP ESC+' instruktion.

3. Under programmering kan 'ESC' og 'ESC()' frit anvendes, men i listninger anvendes 'ESC()'.

Type:

Instruktion

Formål:

At kalde en navngiven programdel og, efter at denne er udført, at vende tilbage til den følgende linie.

Syntaks:

EXEC (procedurenavn) [((aktuel parameterliste))]

Udførelse:

Proceduren, bestemt ved navnet (procedurenavn), kaldes, idet (aktuel parameterliste) indsættes i procedurens formelle parameterliste.
Når procedurens 'ENDPROC' instruktion mødes, fortsættes programafviklingen i første linie efter 'EXEC' instruktionen.

Eksempel:

```
100 EXEC TEST
100 EXEC FATAL_ERROR("ERROR IN X-PL/0-COMPILER")
100 EXEC ERROR(30)
100 EXEC ENTER_(CONSTANT#,LEV#,TX#,DX#)
100 EXEC EXPRESSION(FNINCLUDE(FSYS,RPAREN#),LEV#,TX#)
```

Kommentarer:

1. Antallet af aktuelle parametre skal være det samme som antallet af formelle parametre i 'PROC' instruktionen. Desuden skal der for hver enkelt parameter være overensstemmelse med hensyn til dimensionering og type.

2. Hvis den formelle parameter er angivet med 'REF', skal der indsættes en variabel (evt. indiceret) som aktuel parameter.

3. Hvis den formelle parameter ikke er angivet med 'REF', skal den aktuelle parameter være et udtryk af tilsvarende type, evt. blot et variabelnavn. Dog kan aktuelle heltalsparametre indsættes i en formel reel parameter.

4. Se kapitlet 'Parameter indsættelse' i afsnit 1 for nærmere beskrivelse.

Type:

Aritmetisk funktion

Formål:

At beregne e opløftet til potensen angivet ved et numerisk udtryk.

Syntaks:

EXP((udtryk))

Udførelse:

Grundtallet for de naturlige logaritmer e (=2.718282) opløftes til den ved (udtryk) angivne potens.

Eksempel:

```
10 INPUT A
20 PRINT EXP(A)
```

Kommentarer:

1. (udtryk), som er et aritmetisk, kan være af reel eller heltallig type. Resultatet er altid reelt.

2. Værdien af (udtryk) skal være (= 88.02969 ved 7 cifres nøjagtighed eller 292.4283068102 ved 13 cifres nøjagtighed, eller programafviklingen standser med en fejlmelding.

Type:

Kommando

Formål:

At muliggøre indsættelse af brugerdefinerede instruktioner, funktioner og operatorer i COMAL-80.

Syntaks:

EXTENSION ‹filnavn›

Udførelse:

‹filnavn› åbnes og overføres til arbejdslageret. De heri angivne navne kædes til COMAL-80 og bliver reserverede ord.

Eksempel:

```
EXTENSION GRAPHPAC
```

Kommentarer:

1. Denne kommando er kun tilladt, når der ikke er noget program i datamatens arbejdslager.

2. For nærmere beskrivelse henvises til afsnittet 'EXTENSIONS' i kapitel 1 og tillæg E.

Type:

Systemkonstant

Formål:

Hovedsagelig at tildele en boolesk variabel værdien falsk.

Syntaks:

FALSE

Udførelse:

Returnerer værdien 0.

Eksempel:

```
 10 // PRIMTALPROGRAM
 20 //
 30 DIM FLAGS#(0:8190)
 40 SIZE1:=8190
 50 //
 60 COUNT:=0
 70 MAT FLAGS#:=TRUE
 80 //
 90 FOR I:=0 TO SIZE1 DO
100  IF FLAGS#(I) THEN
110   PRIME:=I+I+3
120   K:=I+PRIME
130   WHILE K<=SIZE1 DO
140    FLAGS#(K):=FALSE
150    K:+PRIME
160   ENDWHILE
170   COUNT:+1
180  ENDIF
190 NEXT I
200 PRINT "TOTAL NUMBER OF PRIMES: ",COUNT
```

Type:

Instruktion

Formål:

At afgrænse en programdel og angive hvor mange gange denne del skal udføres.

Syntaks:

```
FOR (styrevariabel) := (start) TO (slut) [STEP (trin)]
.
.
.
.
NEXT (styrevariabel)
```

Udførelse:

Når 'FOR' instruktionen mødes, sættes (styrevariabel) = (start), og det beregnes, om uligheden:

((slut)-(styrevariabel))*SGN((trin)))= 0

er opfyldt. Er dette ikke tilfældet, overspringes den af 'FOR...NEXT' strukturen omgivne programdel, og programafviklingen fortsætter i første udførbare linie efter 'NEXT' instruktionen.
Er uligheden opfyldt, gennemløbes programdelen til 'NEXT' instruktionen mødes. Herefter springes tilbage til første udførbare linie efter 'FOR' instruktionen, den beregnede værdi for (trin) adderes til (styrevariabel), og ovenstående ulighed undersøges igen med den nye værdi for (styrevariabel) og den tidligere beregnede værdi for (slut).
Dette gentages, indtil uligheden ikke længere er opfyldt.

Eksempel:

```
10 FOR I = 1 TO 100 STEP 5
20 PRINT I, " ",
30 NEXT I
40 STOP
```

Kommentarer:

1. Hvis (trin) udelades, benyttes værdien 1.

2. Hvis 'DOWNTO' benyttes i stedet for 'TO', negeres (trin)

3. Efter at 'FOR...NEXT' programdelen er udført, har (styrevariabel) den værdi, som ikke opfyldte ovenstående ulighed.

4. Op til 5 'FOR...NEXT' strukturer kan anbringes inden i hinanden, men de skal hver have egen (styrevariabel).
   Hvert niveau af subrutiner tildeles ligeledes en 'FOR...NEXT' dybde på 5, således at man ved brug af 'GOSUB' instruktioner eller ved brug af procedurer, kan opnå enhver ønsket dybde.

5. Hver 'NEXT' instruktion skal indeholde en og kun en (styrevariabel), som skal svare til den, som er anvendt i den tilhørende 'FOR' instruktion.

6. Det er muligt at afbryde en 'FOR...NEXT' programdel med 'GOTO' instruktionen.

7. Startværdien af (styrevariabel) tildeles før (slut). Programkonstruktioner af typen:

```
10 J:= x
20 FOR J=1 TO J+x
30  PRINT J
40 NEXT J
```

vil derfor blive udført x+1 gange.

8. Der skal være en, og kun en, 'NEXT' instruktion for hver 'FOR' instruktion.

9. Under programmering kan ':=' og '=' anvendes frit. I listninger anvendes ':='.

10. (styrevariabel) skal være en aritmetisk variabel.

Type:

Aritmetisk funktion

Formål:

At udskille decimaldelen af et reelt tal.

Syntaks:

FRAC((udtryk))

Udførelse:

Resultatet beregnes efter subtraktionen:

(udtryk)-INT((udtryk))

Eksempel:

```
10 INPUT A
20 PRINT FRAC(A)
30 PRINT FRAC(5.72)
40 PRINT FRAC(-5.72)
```

Kommentarer:

1. (udtryk) skal være aritmetisk og af reel type. Resultatet er altid af reel type.

2. Når (udtryk) er et positivt tal, beregnes resultatet ved at bortkaste cifrene foran decimalpunktummet.
   Er (udtryk) negativt, er resultatet 1 minus decimalerne i (udtryk).

Type:

System funktion

Formål:

At beregne den frie plads i arbejdslageret.

Syntaks:

FREESTORE()

Udførelse:

På grundlag af den øjeblikkelige anvendelse af arbejdslageret beregnes den tilbageværende plads.

Eksempel:

```
10 PRINT FREESTORE()
```

Kommentarer:

1. Under programmering kan 'FREESTORE' og 'FREESTORE()' frit anvendes. I listninger anvendes 'FREESTORE()'.

Type:

Instruktion

Formål:

At definere og navngive en brugerdefineret funktion.

Syntaks:

```
FUNC (navn) [(formel parameter liste)] [CLOSED]
.
.
.
ENDFUNC (navn)
```

Udførelse:

Når en 'FUNC' instruktion mødes, overspringes programdelen til den tilhørende 'ENDFUNC' instruktion og programafviklingen fortsætter i den følgende linie.
Når funktionen kaldes ved at dens navn (eventuelt fulgt af en aktuel parameter liste) anvendes i et udtryk, udregnes funktionen og værdien indsættes i udtrykket, som derefter færdigberegnes.

Eksempler:

```
10 FUNC X_Y_OPLØFTET(X,Y)
20  RETURN X^3/Y^2
30 ENDFUNC X_Y_OPLØFTET
40 I:=2
50 J:=3
60 RESULTAT:=X_Y_OPLØFTET(I,J)
70 PRINT RESULTAT
```

```
10 X:=2
20 Y:=3
30 FUNC X_Y_OPLØFTET CLOSED
40  IMPORT X,Y
50  RETURN X^3/Y^2
60 ENDFUNC X_Y_OPLØFTET
70 RESULTAT:=X_Y_OPLØFTET
80 PRINT RESULTAT
```

Kommentarer:

1. 'FUNC' instruktionen må ikke forekomme inden i følgende instruktioner:
   - betingede instruktioner
   - repeterende instruktioner
   - 'PROC' instruktioner
   - funktionserklæringer

2. (navn) skal være et legalt variabelnavn.

3. En funktion kan kalde andre procedurer eller sig selv (rekursion). En lukket funktion kan kun kalde lukkede funktioner eller procedurer.

4. (formel parameter liste) indeholder navnene på de formelle parametre, som ved kald af funktionen får overført værdier fra de aktuelle parametre i det tilhørende kald.

5. De ændringer, der sker med en variabel i en funktion, er lokale, medmindre det med 'REF' angives, at værdien, når funktionen er færdigafviklet, skal føres tilbage til de aktuelle parametre i hovedprogrammet.

6. 'REF' kan angives for simple variable og skal angives for matricevariable.

7. En funktions type kan være reel, heltal eller streng.

8. Ved matricevariable skal navnet efterfølges af en dimensionangivelse, som består af en parentes med et antal kommaer svarende til dimensionstallet minus 1.
   For en tredimensional matrice indeholder parentesen altså 2 kommaer, medens en vektor efterfølges af en tom parentes.

9. Hvis funktionen er erklæret lukket med 'CLOSED' instruktionen, er alle variabelnavne lokale og kan anvendes i anden betydning uden for funktionen. Dette kan ophæves for en eller flere variable med 'IMPORT' instruktionen.

10. 'INPUT' og 'PRINT USING' instruktioner er ikke tilladt i funktioner.

11. Hvis programafsnittet mellem 'FUNC' og 'ENDFUNC' indeholder en eller flere strukturer, skal den/disse være afsluttet inden 'ENDFUNC' instruktionen.

12. Funktionens værdi returneres fra funktionen med 'RETURN' instruktionen. Udelades denne instruktion er funktionens værdi udefineret.

Type:

Instruktion, kommando

Formål:

At fortælle, hvilken magnetisk baggrundsenhed, der er aktuel standardenhed.

Syntaks:

GETUNIT [<variabel>]

Udførelse:

Navnet på den aktuelle standardenhed tildeles <variabel> som en kode, bestående af to bogstaver og et tal, efterfulgt af kolon.

Eksempel:

```
100 GETUNIT DISK$
    GETUNIT
```

Kommentarer:

1. Når 'GETUNIT' bruges som kommando, skal <variabel> udelades og resultatet udskrives på terminalen.
   I instruktioner skal <variabel> specificeres.

2. De to bogstaver angiver baggrundslagerets type, hvor 'DK' betyder floppydisk. Tallet angiver enhedens nummer.

3. <variabel> er en strengvariabel.

Type:

Instruktion

Formål:

At kalde et underprogram (en subrutine), eventuelt fra flere steder i det samme program, og komme retur til den efter kaldet følgende linie.

Syntaks:

```
                    GOSUB (linienummer)
                    .
                    .
                    .
(linienummer)       .
                    .
                    .
                    RETURN
```

Udførelse:

Når 'GOSUB' instruktionen mødes, fortsætter programafviklingen i det angivne (linienummer), indtil 'RETURN' instruktionen mødes, hvorefter der fortsættes med linien umiddelbart efter den kaldende 'GOSUB' instruktion.

Eksempel:

```
10 PRINT "JEG STARTER I HOVEDPROGRAMMET"
20 GOSUB 50
30 PRINT "JEG ER NU TILBAGE I HOVEDPROGRAMMET"
40 STOP
50 PRINT "JEG ER NU I UNDERPROGRAMMET (SUBRUTINEN)"
60 RETURN
```

Kommentarer:

1. En subrutine kan kaldes et vilkårligt antal gange.

2. En subrutine kan kalde en ny subrutine, som igen kan kalde en ny og så videre. Dybden i dette er kun begrænset af den til rådighed værende hukommelse.

3. 'RETURN' instruktionen får programafviklingen til at fortsætte i linien umiddelbart efter den sidst udførte 'GOSUB' instruktion.

4. En subrutine kan indeholde flere 'RETURN' instruktioner.

5. Subrutiner kan anbringes hvorsomhelst i programmet, men det anbefales, at de adskilles tydeligt fra hovedprogrammet.

6. For at undgå uønsket udførelse af en subrutine, anbefales det, at anbringe en 'STOP', 'GOTO' eller 'END' instruktion umiddelbart før subrutinen.

7. BEMÆRK. Nøgleordet 'RETURN' anvendes også i forbindelse med funktioner og procedurer. Dette er nærmere beskrevet på side 2-098.

Type:

Instruktion

Formål:

At afbryde den normale, sekventielle programafvikling og fortsætte i en angivet linie.

Syntaks:

GOTO (linienummer)
GOTO (navn)

Udførelse:

Programafviklingen fortsættes i den angivne linie eller, hvis denne ikke kan udføres, i den første udførbare linie herefter.

Eksempler:

```
10 PRINT "OL",
20 GOTO 40
30 STOP
40 PRINT "E"
50 GOTO 30
```

```
10 PRINT "OL",
20 GOTO REST
30 LABEL SLUT
40 STOP
50 LABEL REST
60 PRINT "E"
70 GOTO SLUT
```

Kommentarer:

1. Eksempler på ikke-udførbare linier er 'LABEL', 'DATA' og 'REM' instruktioner.

Type:

System variabel

Formål:

At styre, om navne i programlistninger skal vises med store eller små bogstaver.

Syntaks:

IDENTIFIERLOWER

Udførelse:

Værdien af systemvariablen 'IDENTIFIERLOWER' anvendes til at styre udskriftsformatet for navne.

Eksempel:

```
100 IDENTIFIERLOWER:=0
100 IDENTIFIERLOWER:=A
100 IDENTIFIERLOWER:=TRUE
100 PRINT IDENTIFIERLOWER
    IDENTIFIERLOWER:=1
```

Kommentarer:

1. Når COMAL-80 indlæses, tildeles 'IDENTIFIERLOWER' vårdien 0. Dette kan kun ændres med en tildeling.

2. Den tildelte værdi skal være 0 eller 1 og afrundes om nødvendigt.

3. Hvis værdien af 'IDENTIFIERLOWER' er 0, bliver alle navne vist med store bogstaver. Er værdien 1, bliver de vist med små.

4. Dette nøgleord kan anvendes som operand, eller der kan tildeles til det. Når det bruges som operand, er det af heltallig type.

5. 'NEW' instruktionen ændrer ikke den til 'IDENTIFIERLOWER' tildelte værdi.

Type:

Instruktion

Formål:

At udføre eller overspringe en instruktion afhængig af, om et logisk udtryk er sandt eller falsk.

Syntaks:

IF <logisk udtryk> [THEN] <instruktion>

Udførelse:

Er <logisk udtryk> sandt (<> 0) udføres <instruktion>, ellers ikke.

Eksempel:

```
10 INPUT "SKRIV ET TAL",A
20 IF A THEN PRINT "A <> 0"
30 IF A<0 THEN PRINT "A < 0"
40 IF A=0 THEN PRINT "A = 0"
50 IF A=1 THEN PRINT "A = 1"
60 IF A=2 THEN PRINT "A = 2"
70 IF A>2 THEN PRINT "A > 2"
```

Kommentarer:

1. Efter 'IF...THEN' instruktionen kan anvendes følgende instruktioner:
   CALL, CAT, CHAIN, CLEAR, CLOSE, CURSOR, DELETE, END, EXEC, EXIT, GETUNIT, GOSUB, GOTO, INIT, INPUT, LET, LOGOFF, LOGON, MAT, ON, OPEN, OUT, PAGE, POKE, PRINT, QUIT, RANDOM, READ, RECEIVE, RELEASE, RENAME, RESTORE, RETURN, SELECT, STOP, TRAP, UNIT og WRITE samt instruktioner defineret som 'EXTENSIONS'.
   Desuden er en ny 'IF...THEN' instruktion tilladt.

2. 'THEN' kan udelades under indtastning, men vil altid være indsat i programlistninger.

Type:

Instruktion

Formål:

At udføre en programdel, hvis et logisk udtryk er sandt og ellers overspringe denne programdel.

Syntaks:

```
IF (logisk udtryk) [THEN]
.
.
.
ENDIF
```

Udførelse:

Er (logisk udtryk) sand (<> 0), udføres den af 'IF...END-IF' omgivne programdel. Er (logisk udtryk) falsk (= 0) fortsættes med første udførbare linie efter 'ENDIF' instruktionen.

Eksempel:

```
10 IF MEMBER#(1 OR MEMBER#)31 THEN
20  EXEC FATALERROR("ERROR IN X-PL/0-COMPILER")
30 ENDIF
```

Kommentarer:

1. Under indtastning kan 'THEN' udelades, men vil altid være indsat i programlistninger.

Type:

Instruktion

Formål:

At udføre en af to programdele afhængig af, om et logisk udtryk er sandt eller falsk.

Syntaks:

```
IF (logisk udtryk) [THEN]
.
.
.
ELSE
.
.
.
ENDIF
```

Udførelse:

Er (logisk udtryk) sandt (<> 0) udføres den af 'IF... ELSE' omgivne programdel. Er (logisk udtryk) falsk (= 0) udføres den af 'ELSE...ENDIF' omgivne programdel.

Eksempel:

```
10 INPUT "GÆT PÅ ET TAL MELLEM 1 OG 5": A
20 B:=RND(1,5)
30 IF A=B THEN
40  PRINT "RIGTIGT!"
50 ELSE
60  PRINT "FORKERT. TALLET VAR: "; B
70 ENDIF
```

Kommentarer:

1. Under indtastning kan 'THEN' udelades. Det vil altid være indsat i programlistninger.

Type:

Instruktion

Formål:

At udføre en af flere programdele afhængig af, om et af flere logiske udtryk er sande.

Syntaks:

```
IF (logisk udtryk 1) [THEN]
.
.
ELIF (logisk udtryk 2) [THEN]
.
.
ELIF (logisk udtryk n) [THEN]
.
.
[ELSE
.
.]
ENDIF
```

Udførelse:

Hvert (logisk udtryk n) undersøges i rækkefølge. Er et af disse sandt (<> 0) udføres den tilhørende programdel til næste 'ELIF', 'ELSE' eller 'ENDIF' instruktion, hvorefter programafviklingen fortsætter i første udførbare linie efter 'ENDIF' instruktionen.
Er alle (logisk udtryk n) falske (= 0) udføres programdelen omgivet af 'ELSE...ENDIF', hvorefter der fortsættes med første udførbare linie efter 'ENDIF' instruktionen.

Eksempel:

```
10 INPUT "INDTAST ET AF TALLENE 1, 2 ELLER 3: ": A
20 IF A=1 THEN
30  PRINT "TALLET VAR 1"
40 ELIF A=2 THEN
50  PRINT "TALLET VAR 2"
60 ELIF A=3 THEN
70  PRINT "TALLET VAR 3"
80 ELSE
90  PRINT "JEG BAD OM ET AF TALLENE 1, 2 ELLER 3!"
100 ENDIF
```

Kommentarer:

1. Hvis flere (logisk udtryk n) er sande, udføres kun det første første af disse.

2. Hvis 'ELSE' instruktionen udelades, og ingen af (logisk udtryk n) er sande, fortsætter programafviklingen i første linie efter 'ENDIF'.

3. Under indtastning kan 'THEN' udelades, men indsættes altid i programlistninger.

Type:

Instruktion

Formål:

At gøre en eller flere variable i hovedprogrammet eller i en anden 'PROC' eller 'FUNC' kendt inden i en 'PROC' eller 'FUNC' struktur.

Syntaks:

IMPORT ⟨liste af variable⟩

Udførelse:

De i ⟨liste af variable⟩ anførte variable, gøres kendte indeni den 'PROC' eller 'FUNC' struktur som indeholder 'IMPORT' instruktionen.

Eksempel:

```
10 PROC ERROR(N#) CLOSED
20  IMPORT FATAL_FEJL:CC#, ERR_, ERRORS#
30  PRINT "*****"; SPC$(CC#-9); "^"; N#
40  ERR_:=INCLUDE(ERR_,N#+1); ERRORS:+1
50 ENDPROC ERROR
```

Kommentarer:

1. Variabelnavnene i ⟨liste af variable⟩ adskilles af komma Matricevariable kan kun overføres som helhed, og indices er derfor ikke tilladt.

2. Hvert variabelnavn i ⟨liste af variable⟩ kan forsynes med et foranstillet ⟨lukket område navn⟩, hvor ⟨lukket område navn⟩ er navnet på den procedure eller funktion, hvorfra variablen skal hentes.
   ⟨lukket område navn⟩ skal enten direkte eller indirekte (gennem andre lukkede områder), kalde den procedure eller funktion som indeholder 'IMPORT' instruktionen. Variable kan altså kun hentes fra lukkede områder, som i forvejen er blevet gennemløbet.
   Hvis ⟨lukket område navn⟩ udelades, hentes de angivne variable fra hovedprogrammet.

3. Denne instruktion må kun benyttes inden i lukkede procedurer og funktioner.

4. Udførelsen af en 'IMPORT' instruktion påvirker ikke tilgængeligheden af de angivne variable i nogen anden del af programmet end den 'PROC' eller 'FUNC' struktur, som indeholder 'IMPORT' instruktionen.

5. Alle operationer, som i hovedprogrammet er tilladt på de angivne variable, er også tilladt inden i den 'PROC' eller 'FUNC' struktur, som indeholder 'IMPORT' instruktionen.

6. Under programmering kan 'GLOBAL' og 'IMPORT' anvendes efter ønske, men i listninger anvendes 'IMPORT'.

Type:

Strengoperator

Formål:

At undersøge om en tekststreng er indeholdt i en anden.

Syntaks:

<udtryk1> IN <udtryk2>

Udførelse:

Det undersøges, om <udtryk1> er indeholdt i <udtryk2>. Er dette tilfældet, er den logiske værdi sand (= 1). Er det ikke tilfældet, er den logiske værdi falsk (= 0).

Eksempel:

```
10 DIM A$ OF 15
20 DIM B$ OF 15
30 INPUT "SKRIV EN TEKSTSTRENG:        ": A$
40 INPUT "SKRIV EN ANDEN TEKSTSTRENG: ": B$
50 IF B$ IN A$ THEN PRINT "2. STRENG ER INDEHOLDT I 1."
60 IF NOT B$ IN A$ THEN
70  PRINT "2. STRENG ER IKKE INDEHOLDT I 1."
80 ENDIF
```

Type:

System variabel

Formål:

At angive det antal karakterpositioner, som den interne del af en struktur skal indrykkes i programlistninger.

Syntaks:

INDENTION

Udførelse:

Den aktuelle værdi af systemvariablen 'INDENTION' bestemmer hvormange karakterpositioner det indre af en ny struktur skal rykkes mod højre i forhold til foregående linies startposition.

Eksempel:

```
100 INDENTION:=8
100 INDENTION=(A+3)*B
100 PRINT INDENTION
    INDENTION:=3
```

Kommentarer:

1. Når COMAL-80 indlæses, tildeles 'INDENTION' værdien 2.

2. Enhver tildelt værdi skal være et heltal mellem 0 og 10 iklusive. Den tildelte værdi vil blive afrundet om nødvendigt.

3. Der kan tildeles til dette nøgleord og det kan også anvendes som operand. Som operand er det af heltallig type.

4. 'NEW' kommandoen ændrer ikke værdien af systemvariablen 'INDENTION'.

Type:

Instruktion, kommando

Formål:

At klargøre en nyindsat diskette, som tidligere er formatteret og evt. brugt.

Syntaks:

INIT [<enhed>]

Udførelse:

Den angivne <enhed> initialiseres.

Eksempler:

```
100 INIT "DK0:"
    INIT
    INIT DK1:
```

Kommentarer:

1. Under CP/M initialiseres alle diskettedrev og <enhed> angivelsen anvendes ikke, men hvis den angives skal det være navnet på et diskettedrev. Ingen filer må være åbne, når denne instruktion/kommando udføres.

Type:

Maskinsprogsfunktion.

Formål:

At aflæse værdien på en af Z-80 mikroprocessorens inputporte.

Syntaks:

INP((udtryk))

Udførelse:

Inputporten, bestemt ved (udtryk), aflæses.

Eksempel:

```
10 PRINT INP(17)
```

Kommentarer:

1. (udtryk), som skal være numerisk, skal have en værdi i intervallet 0 <= X <= 255.
2. (udtryk) opfattes som et decimaltal, der om nødvendigt afrundes til et heltal.

Type:

Instruktion

Formål:

Under programmets afvikling at indlæse værdier fra terminalen og tildele disse til variable.

Syntaks:

INPUT [⟨tekst⟩:]⟨variabelliste⟩

Udførelse:

Når 'INPUT' instruktionen mødes, standser programafviklingen efter at en eventuel ⟨tekst⟩ er udskrevet på terminalen. Efterhånden som brugeren indtaster værdier, tildeles disse til de i ⟨variabelliste⟩ angivne variable fra venstre mod højre. Når brugeren, efter sidste værdi, trykker på 'RETURN' fortsætter programafviklingen.

Eksempler:

```
100 INPUT ABE, OLE, NAVN$
100 INPUT "SKRIV 3 TAL: ": A, B, C
```

Kommentarer:

1. Hvis 'INPUT' instruktionen indeholder en ⟨tekst⟩ udskrives denne nøjagtig som den står. Hvis der ikke er nogen, udskrives '?' som tegn på, at datamaten afventer inddata

2. Hvis ⟨variabeliste⟩ afsluttes med et komma, sker næste udskrivning på terminalen ved starten af næste printzone, hvor zonernes bredde sættes med 'ZONE'.

3. Hvis ⟨variabelliste⟩ afsluttes med et semikolon, sker næste udskrivning på terminalen i næste position, umiddelbart efter den sidst indtastede værdi.

4. Flere talværdier indtastes adskilt af et tegn, som ikke kan være del af et tal, som for eksempel komma eller blanktegn.

5. En strengkonstant indtastes som en streng af ASCII karakterer. Det er kun muligt at indtaste værdier efter en strengkonstant, hvis den afsluttes med 'RETURN'.
   Når en strengkonstant efterfølger en aritmetisk konstant, betragter COMAL-80 den første karakter, som ikke kan være del af den aritmetriske konstant, som et skilletegn og starter strengkonstanter med det næste tegn.

6. De indlæste værdiers type skal stemme overens med typerne angivet i 'INPUT' instruktionen.

7. (variabelliste) kan indeholde alle variabeltyper, men matricer skal være korrekt indiceret, og delstrenge kan ikke anvendes.

8. Indtastes en konstant af en forkert type, udskrives fejlmeddelelsen 'Fejl i tal' og indtastningen skal korrigeres. Der foretages ingen tildeling, før en acceptabel indtastning er foretaget.

9. Indtastes der for få konstanter, i forhold til det opgivne antal variabelnavne, udskrives et '?' på terminalen, og COMAL-80 afventer yderligere inddata.

10. Indtastes der for mange konstanter, i forhold til det opgivne antal variabelnavne, udskrives fejlmeddelelsen 'For meget input', og hele indtastningen må korrigeres.

11. 'INPUT' instruktioner er ikke tilladt i funktioner.

Type:
Instruktion

Formål:
At indlæse data fra en ASCII datafil, som er udskrevet med 'PRINT (USING) FILE' instruktionen.

Syntaks:
INPUT FILE ⟨filnummer⟩ [,⟨postnummer⟩]:⟨liste af variable⟩

Udførelse:
Fra den med ⟨filnummer⟩ angivne fil tildeles værdier til de enkelte elementer i ⟨liste af variable⟩.

Eksempel:
```
100 INPUT FILE 3: A$
100 INPUT FILE 0: B#, C
```

Kommentarer:

1. Inden 'INPUT FILE' instruktionen mødes, skal en fil være åbnet, og forbindelse mellem pågældende filnavn og det i 'INPUT FILE' instruktionen benyttede ⟨filnummer⟩ være etableret med en 'OPEN FILE' instruktion eller kommando og type 'READ' eller 'RANDOM'.

2. ⟨postnummer⟩ angives kun ved 'RANDOM' filer og er et aritmetisk udtryk, som evt. afrundes til et heltal.

3. ⟨filnummer⟩ er et aritmetisk udtryk.

4. ⟨liste af variable⟩ kan indeholde alle variabeltyper, men matricer skal være korrekt indekserede og delstrenge må ikke anvendes.

5. Hvert element i ⟨liste af variable⟩ adskilles med komma.

6. Under indtastning kan 'FILE' og '#' anvendes i flæng. I programlistninger anvendes 'FILE'.

7. Kommentarerne 4, 5, 6 og 11 under 'INPUT' instruktionen, gælder også her.

Type:

Aritmetisk funktion

Formål:

At danne det største heltal, som er mindre end eller lig med et givet tal.

Syntaks:

INT((udtryk))

Udførelse:

Det største heltal <= (udtryk) udregnes.

Eksempel:

```
10 INPUT A
20 B:=INT(A)
30 PRINT B
40 PRINT INT(5.72)
50 PRINT INT(-5.72)
```

Kommentarer:

1. (udtryk) er af reel type. Resultatet er et heltal af reel type.

2. Se også 'ROUND' og 'TRUNC' funktionerne.

Type:

Aritmetisk funktion

Formål:

At omsætte et heltal, som forefindes som en tegnstreng til et heltal.

Syntaks:

IVAL((strengudtryk))

Udførelse:

Karaktererne i (strengudtryk), som skal udgøre et heltal, omsættes til heltal.

Eksempel:

```
10 DIM A$ OF 5
20 INPUT "SKRIV ET HELTAL: ": A$
30 B#:=IVAL(A$)
40 PRINT B#
```

Kommentarer:

1. Hvis tegnstrengen i (strengudtryk) indeholder andre karakterer end cifre inklusive evt. fortegn, standser programafviklingen med en fejlmelding.

2. Se også 'VAL' funktionen.

Type:

System variabel

Formål:

At angive om nøgleord i programlistninger skal vises med små eller store bogstaver.

Syntaks:

KEYWORDLOWER

Udførelse:

Den aktuelle værdi af systemvariablen 'KEYWORDLOWER' bestemmer, om nøgleord i programlistninger skal skrives med små eller store bogstaver.

Eksempel:

```
100 KEYWORDLOWER:=0
100 KEYWORDLOWER:=A
100 KEYWORDLOWER:=TRUE
100 PRINT KEYWORDLOWER
    KEYWORDLOWER:=1
```

Kommentarer:

1. Når COMAL-80 indlæses, tildeles 'KEYWORDLOWER' værdien 2

2. Enhver tildelt værdi skal være 0 eller 1. Den tildelte værdi bliver afrundet om nødvendigt.

3. Hvis 'KEYWORDLOWER' har værdien 0 bliver alle nøgleord vist med store bogstaver.

4. Der kan tildeles til dette nøgleord, og det kan bruges som operand. Som operand er det af heltallig type.

5. 'NEW' kommandoen ændrer ikke værdien af systemvariablen 'KEYWORDLOWER'.

Type:

Instruktion

Formål:

At navngive et punkt i et COMAL-80 program som reference for 'GOTO' og 'RESTORE' instruktionerne.

Syntaks:

LABEL (navn)

Udførelse:

'LABEL' instruktionen er ikke udførbar og overspringes under programudførelsen.

Eksempel:

```
10 LABEL START
20 INPUT "SKRIV ET TAL: ": TAL
30 PRINT TAL
40 GOTO START
```

Kommentarer:

1. En 'LABEL' instruktion, som er anvendt indeni en procedure eller funktion, kan der kun refereres til fra det lokale område.

Type:

Aritmetisk funktion

Formål:

At beregne den aktuelle længde af en strengvariabel.

Syntaks:

LEN (⟨variabelnavn⟩)

Udførelse:

Det aktuelle antal karakterer i ⟨variabelnavn⟩ optælles.

Eksempel:

```
10 DIM A$ OF 15
20 INPUT A$
30 B#:=LEN(A$)
40 PRINT B#
```

Kommentarer:

1. Det er variablens aktuelle indhold, der er afgørende for dens længde. Den dimensionerede længde har kun betydning ved at være den maksimale værdi for resultatet.

Type:

Instruktion

Formål:

At tildele værdien af et udtryk til en variabel.

Syntaks:

[LET] ⟨variabel⟩ := ⟨udtryk⟩

Udførelse:

⟨udtryk⟩ beregnes, og den fundne værdi gemmes i de hukommelsespladser, som afsættes til ⟨variabel⟩.

Eksempel:

```
10 LET A := 5
20 LET B := 3
30 LET SUM := A+B
40 A:+B
50 FORSKEL := A-B
60 PRINT SUM
70 PRINT A
80 PRINT FORSKEL
```

Kommentarer:

1. Ordet 'LET' er valgfrit, og kan altså udelades, som vist i eksemplets linie 40. I programlistninger udelades det.

2. '=' og ':=' er ensbetydende her og kan frit anvendes efter ønske. I programlistninger anvendes ':='.

3. ⟨variabel⟩:=⟨variabel⟩+⟨udtryk⟩ kan kort skrives som som ⟨variabel⟩:+⟨udtryk⟩.
⟨variabel⟩:=⟨variabel⟩-⟨udtryk⟩ kan kort skrives som som ⟨variabel⟩:-⟨udtryk⟩. Denne sidste form kan dog ikke anvendes for strengvariable.

4. Typen for ⟨udtryk⟩ og ⟨variabel⟩ skal stemme overens, dog kan et heltalsudtryk tildeles til en reel variabel.

5. I strengvariable, hvor ⟨udtryk⟩ er længere end ⟨variabel⟩, vil ⟨udtryk⟩ blive afkortet fra højre.

6. I strengvariable, hvor ⟨udtryk⟩ er kortere end ⟨variabel⟩, får ⟨udtryk⟩ kun den aktuelle længde.

7. Ved tildeling til delstrenge skal ⟨udtryk⟩ og ⟨variabel⟩ have samme længde.

8. Flere tildelinger kan foretages på samme linie, adskildt af semikolon, men nøgleordet 'LET' (som kan udelades) må kun forekomme før den første tildeling.

Type:

Kommando

Formål:

At udskrive datamatens arbejdslager, helt eller delvist, som en streng af ASCII karakterer.

Syntaks:

```
LIST [<startlinie>][,<slutlinie>][<filnavn>]
LIST [<startlinie>,][<filnavn>]
```

Udførelse:

Den angivne del af programmet, som er i internt format, oversættes til en streng af ASCII karakterer og udskrives på den angivne enhed.

Eksempler:

```
LIST
LIST 10
LIST 10,100
LIST ,100
LIST 100,
LIST TEST
LIST 10,100 TEST
LIST ,100 DK1:TEST
LIST LP0:
```

Kommentarer:

1. Hvis <filnavn> ikke angives, sker alle udskrifter på den tilsluttede terminal med enhedsnavn 'DS0:'.
   Hvis den specificerede listning indeholder flere linier end denne enhed er i stand til at vise i et billede, vises kun den første side, og COMAL-80 afventer at mellemrumstasten nedtrykkes, som tegn på at næste side ønskes, eller 'RETURN' tasten, som tegn på at næste linie skal vises. Hvis 'ESC' tasten nedtrykkes, afbrydes listningen.

2. Hvis <startlinie> og <slutlinie> ikke angives, udskrives hele programmet. Hvis <startlinie> ikke angives, begynder listningen med programmets første linie. Hvis <slutlinie> ikke angives, fortsætter listningen til programmets sidste linie. Hvis kun <startlinie> angives, udskrives kun den pågældende linie.

3. 'LIST' kommandoen opfatter alle udskrifter som en transport af karakterer fra arbejdslageret til en fil.
   Man får derfor et program udskrevet på en tilsluttet printer ved som <filnavn> at angive 'LP:' evt. efterfulgt af den pågældende printers enhedsnummer. Hvis intet enhedsnummer angives vælges 'LP0:'

4. Listninger har ofte ikke det samme format, som da de blev indtastet, således sker der automatisk indrykninger så programstrukturen tydeligt fremgår.
   Dog indrykkes 'LABEL' instruktioner ikke, hvorved de bliver lettere at finde. I tilfælde, hvor flere nøgleord er ensbetydende, vælges en bestemt i alle udskrifter.

5. Hvis <filnavn> ikke indeholder en udvidelse, benyttes '.CML'.

6. Programmer gemt på et baggrundslager med 'LIST' kommandoen, kan senere indlæses med 'ENTER' kommandoen.

7. Programmer, der skal langtidsopbevares, eller udveksles, bør gemmes med 'LIST' kommandoen, da dette format forsøges gjort kompatibelt for alle COMAL-80 versioner.

8. Hvis <filnavn> allerede findes på den aktuelle enhed, meddeles dette brugeren, som får valget mellem at fortsætte, og få den gamle fil slettet, eller at standse.

Type:

Kommando

Formål:

At indlæse en binær fil fra det magnetiske baggrundslager.

Syntaks:

LOAD (filnavn)

Udførelse:

Datamatens arbejdslager slettes, og operativsystemet kaldes og indlæser filen.

Eksempler:

```
LOAD TEST
LOAD DK1:KARTOTEK
```

Kommentarer:

1. Kun binære filer, altså filer, som er gemt med 'SAVE' kommandoen, kan indlæses med 'LOAD' kommandoen. Disse filer kan i katalogudskrifter kendes på, at navnet er udvidet med 'CSB'.

2. Filnavnsudvidelsen '.CSB' leveres altid af COMAL-80 og kan ikke angives af brugeren.

3. Alle 'EXTENSION's, som var indlæst da programmed blev 'SAVE'd, skal også være indlæst når programmet igen indlæses med 'LOAD'.

4. Før et program indlæses med 'LOAD', udføres der automatisk en 'NEW' kommando.

Type:

Aritmetisk funktion

Formål:

At beregne den naturlige logaritme af et aritmetisk udtryk.

Syntaks:

LOG(⟨udtryk⟩)

Udførelse:

Den naturlige logaritme til ⟨udtryk⟩, som er større end 0, beregnes.

Eksempel:

```
10 INPUT A
20 PRINT LOG(A)
```

Kommentarer:

1. ⟨udtryk⟩ skal være aritmetisk og af reel eller heltallig type. Resultatet er altid reelt.

2. Hvis ⟨udtryk⟩ er mindre end eller lig med 0 standser programafviklingen med en fejlmeddelelse.

Type:

Instruktion, kommando

Formål:

At afslutte logningen af udskrifter på dataskærmen og lukke logfilen.

Syntaks:

LOGOFF

Udførelse:

Logningen stoppes, og den hertil anvendte fil lukkes.

Eksempel:

```
100 LOGOFF
    LOGOFF
```

Type:
Instruktion, kommando

Formål:
At danne en log over alt, hvad der vises på skærmen.

Syntaks:
LOGON (filnavn)

Udførelse:
En fil med det angivne (filnavn) åbnes og alt hvad COMAL-80 udskriver på dataskærmen, udskrives også i denne fil.

Eksempel:
```
100 LOGON "LOGFILE.LOG"
    LOGON LOGFILE
    LOGON LP:
```

KOMMENTARER:

1. En logfil, som er gemt i en diskfil, kan indlæses og udskrives på skærmen med følgende program:

```
10 DIM A$ OF 160
20 OPEN FILE 0, "LOGFILE.LOG", READ
30 REPEAT
40  INPUT FILE 0: A$
50  PRINT A$
60 UNTIL EOF(0)
70 CLOSE
```

2. Hvis logningen stoppes med 'LOGOFF' instruktionen og derefter startes igen under samme (filnavn), bliver den nye information tilføjet til den allerede eksisterende fil.

Type:

Instruktion

Formål:

At gentage afviklingen af en programdel indtil en indre betingelse er opfyldt.

Syntaks:

```
LOOP
.
.
.
ENDLOOP
```

Udførelse:

Den af 'LOOP...ENDLOOP' omgivne programdel udføres, indtil en indre 'EXIT' instruktion mødes. Herefter fortsætter programafviklingen i første udførbare linie efter 'ENDLOOP' instruktionen.

Eksempel:

```
10 ANTAL:=0
20 LOOP
30  ANTAL:+1
40  PRINT ANTAL
50  IF ANTAL:=8 THEN EXIT
60 ENDLOOP
```

Kommentarer:

1. Udførelsen af 'LOOP...ENDLOOP' programdelen kan også afbrydes med en 'GOTO' instruktion.

2. 'EXIT' instruktionen afbryder den inderste 'LOOP...ENDLOOP' struktur, der omslutter 'EXIT' instruktionen.

Type:
Instruktion

Formål:
At tildele en værdi til hvert element i en matrice.

Syntaks:
MAT ⟨variabel⟩:=⟨udtryk⟩

Udførelse:
Hvert enkelt element i variabel tildeles værdien af ⟨udtryk⟩.

Eksempel:
```
10 DIM IND(50)
20 MAT IND:=10
```

Kommentarer:

1. ⟨variabel⟩ og ⟨udtryk⟩ skal være af samme type, dog kan et heltalligt udtryk tildeles til elementerne i en reel matrice.

2. '=' og ':=' er ensbetydende og kan frit anvendes. I programlistninger anvendes ':='.

3. I strengvariable, hvor ⟨udtryk⟩ er længere end ⟨variabel⟩, vil ⟨udtryk⟩ blive afkortet fra højre.

4. I strengvariable, hvor ⟨udtryk⟩ er kortere end ⟨variabel⟩, får denne kun den aktuelle længde.

5. Der kan på samme linie foretages flere tildelinger, adskildt af semikolon, men nøgleordet 'MAT' må kun forekomme foran den første tildeling.

Type:

Aritmetisk operator

Formål:

At bestemme resten efter en heltalsdivision.

Syntaks:

⟨udtryk1⟩ MOD ⟨udtryk2⟩

Udførelse:

⟨udtryk1⟩ heltalsdivideres med ⟨udtryk2⟩ og resten, som er ⟨udtryk1⟩ minus resultatet gange ⟨udtryk2⟩ bestemmes.

Eksempel:

```
10 INPUT A
20 B:=A MOD 7
30 PRINT B
```

Kommentarer:

1. Resultatet N er defineret ved den mindste ikke negative værdi, som udtrykket:

   ⟨udtryk1⟩ - N * ⟨udtryk2⟩

   kan antage for heltallige N.

2. Resultatets type afhænger af typen af ⟨udtryk1⟩ og ⟨udtryk2⟩ på følgende måde:

⟨udtryk1⟩	MOD	⟨udtryk2⟩	resultat
reel		reel	reel
reel		int	reel
int		reel	reel
int		int	int

3. Se også 'DIV' operatoren.

Type:

Kommando

Formål:

At slette datamatens arbejdslager og klargøre COMAL-80 systemet til et nyt program.

Syntaks:

NEW

Udførelse:

Det gemte program (hvis noget), og alle variable fra en tidligere programudførelse slettes og pladsen frigives til et nyt program.
Desuden udføres handlinger svarende til følgende program:

```
10 CLOSE
20 SELECT OUTPUT "DS:"
30 TRAP ERR+
40 TRAP ESC+
```

og systemfunktionerne 'ERR()' og 'ESC()' har derfor værdien 0.

Eksempel:

```
NEW
```

Kommentarer:

1. 'NEW' kommandoen bør altid anvendes før et nyt program indtastes eller indlæses.

2. Systemvariable ('KEYWORDLOWER', 'IDENTIFIERLOWER', 'INDENTION', 'PAGEWIDTH', 'PAGELENGTH' og 'ZONE' påvirkes ikke af denne kommando.

3. Se også kommentar 2 til 'ENTER' kommandoen.

Type:
Logisk operator

Formål:
At negere en logisk værdi.

Syntaks:
NOT (udtryk)

Udførelse:
Den logiske værdi af (udtryk) negeres.

Eksempel:
```
100 IF NOT ERR THEN EXEC READ_OK
```

Kommentarer:
1. Operatoren har følgende sandhedstabel:

(udtryk)	resultat
sand	falsk
falsk	sand

Type:

Instruktion

Formål:

Udfra værdien af et aritmetisk udtryk at vælge et af en række linienumre.

Syntaks:

ON (udtryk) GOTO (liste af linienumre)
ON (udtryk) GOSUB (liste af linienumre)

Udførelse:

(udtryk) beregnes og afrundes om nødvendigt til et heltal. I (liste af linienumre) vælges derefter det hertil svarende linienummer, således at (udtryk) = 1 svarer til første linienummer fra venstre; (udtryk) = 2 svarer til andet linienummer osv.

Eksempel:

```
10 INPUT "SKRIV ET TAL MELLEM 1 OG 3 INCL? ": TAL
20 ON TAL GOTO 40,60,80
30 GOTO 10
40 PRINT "DU SKREV 1"
50 GOTO SLUT
60 PRINT "DU SKREV 2"
70 GOTO SLUT
80 PRINT "DU SKREV 3"
90 LABEL SLUT
```

Kommentarer:

1. I modsætning til 'GOTO' instruktionen kan der ikke anvendes navne i 'ON...GOTO' instruktionen.

2. Hvis den afrundede værdi af (udtryk) ikke opfylder uligheden:

   1 (= (udtryk) (= antal linienumre

   overspringes instruktionen, og der fortsættes med næste udførbare instruktion.

3. I 'ON...GOSUB' instruktionen skal hvert linienummer i (liste af linienumre) være den første instruktion i en subrutine, som slutter med en 'RETURN' instruktion. Når denne mødes fortsætter programafviklingen i første udførbare linie efter 'ON...GOSUB' instruktionen.

4. Se også 'GOSUB' instruktionen.

Type:

Instruktion, kommando

Formål:

At åbne en datafil.

Syntaks:

OPEN FILE ⟨filnummer⟩,⟨filnavn⟩,⟨type⟩[,⟨blokstørrelse⟩]

Udførelse:

Ved filer af type 'WRITE' undersøges, at det angivne ⟨filnavn⟩ ikke allerede findes på baggrundslageret. Er dette tilfældet, standser programafviklingen med en fejlmeddelse, ellers oprettes filen.
Ved filer af type 'READ', 'APPEND' eller 'RANDOM' undersøges, om det angivne ⟨filnavn⟩ findes på baggrundslageret. Er dette ikke tilfældet, gives ved type 'READ' en fejlmeddelelse og ved type 'RANDOM' og 'APPEND' oprettes filen.
Derefter sammenknyttes ⟨filnavn⟩ og ⟨filnummer⟩ således, at alle referencer til ⟨filnavn⟩ sker med ⟨filnummer⟩ indtil filen igen lukkes med en 'CLOSE' instruktion eller kommando

Eksempel:

```
100 OPEN FILE 2,"TEST",WRITE
100 OPEN FILE 0,"DK1:DATA.RAN",RANDOM,40
```

Kommentarer:

1. ⟨filnummer⟩ er et aritmetisk udtryk som, evt. efter afrunding, skal antage en af værdierne 0, 1, 2, 3, 4, 5, 6, 7, 8 eller 9.

2. ⟨filnavn⟩ er et strengudtryk, som COMAL-80 tillader kan indeholde op til 80 karakterer, men kun de første 8 karakterer overføres til CP/M.

3. ⟨type⟩ angiver, hvorledes filen bruges. Der er følgende muligheder:

   READ der læses sekventielt fra filen
   WRITE der skrives sekventielt i filen
   RANDOM der både læses fra og skrives i filen
   APPEND tilføjer ny information til en allerede eksisterende fil af sekventiel type.

4. ⟨blokstørrelse⟩ anvendes kun ved filer af type 'RANDOM' og angiver det samlede antal byte, der skal udskrives eller læses i hver blok. Den nødvendige størrelse beregnes således:

   Heltal fylder 2 byte.
   Reelle tal fylder 4 byte ved 7 cifres nøjagtigned og 8 byte ved 13 cifres nøjagtighed.
   Strenge fylder 2 byte plus ligeså mange byte, som der er dimensioneret til.

5. Der kan være op til 8 filer åbne ad gangen. Dette muliggør, at der yderligere kan åbnes 2 ikke diskbaseret filer. Hvis diskfiler bruges i forbindelse med 'SELECT OUTPUT', 'LIST', 'SAVE', 'CAT', 'ENTER' eller 'LOAD' kan i visse tilfælde færre end 8 filer åbnes samtidig.
   En fil kan være åben under flere filnumre samtidig, under forudsætning af, at samme ⟨type⟩ specificeres.

6. En fil af type 'RANDOM' skal altid åbnes med samme ⟨blokstørrelse⟩, som den oprindelig blev åbnet med.
   ⟨blokstørrelse⟩ for en eksisterende fil kan bestemmes med programmet:

```
10 OPEN FILE 0,"⟨filnavn⟩.RAN",READ
20 READ FILE 0; BLOKSTØRRELSE#
30 PRINT BLOKSTØRRELSE#
40 CLOSE
```

Type:

Logisk operator

Formål:

At danne logisk 'eller' mellem 2 udtryk.

Syntaks:

⟨udtryk1⟩ OR ⟨udtryk2⟩

Udførelse:

⟨udtryk1⟩ og ⟨udtryk2⟩ beregnes og betragtes hver for sig som falske, hvis deres værdi er lig med 0 og ellers som sande. Logisk 'OR' dannes derefter mellem de 2 værdier.

Eksempel:

```
100 IF END_DATA1 OR END_DATA2 THEN EXEC END_DATA
```

Kommentarer:

1. Operatoren har sandhedstabellen:

⟨udtryk1⟩	⟨udtryk2⟩	resultat
sand	sand	sand
sand	falsk	sand
falsk	sand	sand
falsk	falsk	falsk

Type:

Aritmetisk funktion

Formål:

At omsætte et tegn i en tekststreng til dets decimale repræsentation.

Syntaks:

ORD((strengudtryk))

Udførelse:

Den decimale kode for første tegn i strengen (strengudtryk) bestemmes.

Eksempel:

```
10 DIM A$ OF 1
20 INPUT A$
30 PRINT ORD(A$)
```

Kommentarer:

1. Resultatet bliver heltallig og større end eller lig med 0 og mindre end eller lig med 255.

Type:

Maskinsprogsfunktion

Formål:

At sende en byte direkte til en af datamatens udgangsporte.

Syntaks:

OUT ⟨udtryk1⟩,⟨udtryk2⟩

Udførelse:

⟨udtryk1⟩ og ⟨udtryk2⟩ beregnes og afrundes om nødvendigt. Værdien af ⟨udtryk2⟩ sendes derefter til den af ⟨udtryk1⟩ fastlagte udgangsport.

Eksempel:

```
10 INPUT A
20 OUT 15,A
```

Kommentarer:

1. Værdien af ⟨udtryk1⟩ og ⟨udtryk2⟩ skal være et helt eller reelt tal større end eller lig med 0 og mindre end eller lig med 255.

2. Se også 'INP' funktionen.

Type:

Instruktion, kommando

Formål:

At fremføre papiret i en tilsluttet linieprinter til begyndelsen af næste side.

Syntaks:

PAGE

Udførelse:

Hvis systemvariablen 'PAGELENGTH' har værdien 0, sendes en form feed karakter til printeren. I alle andre tilfælde sendes line feed karakteren (0AH) indtil toppen af næste side nås.

Eksempler:

```
100 PAGE
    PAGE
```

Kommentarer:

1. Sideskift styres af en tæller i COMAL-80 når 'PAGELENGTH' er større end 0. I dette tilfælde er det derfor vigtigt, at papiret er korrekt isat printeren og ikke fremføres manuelt.

2. Længden af en side kan ændres med en 'PAGELENGTH' instruktion eller kommando.

Type:

System variabel

Formål:

At fastlægge antal linier per side på en tilsluttet linieprinter.

Syntaks:

PAGELENGTH

Udførelse:

En i COMAL-80 indbygget tæller husker hele tiden, hvormange linier der er udskrevet på den igangværende side. Dette tal benyttes, når en 'PAGE' instruktion eller kommando skal udføres, og 'PAGELENGTH' er forskellig fra 0, til at beregne, hvormange line feed karakterer der skal udsendes, for at fremføre papiret til næste side.

Eksempel:

```
100 PAGELENGTH:=72
100 PRINT PAGELENGTH
100 PAGELENGTH:=MAX_LINIER
    PAGELENGTH=88
```

Kommentarer:

1. Når COMAL-80 indlæses, tildeles 'PAGELENGTH' værdien 72.

2. En tildelt værdi skal være mellem 0 og 254 inklusive.

3. Dette nøgleord kan anvendes som operand eller kan tildeles til. Som operand er det af heltallig type.

4. Den øjeblikkelige værdi gælder for begge de printere, der kan være tilsluttet.

5. Værdien 0 stopper den interne tæller og oversættelsen af form feed karakterer til line feed karakterer. Det er herved muligt at sende form feed karakterer til printere, som er forsynet med denne mulighed.

6. 'NEW' kommandoen ændrer ikke værdien af systemvariablen 'PAGELENGTH'.

Type:

System variabel

Formål:

At fastlægge antal karakterer per linie på en tilsluttet linieprinter.

Syntaks:

PAGEWIDTH

Udførelse:

En i COMAL-80 indbygget tæller husker hele tiden, hvormange karakterer der er skrevet på den igangværende linie og sender en vognretur plus line feed når det højst tilladte antal karakterer er blevet udskrevet.

Eksempel:

```
100 PAGEWIDTH:=40
100 PRINT PAGEWIDTH
100 PAGEWIDTH:=MAX_KARAKTERER
    PAGEWIDTH=80
```

Kommentarer:

1. Når COMAL-80 indlæses, tildeles 'PAGEWIDTH' værdien 80.

2. En tildelt værdi skal være mellem 0 og 254 inklusive.

3. Dette nøgleord kan anvendes som operand eller kan tildeles til. Som operand er det af heltallig type.

4. Den øjeblikkelige værdi gælder for begge de printere, der kan være tilsluttet.

5. Værdien 0 stopper den automatiske udsendelse af vognretur og line feed.

6. 'NEW' kommandoen ændrer ikke værdien af systemvariablen 'PAGELENGTH'.

Type:

Maskinsprogsfunktion

Formål:

At undersøge værdien af en hukommelsesplads, bestemt ved et numerisk udtryk.

Syntaks:

PEEK(<udtryk>)

Udførelse:

<udtryk> beregnes og afrundes om nødvendigt. Værdien i den hertil svarende hukommelsesplads findes.

Eksempel:

```
 10 DIM B$ OF 1
 20 TRAP ESC-
 30 EXEC HENT_KARAKTER
 40 PRINT B$
 50 PROC HENT_KARAKTER
 60  // HENT TASTATURINPUT UDEN EKKO PÅ SKÆRMEN
 70  // 'ESC' TASTEN BEHANDLES SOM ENHVER ANDEN TAST.
 80  // 'TRAP ESC-' INSTRUKTIONEN SKAL UDFØRES FØR DENNE
 90  // PROCEDURE KALDES.
100  POKE 256,255
110  REPEAT
120   IF ESC THEN POKE 256,27
130  UNTIL PEEK(256)<>255
140  A$:=CHR$(PEEK(256))
150 ENDPROC HENT_KARAKTER
```

Kommentarer:

1. <udtryk> skal være et helt eller reelt tal større end eller lig med 0 og mindre end eller lig med 65535. Resultatet vil være af heltallig type og større end eller lig med 0 og mindre end eller lig med 255.

Type:

Maskinsprogsfunktion

Formål:

At sætte indholdet af en hukommelsesplads, bestemt ved et aritmetisk udtryk.

Syntaks:

POKE ⟨udtryk1⟩,⟨udtryk2⟩

Udførelse:

⟨udtryk1⟩ og ⟨udtryk2⟩ beregnes og afrundes om nødvendigt. Værdien af ⟨udtryk2⟩ gemmes derefter i den af ⟨udtryk1⟩ fastlagte hukommelsesplads.

Eksempel:

```
10 INPUT A
20 POKE 15,A
```

Kommentarer:

1. Værdien af ⟨udtryk1⟩ skal være et helt eller reelt tal større end eller lig med 0 og mindre end eller lig med 65535. Værdien af ⟨udtryk2⟩ skal være større end eller lig med 0 og mindre end eller lig med 255.

Type:

Aritmetisk funktion

Formål:

At undersøge, om en tegnstreng er indeholdt i en anden og, hvis det er tilfældet, hvor langt inde.

Syntaks:

POS(⟨strengudtryk1⟩,⟨strengudtryk2⟩)

Udførelse:

Karakter for karakter undersøges om ⟨strengudtryk1⟩ er indeholdt i ⟨strengudtryk2⟩. Er dette tilfældet, er resultatet et heltal, som angiver hvor mange karakterpositioner (regnet fra venstre mod højre) inde i ⟨strengudtryk2⟩, ⟨strengudtryk1⟩ begynder.

Eksempel:

```
10 DIM A$ OF 3
20 DIM B$ OF 6
30 A$:="AND"
40 B$:="VANDRE"
50 C#:=POS(A$,B$)
60 PRINT C#
```

Kommentarer:

1. Hvis ⟨strengudtryk1⟩ er en tom streng, giver 'POS' funktionen resultatet 1.

2. Hvis ⟨strengudtryk1⟩ ikke forekommer i ⟨strengudtryk2⟩ giver 'POS' funktionen resultatet 0.

3. Resultatet af 'POS' funktionen er af heltallig type.

Type:
Instruktion, kommando

Formål:
At udskrive data på udskriftenheden.

Syntaks:
PRINT [⟨liste af udtryk⟩]

Udførelse:
De i ⟨liste af udtryk⟩ indgående variable, talkonstanter og/eller tekststrenge udskrives på udskriftenheden.

Eksempel:
```
100 PRINT "RESULTATET ER: "; A
100 PRINT TAB(15); A, B
```

Kommentarer:

1. De enkelte elementer i ⟨liste af udtryk⟩ adskilles af komma eller semikolon.
   Adskilles to elementer med semikolon udskrives andet element umiddelbart efter første, dog indskydes et mellemrum efter et numerisk udtryk.
   Adskilles to elementer af komma, udskrives andet element ved starten af næste printzone.
   Bredden af printzonerne kan ændres med 'ZONE:=⟨aritmetisk udtryk⟩' udført som instruktion eller kommando, og hvor ⟨aritmetisk udtryk⟩ afrundes til et heltal)= 0 og (= 160.
   Reglerne for semikolon og komma gælder også efter sidste element i ⟨liste af udtryk⟩, idet virkningen videreføres på første element i næste printsætning.
   Når ⟨liste af udtryk⟩ afsluttes uden et komma eller et semikolon, afsluttes udførelsen af sætningen med skift til ny linie.
   Der skiftes også til ny linie, hvis ⟨liste af udtryk⟩ udelades.

2. Hvis den resterende plads på den aktuelle linie er for kort til at rumme næste element, udskrives dette i begyndelsen af næste linie.

3. Skift mellem udskriftenheder udføres med en 'SELECT OUTPUT' instruktion.

4. Med funktionen 'TAB(⟨udtryk⟩)', hvor ⟨udtryk⟩ er lig med antal karakterpositioner regnet fra venstre kant, kan tabuleres til en ønsket karakterposition.

5. Under programindtastning kan 'PRINT' erstattes af ';'. I programlistninger anvendes 'PRINT'.

Type:

Instruktion

Formål:

At udskrive data på ASCII form i en datafil.

Syntaks:

PRINT FILE ⟨filenummer⟩[,⟨postnummer⟩]:⟨liste af udtryk⟩

Udførelse:

De i ⟨liste af udtryk⟩ angivne udtryk udskrives som ASCII karakterer i den ved ⟨filnummer⟩ angivne fil. Er ⟨postnummer⟩ udeladt sker udskrivningen sekventielt; ellers i den ved ⟨postnummer⟩ angivne post.

Eksempel:

```
100 PRINT FILE 0,RECNO: A$, B, C+D

100 DIM A$ OF 5
110 A$:="##.##"
120 PRINT FILE 3: USING "##.##": A, B, C^2
130 PRINT FILE 4: USING A$: D
```

Kommentarer:

1. Inden 'PRINT (USING) FILE' instruktionen mødes, skal en fil være åbnet og forbindelse mellem pågældende filnavn og det i 'PRINT (USING) FILE' instruktionen benyttede ⟨filnummer⟩ være etableret med en 'OPEN FILE' instruktion eller kommando og type 'WRITE', 'APPEND' eller 'RANDOM'.

2. ⟨postnummer⟩ angives kun ved 'RANDOM' filer og er et aritmetisk udtryk, som, efter evt. afrunding til et heltal angiver nummeret på den aktuelle post.

3. ⟨filnummer⟩ er et numerisk udtryk.

4. Elementerne i ⟨liste af udtryk⟩ adskilles med komma eller semikolon som nærmere beskrevet under 'PRINT' og 'PRINT USING'.

5. 'PRINT FILE' og 'PRINT FILE USING' arbejder nøjagtigt som 'PRINT' og 'PRINT USING', idet den eneste forskel er, hvor udskrivningen sker.
   Syntaksen for 'PRINT FILE USING' findes ved i den ovenfor anførte syntaks at erstatte ⟨liste af udtryk⟩ med:
   USING ⟨strengudtryk⟩:⟨liste af udtryk⟩

6. Under programindtastning kan 'FILE' erstattes af '#'. I listninger anvendes 'FILE'.

7. Under programindtastning kan 'PRINT' erstattes med ';'. I listninger anvendes 'PRINT'.

Type:

Instruktion

Formål:

At udskrive tekststrenge og/eller data under anvendelse af et specificeret format.

Syntaks:

PRINT USING (strengudtryk) : (liste af udtryk)

Udførelse:

Den i (strengudtryk) angivne tekststreng overføres, karakter for karakter, til udskriftenheden, idet variable og/eller data fra (liste af udtryk) indsættes på de med nummertegnet '#' angivne steder.

Eksempler:

```
100 PRINT USING "RESULTATET ER ###.##": A

 10 DIM A$ OF 6
 20 A$:="##.###"
 30 PRINT USING A$: B
```

Kommentarer:

1. De enkelte tegn i (strengudtryk) virker således:
   '#' : Karakterposition og fortegn.
   '.' : Decimalpunktum, hvis omgivet af '#'.
   '+' : Foranstillet plus, hvis '#' følger umiddelbart efter.
   '-' : Foranstillet minus, hvis '#' følger umiddelbart efter.
   Alle andre tegn overføres uændret.

2. Begyndes et format med '+', afsættes plads til aritmetiske værdiers fortegn, og dette udskrives for både positive og negative værdier.

3. Begyndes et format med '-', afsættes plads til aritmetiske værdiers fortegn, men dette udskrives kun for negative værdier.

4. Ved alfanumeriske strenge virker foranstillet '+' og '-' som '#'.

5. Hvis en aritmetisk værdi består af for mange cifre til at finde plads i det angivne format, udfyldes positionen med '*'. Har den numeriske værdi flere decimaler, end formatet angiver, foretages en afrunding.

6. Alfanumeriske strenge venstrestilles i formatet. Er tegnstrengen for lang, afskæres det nødvendige antal tegn fra højre. Er tegnstrengen for kort, fyldes op efter strengen med blanktegn.

7. Når der ikke er flere udtryk i (liste af udtryk), standses afviklingen af 'PRINT USING' instruktionen. Hvis der er flere udtryk i (liste af udtryk) end angivet i (strengudtryk), bruges formaterne heri igen fra venstre.

8. Afsluttes 'PRINT USING' instruktionen med et semikolon, vil næste udskrift starte umiddelbart efter den igangværende. Afsluttes der med komma, vil næste udskrift starte i begyndelsen af næste printzone. Ellers afsluttes udførelsen af instruktionen altid med skift til næste linie.

9. 'PRINT USING' instruktionen kan anvendes til at skrive til en datafil efter nøjagtig samme regler som angivet under 'PRINT FILE' instruktionen.

10. Under programindtastning kan 'PRINT' indtastes som ';'. I listninger anvendes 'PRINT'.

Type:

Instruktion

Formål:

At definere et underprogram (en procedure).

Syntaks:

```
PROC (navn) [[REF] (variabel) [(dim)]] [CLOSED]
.
.
.
ENDPROC (navn)
```

Udførelse:

Når en 'PROC' instruktion mødes, overspringes programdelen til den tilhørende 'ENDPROC' instruktion og udføres først, når proceduren kaldes med en tilhørende 'EXEC' instruktion.

Eksempler:

```
10 PROC ERROR(N#) CLOSED
20  GLOBAL CC#, ERR_, ERRORS#
30  PRINT "*****";SPC$(CC#-9);"^";N#
40  ERR_:=FNINCLUDE(ERR_,N#+1); ERRORS#:+1
50 ENDPROC ERROR

Forskellige procedurehoveder:
10 PROC XYZ(A,B,REF C$) CLOSED
10 PROC ZYX(REF A#(,,), REF C(), D$)
10 PROC YZX(REF D$(,,), REF E#, REF C) CLOSED
```

Kommentarer:

1. 'PROC' instruktionen må ikke forekomme inden i følgende instruktioner:
   - betingede instruktioner
   - repeterende instruktioner
   - 'PROC' instruktioner
   - funktionserklæringer
2. (navn) skal være et legalt variabelnavn.
3. En procedure kan kalde andre procedurer og funktioner eller sig selv (rekursion).
4. (variabel) indeholder navnene på de formelle parametre, som ved kald af proceduren får overført værdier fra de aktuelle parametre i den tilhørende 'EXEC' instruktion.
5. De ændringer, der sker med en variabel i en procedure, er lokale, medmindre det med 'REF' angives, at værdien, når proceduren er færdigafviklet, skal føres tilbage til de aktuelle parametre i hovedprogrammet.
6. 'REF' kan angives for simple variable og skal angives for matricevariable.

7. En procedures type kan være enten reel, heltallig eller streng.

8. Ved matricevariable skal navnet efterfølges af en dimensionangivelse, som består af en parentes med et antal kommaer svarende til dimensionstallet minus 1.
   For en tredimensional matrice indeholder parentesen altså 2 kommaer, medens en vektor efterfølges af en tom parentes.

9. Hvis proceduren er erklæret lukket med 'CLOSED' instruktionen, er alle variabelnavne lokale og kan anvendes i anden betydning uden for proceduren. Dette kan ophæves for en eller flere variable med 'IMPORT' instruktionen.

10. Hvis programafsnittet mellem 'PROC' og 'ENDPROC' indeholder instruktioner over flere linier, skal disse i deres helhed være indeholdt i proceduren.

11. Der kan returneres fra en procedure både med 'ENDPROC (navn)' instruktionen, såvel som med 'RETURN' instruktionen.

12. Afsnit 'PROCEDURER' og 'PARAMETER INDSÆTTELSE' i kapitel 1 giver en mere dybtgående forklaring på disse nøgleord.

Type:

Instruktion, kommando

Formål:

At standse COMAL-80 systemet og vende tilbage til datamatens operativsystem.

Syntaks:

QUIT

Udførelse:

Under CP/M udføres en varmstart, hvorefter kontrollen overføres til CCP.

Eksempler:

```
100 QUIT
    QUIT
```

Type:

Instruktion, kommando

Formål:

At sætte et tilfældigt startpunkt for 'RND' funktionen.

Syntaks:

```
RANDOM
RANDOMIZE
```

Udførelse:

En i Z-80 CPUen indbygget tæller aflæses, og den fundne værdi bruges som grundlag for algoritmen, som ved kald af 'RND' funktionen leverer et tilfældigt tal.

Eksempler:

```
100 RANDOM
    RANDOM
```

Kommentrer:

1. 'RANDOM' og 'RANDOMIZE' kan anvendes efter behag, men i programudskrifter benyttes 'RANDOM'.

2. Den ovenfor nævnte tæller arbejder konstant, når CPUen er aktiv. Dens klokfrekvens er ca. 500 KHz, når CPUens klokfrekvens er 2,5 MHz.

3. Hvis 'RANDOM' ikke findes i et program, som kalder 'RND' funktionen, vil hver afvikling af programmet give den samme sekvens af tilfældige tal.

Type:

Instruktion

Formål:

At tildele værdier til variable fra en dataliste.

Syntaks:

READ (liste af variable)

Udførelse:

De enkelte elementer i (liste af variable) tildeles værdier fra datalisten. Denne tildeling sker i rækkefølge fra venstre mod højre.

Eksempel:

```
10 DIM FORNAVN$ OF 10
20 DIM EFTERNAVN$ OF 10
30 DATA "OLE", "JENSEN", 15
40 READ FORNAVN$, EFTERNAVN$
50 PRINT FORNAVN$+" "+EFTERNAVN$
60 READ ALDER
70 PRINT ALDER; " AR"
```

Kommentarer:

1. Hvis ikke den læste værdis type stemmer overens med den angivne variabels type, eller hvis datalisten er tømt, standser programafviklingen med en fejlmelding.

2. Tildeling af værdier til en strengvariabel følger de samme regler som beskrevet under 'LET' instruktionen.

3. Se også 'DATA' instruktionen.

Type:

Instruktion

Formål:

At indlæse data fra en binær datafil, som er udskrevet med 'WRITE FILE' instruktionen.

Syntaks:

READ FILE ⟨filnummer⟩ [,⟨postnummer⟩]:⟨liste af variable⟩

Udførelse:

Fra den med ⟨filnummer⟩ angivne fil tildeles værdier til de enkelte elementer i ⟨liste af variable⟩.

Eksempel:

```
100 READ FILE 5, REC_NO: A
100 READ FILE 3, A, B, C
```

Kommentarer:

1. Inden 'READ FILE' instruktionen mødes, skal en fil være åbnet og forbindelse mellem pågældende filnavn og det i 'READ FILE' instruktionen benyttede ⟨filnummer⟩ være etableret med en 'OPEN FILE' instruktion eller kommando og type 'READ' eller 'RANDOM'.

2. ⟨postnummer⟩ angives kun ved 'RANDOM' filer og er et numerisk udtryk, som evt. afrundes til et heltal.

3. ⟨filnummer⟩ er et numerisk udtryk.

4. ⟨liste af variable⟩ kan indeholde alle variabeltyper. Hvis der for en matricevariabel ikke angives indices, indlæses hele matricen.

5. Hvert element i ⟨liste af variable⟩ adskilles med komma.

6. Under indtastning kan 'FILE' og '#' anvendes efter behag. I programlistninger anvendes 'FILE'.

Type:

Instruktion

Formål:

At modtage variable som overføres fra en programdel til en anden i forbindelse med 'CHAIN' instruktionen.

Syntaks:

RECEIVE ‹liste af variable›

Udførelse:

Når 'CHAIN' instruktionen, som indlæser det program, der indeholder 'RECEIVE' instruktionen, udføres, gemmes værdierne af de i 'CHAIN' instruktionen angivne variable og den nye programdel indlæses.
'RECEIVE' instruktionen bruges derefter til at binde de gemte værdier sammen med de i den nye programdel anvendte variabelnavne.

Eksempel:

```
100 RECEIVE A, B, C
100 RECEIVE A$, B#, C
```

Kommentarer:

1. I sammenhørende 'CHAIN' og 'RECEIVE' instruktioner skal hver enkelt variabel være af samme type.

2. Variable som representerer matricer og strenge overfører deres dimension fra den gamle programdel til den nye, og må derfor ikke dimensioneres igen i den nye del.

Type:

Instruktion, kommando

Formål:

At undersøge om alle filer er lukkede.

Syntaks:

RELEASE [⟨enhed⟩]

Udførelse:

Det undersøges om alle filer på den angivne enhed er lukkede.

Eksempler:

```
100 RELEASE ""
100 RELEASE "DK1:"
100 RELEASE "DK"+DISK$+":"
    RELEASE
    RELEASE DK1:
```

Kommentarer:

1. Når CP/M er det anvendte operativsystem, benyttes ⟨enhed⟩ ikke; men angives det, skal det være navnet på et diskdrev.

2. Hvis en fil er åben, afbrydes programudførelsen og en fejlmelding gives.

3. ⟨enhed⟩ skal angives, når 'RELEASE' anvendes som instruktion, men kan være en tom streng.

Type:

Instruktion

formål:

At muliggøre indsættelse af forklarende tekst i et COMAL-80 program.

Syntaks:

```
//
REM
!
```

Udførelse:

'REM' instruktioner overspringes under programafviklingen.

Eksempel:

```
10 //PROGRAM TIL BEREGNING
20 REM AF POLYNOMIER.
30 ! 30/10/1980 MP
40 OPEN FILE 4,"TEST",READ // KLARGØR DATAFIL
```

Kommentarer:

1. Under programindskrivningen kan valgfrit skrives 'REM', '//' eller '!', men i programudskrifter benyttes altid '//'.

2. Alle instruktioner kan efterfølges af en kommentar.

Type:

Instruktion, kommando

Formål:

At ændre navnet for en fil på baggrundslageret.

Syntaks:

RENAME ⟨gl. navn⟩,⟨nyt navn⟩

Udførelse:

Datamatens operativsystem kaldes med oplysninger om det gamle og det nye filnavn.

Eksempler:

```
220 RENAME "DK1:FIL.CML","DK1:FIL.BAK"
    RENAME FIL.CML,FIL.BAK
    RENAME DK0:FIL.CML,DK0:FIL.BAK
```

Kommentarer:

1. ⟨gl. navn⟩ skal eksistere på den angivne enhed.

2. Hvis ingen enhed angives, udføres instruktionen på standardenheden.

3. Hvis ⟨nyt navn⟩ i forvejen findes, udskrives meddelelse herom, og kommandoen/instruktionen udføres ikke.

4. Hvis der i et af navnene indgår en enhedsangivelse, skal den samme enhedsangivelse indgå i det andet navn.

Type:

Kommando

Formål:

At omnummerere programlinier samt at flytte programområder.

Syntaks:

RENUM [[⟨startlinie⟩:⟨slutlinie⟩,]⟨start⟩[,⟨spring⟩]]

Udførelse:

Hvis et område af programmet skal flyttes, undersøges først, om der er plads til det ved ⟨startlinie⟩ og ⟨slutlinie⟩ angivne område på det ved ⟨start⟩ angivne sted og med den ved ⟨spring⟩ angivne afstand mellem linienumrene. Er dette ikke tilfældet standses med en fejlmelding. Ellers udregnes de nye linienumre og gemmes. Programmet gennemgås derefter, og alle referencer ('GOTO', 'GOSUB' osv.) ajourføres. Til sidst slettes de gamle linienumre.

Eksempler:

```
RENUM
RENUM 15
RENUM 15,3
RENUM 20:90,310,1
```

Kommentarer:

1. Hvis ⟨spring⟩ ikke angives, benyttes værdien 10.

2. Hvis ⟨start⟩ ikke angives, benyttes værdien 10.

3. ⟨startlinie⟩ og ⟨slutlinie⟩ anvendes, når kun en del af et program skal omnummereres og angiver henholdsvis første og sidste linie i området. I dette tilfælde angiver ⟨start⟩ det første nye linienummer og ⟨spring⟩ den nye afstand mellem de flyttede linier. På denne måde kan en programsektion til enhver tid flyttes til et nyt sted, hvis der her er et tilstrækkeligt antal frie linienumre. Blanding og overskrivning er ikke tilladt og et sådant forsøg afvises med en fejlmelding.

4. Hvis ⟨startlinie⟩ og ⟨slutlinie⟩ ikke angives, omnummereres hele programmet.

Type:

Instruktion

Formål:

At gentage udførelsen af en programdel, indtil en i 'UNTIL' instruktionen indbygget betingelse er opfyldt.

Syntaks:

```
REPEAT
.
.
.
UNTIL (logisk udtryk)
```

Udførelse:

Når 'UNTIL' instruktionen mødes, beregnes værdien af (logisk udtryk). Er denne sand, fortsætter programafviklingen i første udførbare instruktion efter 'UNTIL' instruktionen. Er (logisk udtryk) falsk, fortsættes i første udførbare instruktion efter den tilhørende 'REPEAT' instruktion.

Eksempel:

```
 10 DIM A$ OF 1
 20 DIM B$ OF 25
 30 PRINT "PROGRAMMET STANDSES VED "
 40 PRINT "TRYK PA 'ESC'-TASTEN"
 50 TRAP ESC-
 60 REPEAT
 70  INPUT "SKRIV ET BOGSTAV: ": A$
 80  B$:=B$+A$
 90 UNTIL ESC
100 PRINT "DU SKREV: ";B$
```

Kommentarer:

1. En programdel omgivet af 'REPEAT...UNTIL' udføres mindst en gang.

Type:

Instruktion

Formål:

At flytte datalistens pointer og derved muliggøre hel eller delvis genlæsning af datalisten.

Syntaks:

```
RESTORE
RESTORE (linienummer)
RESTORE (navn)
```

Udførelse:

Datalistens pointer stilles til at udpege første konstant i programmet, i det angivne (linienummer) eller efter det specificerede (navn).

Eksempler:

```
 10 LABEL IGEN
 20 RESTORE DATA2
 30 READ X
 40 PRINT X
 50 LABEL DATA1
 60 DATA 47
 70 RESTORE DATA1
 80 READ X
 90 PRINT X
100 GOTO IGEN
110 LABEL DATA2
120 DATA -47
```

Kommentarer:

1. Hvis 'RESTORE' instruktionen indeholder et linienummer, skal den pågældende linie indeholde en 'DATA' instruktion.

2. Hvis 'RESTORE' instruktionen indeholder et navn, skal linien umiddelbart efter den tilsvarende 'LABEL' instruktion indeholde en 'DATA' instruktion.

3. Hvis 'RESTORE' instruktionen hverken indeholder et linienummer eller et navn, stilles pointeren til første konstant i programmets første 'DATA' instruktion.

Type:

Instruktion

Formål:

At afbryde en subrutine eller en procedure, eller at afbryde en brugerdefineret funktion og returnere funktionsværdien.

Syntaks:

```
RETURN                  (for procedurer og subrutiner)
RETURN (udtryk)         (for funktioner)
```

Udførelse:

Ved procedurer og subrutiner afbrydes udførelsen, og programafviklingen fortsætter i første linie efter den linie der kaldte proceduren eller subrutinen.
Ved funktioner afbrydes udførelsen af funktionen og funktionsværdien indsættes i udtrykket som forårsagede kaldet.

Eksempel:

```
10 FUNC X_Y_OPLØFTET(X,Y)
20  RETURN X^3/Y^2
30 ENDFUNC X_Y_OPLØFTET
40 I:=2
50 J:=3
60 RESULTAT:=X_Y_OPLØFTET(I,J)
70 PRINT RESULTAT
```

```
10 PRINT "HOVEDPROGRAM"
20 GOSUB 50
30 STOP
40 PRINT "SUBRUTINE"
50 RETURN
```

```
10 EXEC OPEN_FILE
20 PROC OPEN_FILE
30  IF A$="DEFAULT" THEN RETURN
40  OPEN FILE 3, "DK1:"+A$, READ
50 ENDPROC OPEN_FILE
```

Kommentarer:

1. I brugerdefinerede funktioner kan funktionsværdien kun returneres ved hjælp af 'RETURN' instruktionen. Hvis denne instruktion ikke findes, er funktionsværdien udefineret og en fejlmelding vil blive vist.

2. (udtryk) i 'RETURN' instruktionen skal være af samme type som funktionsnavnet. Eneste undtagelse er, at et heltalligt udtryk bliver accepteret i en funktion af reel type.

3. Indeni en procedure kan en 'RETURN' instruktion uden (udtryk) ikke anvendes til at returnere fra en subrutine. I hovedprogrammet kan en 'RETURN' instruktion kun anvendes til at returnere fra en subrutine.

Type:

Aritmetisk funktion

Formål:

At danne et tilfældigt tal.

Syntaks:

RND [()]
RND (udtryk1),(udtryk2)

Udførelse:

Et tilfældigt tal i det lukkede interval fra (udtryk1) til (udtryk2) dannes.

Eksempler:

```
100 A:=RND()
100 B:=RND(-5,17)
```

Kommentarer:

1. Enhver afvikling af et program giver den samme sekvens af tilfældige tal, med mindre der tidligere under programafviklingen er udført en 'RANDOM' instruktion.

2. Udelades de to grænseangivelser (udtryk1) og (udtryk2) dannes et tilfældigt reelt tal i det åbne interval 0 til 1.

3. Når grænser angives, afrundes disse om nødvendigt til heltal, ligesom resultatet altid vil være et heltal i det lukkede interval fra (udtryk1) til (udtryk2).

4. Under programmering kan paranteserne efter 'RND' udelades, hvis de er tomme. I programlistninger anvendes 'RND()'.

Type:

Aritmetisk funktion

Formål:

At omdanne et udtryk fra reel til heltalstype.

Syntaks:

ROUND((udtryk))

Udførelse:

(udtryk) afrundes og resultatet omdannes til heltalstype.

Eksempel:

```
10 INPUT A
20 B#:=ROUND(A)
30 PRINT B#
40 PRINT ROUND(5.72)
50 PRINT ROUND(-5.72)
```

Kommentarer:

1. Afrundingen udføres til nærmeste heltal. Hvis der er lige langt til 2 af disse, vælges den, der har den højeste absolutte værdi.

2. (udtryk) er af reel type. Resultatet er af heltalstype. Bemærk at heltal kan tildeles til en reel variabel.

3. Se også 'INT' og 'TRUNC' funktionerne.

Type:

Kommando

Formål:

At starte afviklingen af et program.

Syntaks:

RUN [(linienummer)]

Udførelse:

COMAL-80 bringes i en veldefineret udgangstilstand, som blandt andet lukker åbne filer og nulstiller variabelområdet.
Derefter undersøger et særligt prepass, om programmet indeholder strukturer (FOR...NEXT, LOOP...ENDLOOP osv.) samt henvisninger (EXEC, LABEL osv.), og den interne repræsentation af sådanne instruktioner udvides med oplysninger, som forøger arbejdshastigheden.
Herefter startes programafviklingen med det angivne (linienummer).

Eksempler:

```
RUN
RUN 230
```

Kommentarer:

1. Udelades (linienummer) startes med programmets laveste linienummer.

Type:

Kommando

Formål:

At gemme programmer på det magnetiske baggrundslager i den interne (binære) form, programmet har i datamatens arbejdslager.

Syntaks:

SAVE (filnavn)

Udførelse:

Datamatens operativsystem kaldes med oplysninger om (filnavn) samt hvilket område af arbejdslageret, der skal udskrives.

Eksempler:

```
SAVE TEST
SAVE DK1:TEST
```

Kommentarer:

1. Hvis et program skal kunne kaldes fra andre programmer ved hjælp af 'CHAIN' instruktionen, skal det lagres med 'SAVE' kommandoen.

2. Programmer, gemt med 'SAVE' kommandoen, kan indlæses igen med 'LOAD' kommandoen.

3. Det interne format kan være forskelligt for forskellige versioner af COMAL-80. Et program kan derfor ikke altid gemmes med 'SAVE' kommandoen under en version og indlæses med 'LOAD' kommandoen under en anden version. Programmer, der skal udveksles eller langtidsopbevares bør derfor gemmes med 'LIST' kommandoen.

4. Hvis (filnavn) allerede findes på den aktuelle enhed, meldes dette og brugeren får valget mellem at fortsætte og få slettet den gamle fil, eller standse.

5. Filnavnsudvidelsen 'CSB' leveres altid af COMAL-80 og kan ikke specificeres af brugeren.

6. Informationer om eventuelle 'EXTENSION', som var aktive da 'SAVE' kommandoen blev udført, gemmes også. Disse oplysninger undersøges, når en 'LOAD' eller 'CHAIN' instruktion udføres og en på dette tidspunkt manglende 'EXTENSION' er en fejl.

Type:

Instruktion, kommando

Formål:

At angive en ny standardenhed/fil for udskrifter fra 'PRINT' og 'PRINT USING' instruktioner.

Syntaks:

SELECT OUTPUT (strengudtryk)

Udførelse:

Interne pointere i COMAL-80 systemet omstilles til at udpege det angivne udskriftsted.

Eksempler:

```
    SELECT OUTPUT "LP:"
220 SELECT OUTPUT "LP0:"
220 SELECT OUTPUT "DK1:TEKST"
220 SELECT OUTPUT "TEKST"
220 SELECT OUTPUT "DS:"
```

Kommentarer:

1. Hver gang en programafvikling startes med 'RUN' kommandoen, vælges dataskærmen som standard udskrivningsenhed. Under programafviklingen kan en ny standardenhed specificeres ved hjælp af denne instruktion/kommando og et enhedsnavn eller et filnavn.
   Når programafviklingen standses, enten fordi programmet er færdigt, eller fordi det afbrydes undervejs, vælges dataskærmen igen som standardudskrivningsenhed.

Type:

Aritmetisk funktion

Formål:

At bestemme fortegnet for et aritmetisk udtryk.

Syntaks:

SGN((udtryk))

Udførelse:

(udtryk), som skal være aritmetisk, beregnes. Er værdien) 0 får udtrykket 'SGN((udtryk))' værdien 1. Er værdien = 0 er SGN(X)=0 og er værdien (0 er SGN(X)=-1

Eksempel:

```
10 INPUT "SKRIV ET TAL: ": A
20 ON SGN(A)+2 GOTO 30,50,70
30 PRINT "A(0"
40 STOP
50 PRINT "A=0"
60 STOP
70 PRINT "A)0"
80 STOP
```

Type:
Trigonometrisk funktion

Formål:
At beregne sinus af et aritmetisk udtryk.

Syntaks:
SIN(⟨udtryk⟩)

Udførelse:
Sinus til ⟨udtryk⟩, som regnes i radianer, beregnes.

Eksempel:
```
10 INPUT A
20 PRINT SIN(A)
```

Kommentarer:
1. ⟨udtryk⟩ er et numerisk udtryk af reel eller heltallig type. Resultatet er altid reelt.

Type:

Kommando

Formål:

At fortælle, hvor stor en del af datamatens arbejdslager, der er brugt.

Syntaks:

SIZE

Udførelse:

På terminalen udskrives, hvor stor en del af datamatens arbejdslager, der er brugt, hvor meget fri lagerplads der er tilbage, og hvor meget plads der er brugt til variable.

Eksempel:

SIZE

Kommentarer:

1. Alle opgivelser er i antal byte.

2. Pladsforbruget til variable gælder den sidst foretagne programafvikling og gælder kun for de variable, der reelt blev dimensioneret eller tildelt en værdi.

3. Størrelsen af COMAL-80 vises ikke.

Type:

Tegnfunktion

Formål:

At danne en tegnstreng, bestående af et antal blanktegn, hvor antallet bestemmes af et aritmetisk udtryk.

Syntaks:

SPC$((udtryk))

Udførelse:

Det aritmetiske udtryk beregnes og afrundes om nødvendigt. Det herved bestemte antal blanktegn indsættes i tegnstrengen.

Eksempel:

```
10 INPUT A
20 PRINT SPC$(3*5),A
```

Kommentarer:

1. (udtryk) skal være >= 0

Type:

Aritmetisk funktion

Formål:

At beregne kvadratroden af et aritmetisk udtryk.

Syntaks:

SQR((udtryk))

Udførelse:

Kvadratroden af (udtryk), som skal være større end eller lig med 0, beregnes.

Eksempel:

```
10 INPUT A
20 PRINT SQR(A)
```

Kommentarer:

1. (udtryk) skal være aritmetisk og af reel eller heltallig type. Resultatet er altid reelt.

3. Hvis (udtryk) er mindre end 0, standser programafviklingen med en fejlmelding. Hvis disse er frakoblet med en 'TRAP ERR-' instruktion, sættes systemvariablen 'ERR' sand og resultatet beregnes udfra udtrykket:

   SQR(ABS((udtryk)))

Type:
Instruktion

Formål:
At standse udførelsen af et program.

Syntaks:
STOP

Udførelse:
Programudførelsen standses og følgende udskrift vises på skærmen:

Stop i linie xxxx

hvor xxxx angiver 'STOP' instruktionens linienummer.

Eksempel:
```
540 STOP
```

Kommentarer:
1. 'STOP' instruktionen anvendes normalt til at standse programafviklingen et andet sted end i programmets sidste linie.

2. Programafviklingen kan genoptages med en 'CON' kommando.

Type:

Tegnfunktion

Formål:

At omsætte et aritmetisk udtryk til en tegnstreng.

Syntaks:

STR$((udtryk))

Udførelse:

Det aritmetiske udtryk beregnes og omsættes til en tegnstreng.

Eksempel:

```
10 DIM A$ OF 4
20 A$:=STR$(7*3.9)
30 PRINT A$
```

Type:

Aritmetisk funktion

Formål:

I forbindelse med 'PRINT' instruktioner at tabulere frem til den karakterposition, hvor næste udskrift skal begynde.

Syntaks:

TAB(⟨udtryk⟩)

Udførelse:

⟨udtryk⟩ beregnes og afrundes om nødvendigt til et heltal, som angiver næste printposition.

Eksempel:

```
10 INPUT A
20 PRINT TAB(A),"*"
30 GOTO 10
```

Kommentarer:

1. 'TAB(⟨udtryk⟩)' kan kun benyttes i forbindelse med 'PRINT' instruktioner.

2. ⟨udtryk⟩ regnes absolut fra udskriftenhedens venstre margen. Hvis sidste udskrift inden 'TAB(⟨udtryk⟩)' har passeret den ved ⟨udtryk⟩ angivne position, standser programafviklingen med en fejlmeddelelse.

3. ⟨udtryk⟩ skal være aritmetisk og større end eller lig med 1 og mindre end eller lig med det maksimale antal karakterer i bredden på udskriftenheden.

Type:

Trigonometrisk funktion

Formål:

At beregne tangens til et aritmetisk udtryk.

Syntaks:

TAN((udtryk))

Udførelse:

Tangens til (udtryk), som regnes i radianer, beregnes.

Eksempel:

```
10 INPUT A
20 PRINT TAN(A)
```

Kommentarer:

1. (udtryk) skal være af reel eller heltallig type. Resultatet er altid reelt.

Type:

Instruktion, kommando

Formål:

At ændre systemets normale reaktion på en ikke-fatal fejl.

Syntaks:

```
TRAP ERR-
TRAP ERR+
```

Udførelse:

Under normal programafvikling vil enhver fejl standse programafviklingen med en fejlmeddelelse. Der findes imidlertid et antal fejl, som kan behandles på en veldefineret måde.
I sådanne tilfælde kan programafbrydelser undgås ved, at der er blevet udført en 'TRAP ERR-' instruktion inden fejlen opstår. I stedet bliver systemvariablen 'ERR()' tildelt fejlens nummer. Denne værdi vil i alle tests blive betragtet som sand, fordi den er forskellig fra 0. Fejlen omgås derefter på forud defineret vis og programafviklingen fortsætter.

Eksempel:

```
10 INIT "", FILNAVN$
20 TRAP ERR-
30 OPEN FILE 0, "XPLOCOMM", READ
40 TRAP ERR+
50 IF NOT ERR() THEN
60  INPUT FILE 0: DEFAULT_FILNAVN$
70 ELSE
80  DEFAULT_FILNAVN$:="XPLOPROG"
90 ENDIF
100 CLOSE
```

Kommentarer:

1. Før hver programafvikling starter, tildeles systemvariablen 'ERR' værdien falsk (= 0). Hvis en 'TRAP ERR-' instruktion udføres, og en ikke-fatal fejl derefter opstår, fortsætter programafviklingen, men systemvariablen 'ERR()' tildeles fejlens nummer som værdi og benolder denne værdi indtil dens status undersøges.
   Umiddelbart efter at den er blevet brugt, tildeles 'ERR()' igen værdien falsk (= 0).
   Normalt gør COMAL-80 en variabel sand ved at tildele den værdien 1, men i dette tilfælde anvendes fejlens nummer.

2. Systemet går tilbage til normal fejlbehandling, når en 'TRAP ERR+' instruktion udføres.

3. Under programmering kan 'ERR' og 'ERR()' frit anvendes, men i programlistninger anvendes 'ERR()'.

Type:

Instruktion, kommando

Formål:

At ændre systemets normale reaktion til et tryk på 'ESC' tasten.

Syntaks:

```
TRAP ESC-
TRAP ESC+
```

Udførelse:

Under normal programudførelse undersøges det, før udførelsen af hver enkelt programlinie, om 'ESC' tasten er blevet nedtrykket. I bekræftende fald standses programafviklingen. Hvis en 'TRAP ESC-' instruktion er blevet udført, blokeres denne virkning, systemvariablen 'ESC()' tildeles i stedet værdien sand, og programafviklingen fortsætter.

Eksempel:

```
10 TRAP ESC-
20 REPEAT
30  PRINT "DER ER IKKE BLEVET TRYKKET PA ESC TASTEN"
40 UNTIL ESC
50 TRAP ESC+
60 PRINT "DER ER BLEVET TRYKKET PA ESC TASTEN"
```

Kommentarer:

1. Før hver programafvikling starter, tildeles systemvariablen 'ESC()' værdien falsk (= 0). Hvis en 'TRAP ESC-' instruktion udføres og 'ESC' tasten derefter nedtrykkes, fortsætter programafviklingen, men systemvariablen 'ESC' tildeles værdien sand (= 1) og beholder denne værdi, indtil dens status undersøges.
   Umiddelbart efter at den er blevet brugt, tildeles 'ESC()' igen værdien falsk (= 0).

2. Systemet går tilbage til normal behandling af 'ESC' tasten når en 'TRAP ESC+' instruktion er blevet udført.

3. Under programmering kan 'ESC' og 'ESC()' frit anvendes, men i programlistningen anvendes 'ESC()'.

Type:
Systemkonstant

Formål:
Hovedsagelig at tildele en boolesk variabel værdien sand.

Syntaks:
TRUE

Udførelse:
Returnerer værdien 1.

Eksempel:

```
 10 // PRIMTALPROGRAM
 20 //
 30 DIM FLAGS#(0:8190)
 40 SIZE1:=8190
 50 //
 60 COUNT:=0
 70 MAT FLAGS#:=TRUE
 80 //
 90 FOR I:=0 TO SIZE1 DO
100  IF FLAGS#(I) THEN
110   PRIME:=I+I+3
120   K:=I+PRIME
130   WHILE K<=SIZE1 DO
140    FLAGS#(K):=FALSE
150    K:+PRIME
160   ENDWHILE
170   COUNT:+1
180  ENDIF
190 NEXT I
200 PRINT "PRIMTAL IALT: ", COUNT
```

Type:

Aritmetisk funktion

Formål:

At omdanne et reelt tal til et heltal.

Syntaks:

TRUNC(⟨udtryk⟩)

Udførelse:

⟨udtryk⟩ beregnes og omdannes til et heltal ved at bortkaste decimalerne.

Eksempel:

```
10 INPUT A
20 B#:=TRUNC(A)
30 PRINT B#
40 PRINT TRUNC(5.72)
```

Kommentarer:

1. ⟨udtryk⟩ er af reel type.
   Resultatet er af heltalstype.

2. Se også 'ROUND' og 'INT' funktionerne.

Type:
Instruktion, kommando

Formål:
At udpege en magnetisk baggrundsenhed som standardenhed.

Syntaks:
UNIT <enhed>

Udførelse:
Interne pointere omstilles til at udpege den angivne enhed.

Eksempler:
```
100 UNIT "DK1:"
    UNIT DK1:
```

Kommentarer:
1. <enhed> angives med to bogstaver, som angiver det magnetiske baggrundslagers type, og ennedsnummeret efterfulgt af kolon.

Type:

Aritmetisk udtryk

Formål:

At omsætte et reelt tal, som forefindes som en tegnstreng, til et reelt tal.

Syntaks:

VAL((strengudtryk))

Udførelse:

Det reelle tal i (strengudtryk) omsættes til et tal af reel type.

Eksempel:

```
10 DIM A$ OF 5
20 INPUT "SKRIV ET TAL: ": A$
30 B:=VAL(A$)
40 PRINT B
```

Kommentarer:

1. Hvis der i (strengudtryk) findes andre karakterer, end dem der normalt indgår i et tal, standser programafviklingen med en fejlmelding.

2. Se også 'IVAL' funktionen.

Type:

Maskinsprogsfunktion

Formål:

At bestemme den absolutte adresse i hukommelsen, hvor en variabel er gemt.

Syntaks:

VARPTR ((variabelnavn))

Udførelse:

Den decimale, absolutte adresse i hukommelsen, hvor første byte af variablen (variabelnavn) er gemt, bestemmes.

Eksempel:

```
10 INPUT A
20 PRINT VARPTR(A)
```

Kommentarer:

1. Adressen angiver, hvor første byte af variablen gemmes. De resterende byte ligger i rækkefølge på de følgende pladser.
   Heltal fylder 2 byte, hvor laveste del af tallet ligger først.
   Reelle tal fylder 4 byte ved 7 cifres nøjagtighed.
   Reelle tal fylder 8 byte ved 13 cifres nøjagtighed.
   Ved strengvariable angiver de 2 første byte længden og derefter gemmes strengen i rækkefølge.

2. Resultatet er af reel type.

3. (variabelnavn) kan være en matrice med eller uden indices. Hvis indices ikke angives, bliver resultatet lig med adressen på starten af første element.

4. Advarsel: I et tilfælde flyttes en variabel efter at den er blevet tildelt hukommelsesplads, og dens adresse ændres derfor. Dette sker, når der returneres fra en ikke lukket procedure, for alle de variable, der ved det aktuelle kald af proceduren er blevet tildelt plads for første gang.

Type:

Instruktion

Formål:

At gentage en programdel, så længe en i 'WHILE' instruktionen indbygget betingelse er sand.

Syntaks:

```
WHILE <logisk udtryk>
.
.
.
ENDWHILE
```

Udførelse:

Når 'WHILE' instruktionen mødes, beregnes værdien af <logisk udtryk>. Er denne værdi sand (<> 0), udføres de følgende sætninger til 'ENDWHILE' instruktionen, hvorefter programafviklingen fortsætter med 'WHILE' instruktionen, og ovennævnte forløb gentages til <logisk udtryk> er falsk.
Er <logisk udtryk> falsk (= 0) fortsætter programafviklingen i første linie efter 'ENDWHILE' instruktionen.

Eksempel:

```
10 OPEN FILE 0,"DATA",READ
20 WHILE NOT EOF(0) DO
30  READ FILE 0: INDEX, NUMBER#, TEXT$
40 ENDWHILE
```

Type:

Instruktion

Formål:

At udskrive data på binær form i en datafil.

Syntaks:

WRITE FILE ⟨filnummer⟩ [,⟨postnummer⟩]: ⟨liste af variable⟩

Udførelse:

Til den med ⟨filnummer⟩ angivne fil udskrives værdierne for de i ⟨liste af variable⟩ angivne variable.

Eksempler:

```
100 WRITE FILE 7,REC_NO: A, B, C
100 WRITE FILE 3: A$, B#, C
```

Kommentarer:

1. Inden 'WRITE FILE' instruktionen mødes, skal en fil være åbnet, og forbindelse mellem pågældende filnavn og det i 'WRITE FILE' instruktionen benyttede ⟨filnummer⟩ være etableret med en 'OPEN FILE' instruktion eller kommando og type 'WRITE', 'APPEND' eller 'RANDOM'.

2. ⟨postnummer⟩ angives kun ved 'RANDOM' filer og er et aritmetisk udtryk, som evt. afrundes til et heltal.

3. ⟨filnummer⟩ er et aritmetisk udtryk.

4. ⟨liste af variable⟩ kan indeholde alle variabeltyper. Hvis der for en matricevariabel ikke angives indices, udskrives hele matricen.

5. Hvert element i ⟨liste af variable⟩ adskilles af komma.

6. Under indtastning kan 'FILE' og '#' anvendes efter behag. I programlistninger anvendes altid 'FILE'.

Type:

Systemvariabel

Formål:

At tildele systemvariablen 'ZONE' bredden af en printzone i antal karakterer.

Syntaks:

ZONE:=(aritmetisk udtryk)

Udførelse:

(aritmetisk udtryk) beregnes og afrundes om nødvendigt. Den fundne værdi tildeles derefter til 'ZONE'.

Eksempler:

```
100 ZONE:=17
100 ZONE:=A*B
    ZONE=12
```

Kommentarer:

1. Ved indlæsning af COMAL-80 systemet, tildeles 'ZONE' værdien 0. Denne værdi kan kun ændres ved at udføre en 'ZONE' instruktion eller kommando.

2. 'NEW' kommandoen ændrer ikke værdien af 'ZONE' system-variablen.

3. Se også 'PRINT' instruktionen.

OVERSIGT OVER FEJLMELDINGER

FEJL	TEKST
1	Lagerplads opbrugt
2	Syntaksfejl
3	Overløb
4	Ikke $/# her
5	Kun for strenge
6	Fejl i kommando
7	Ikke flere nye navne !
8	Ikke afsluttet streng
9	Ulovligt tegn
10	Ulovligt tegn
11	Ulovligt linienummer
12	For lang linie
13	Variabel forventet
14	')' forventet
15	Typekonflikt
16	For kompliceret udtryk
17	'(' forventet
18	Typekonflikt i parameter
19	Har ingen parametre
20	Forkert type
21	',' forventet
22	TAB ikke tilladt her
23	Operand forventet
24	Konstant forventet
25	':' forventet
26	Funktion ikke tilladt her
27	:=/:+/:-/= brugt forkert
28	:=/:+/:- forventet
29	';'ikke tilladt her
30	'FILE' forventet
31	Linie-slut her ?
32	Ukendt I/U-enhed
33	Et navn forventet
34	Se manualen
35	'OF' forventet
36	Ikke streng-funktion
37	Linie-nummer forventet
38	GOTO/GOSUB forventet
39	Ikke efter 'THEN'
40	Se manualen
41	Tabel ikke tilladt
42	TO/DOWNTO forventet
43	READ/WRITE/APPEND/RANDOM forventet
44	Fra)= til
45	Linieslut forventet

46 Programsætning forventet
47 Kommando forventet
48 Fejl i program-struktur
49 Typekonflikt
50 Fejl i program-struktur
51 Dobbeltdefineret
52 Typekonflikt i RETURN-sætning
53 Navne-konflikt med PROC/DEF
54 FOR-NEXT dybde
55 Ukendt linie-nummer
56 RESTORE: kun til data-sætning
57 IF uden ENDIF
58 CASE uden ENDCASE
59 PROC uden ENDPROC
60 FUNC uden ENDFUNC
61 REPEAT uden UNTIL
62 WHILE uden ENDWHILE
63 FOR uden NEXT
64 Ukendt PROC/FUNC/LABEL
65 For kompliceret program-struktur
66 Fejl ved udskrivning på log
67 Index-fejl
68 Ulovligt postnummer
69 Ikke delstreng her
70 For få indices
71 For mange indices
72 Ikke flere data
73 Fejl i tildeling til delstreng
74 Kun for arrays
75 Fejl i USING-strengen
76 Ulovlig TAB-værdi
77 Variablen findes allerede
78 Kan ikke returnere
79 Kan ikke returnere (ulovlig GOTO/GOSUB)
80 CASE-værdi findes ikke
81 STEP = 0
82 SYSTEMFEJL
83 SYSTEMFEJL
84 Ude af definitionsområdet
85 For lang
86 Overløb
87 Udefineret variabel
88 For lang
89 Ikke kald af FUNC som kommando
90 Index-fejl
91 Typekonflikt i parameter
92 For mange parametre
93 For få parametre

```
 94     Division med 0
 95     SYSTEMFEJL
 96     Typekonflikt
 97     For lang linie
 98     Ikke nu
 99     Fejl i NEXT
100     ':'ikke tilladt her
101     Ingen linier med angivet nr.
102     Umuligt
103     Umuligt
104     Umuligt
105     Auto overløb
106     Ikke nu
107     Save'd under uforligelig COMAL-version
108     Tabeller skal have REF
109     Parameteren skal være en variabel
110     Parameteren har forkert dimension
111     EXIT uden LOOP
112     LOOP uden ENDLOOP
113     Umuligt
114     Umuligt
115     Ikke nu
116     PROC/FUNC skal være både kaldt og CLOSED
117     Delstreng af FUNC-kald ikke tilladt
118     Kan ikke kaldes fra en CLOSED PROC/FUNC
119     Skal returnere med 'RETURN'
120     Fejl i brug af "extension"
121     Fejl i brug af "extension"
122     Fejl i brug af "extension"
123     Fejl i brug af "extension"
124     Fejl i brug af "extension"
125     Fejl i brug af "extension"
126     Fejl i brug af "extension"
127     Fejl i brug af "extension"
128     Fejl i brug af "extension"
129     Fejl i brug af "extension"
130     Allerede eksisterende "extension"
131     Manglende "extension"
132     Bruger en "EXTENSION" forkert
133     For mange "extensions"
134     Ulovlig værdi for en systemvariabel
135     For mange variable i RECEIVE
136     Typekonflikt
137     Fejl i initialiseringsfil
138     Kanalen er allerede åben
139     Kanalen er ikke åben
140     Ulovligt kanalnummer
141     Ukendt I/U-enhed
142     Ukendt I/U-enhed
```

143 Fejl i filnavn
144 SYSTEMFEJL
145 SYSTEMFEJL
146 SYSTEMFEJL
147 Filtype ikke tilladt her
148 Umuligt
149 Umuligt
150 Fejl i initialiseringsfil
151 Kanalen er allerede åben
152 Kanalen er ikke åben
153 Ulovligt kanal-nummer
154 Ukendt I/U-enhed
155 Ukendt I/U-enhed
156 Fejl i filnavn
157 SYSTEMFEJL
158 SYSTEMFEJL
159 SYSTEMFEJL
160 Fil-type ikke tilladt her
161 Umuligt
162 Umuligt
163 SYSTEMFEJL
164 Kan ikke skrive
165 Kan ikke læse
166 Allerede åben i en anden måde
167 Filen er i brug
168 SYSTEMFEJL
169 Der kan ikke åbnes flere diskfiler nu
170 Filen findes ikke
171 SYSTEMFEJL
172 SYSTEMFEJL
173 SYSTEMFEJL
174 Umuligt: en fil er åben
175 SYSTEMFEJL
176 Simpel I/U-enhed
177 SYSTEMFEJL
178 SYSTEMFEJL
179 SYSTEMFEJL
180 Filkataloget er fuldt
181 Disken eller filen er fuld
182 SYSTEMFEJL
183 Forkert brug af filen
184 "End-Of-File"
185 Fejl ved lukning
186 SYSTEMFEJL
187 Forkert længde
188 SYSTEMFEJL
189 Læsefejl

190 SYSTEMFEJL
191 SYSTEMFEJL
192 SYSTEMFEJL
193 SYSTEMFEJL
194 SYSTEMFEJL
195 SYSTEMFEJL
196 SYSTEMFEJL
197 SYSTEMFEJL
198 SYSTEMFEJL
199 SYSTEMFEJL
200 Fejl i initialiseringsfil
201 Kanalen er allerede åben
202 Kanalen er ikke åben
203 Ulovligt kanal-nummer
204 Ukendt I/U-enhed
205 Ukendt I/U-enhed
206 Fejl i filnavn
207 SYSTEMFEJL
208 SYSTEMFEJL
209 SYSTEMFEJL
210 Filtype ikke tilladt her
211 Umuligt
212 Umuligt
213 SYSTEMFEJL
214 Kan ikke skrive
215 Kan ikke læse
216 Allerede åben i en anden måde
217 Filen er i brug
218 SYSTEMFEJL
219 Der kan ikke åbnes flere diskfiler nu
220 Filen findes ikke
221 SYSTEMFEJL
222 SYSTEMFEJL
223 SYSTEMFEJL
224 Umuligt: en fil er åben
225 SYSTEMFEJL
226 Simpel I/U-enhed
227 SYSTEMFEJL
228 SYSTEMFEJL
229 SYSTEMFEJL
230 Fil-kataloget er fuldt
231 Disken eller filen er fuld
232 SYSTEMFEJL
233 Forkert brug af filen
234 "End-Of-File"
235 Fejl ved lukning
236 SYSTEMFEJL
237 Forkert længde

238 SYSTEMFEJL
239 Læsefejl
240 SYSTEMFEJL
241 SYSTEMFEJL
242 SYSTEMFEJL
243 SYSTEMFEJL
244 SYSTEMFEJL
245 SYSTEMFEJL
246 SYSTEMFEJL
247 SYSTEMFEJL
248 SYSTEMFEJL
249 SYSTEMFEJL
250 SYSTEMFEJL
251 SYSTEMFEJL
252 SYSTEMFEJL
253 SYSTEMFEJL
254 SYSTEMFEJL
255 SYSTEMFEJL
256 SYSTEMFEJL
257 SYSTEMFEJL
258 Blokken er overskredet
259 Ulovlig bloklængde
260 Det er ikke en RANDOM-fil
261 Forkert bloklængde
262 Filen findes allerede
263 SYSTEMFEJL
264 SYSTEMFEJL
265 Fejl i filnavn
266 Forskellige I/U-enheder er angivet
267 SYSTEMFEJL
268 SYSTEMFEJL
269 SYSTEMFEJL
270 SYSTEMFEJL
271 SYSTEMFEJL
272 SYSTEMFEJL
273 SYSTEMFEJL
274 SYSTEMFEJL
275 SYSTEMFEJL
276 SYSTEMFEJL
277 SYSTEMFEJL
278 SYSTEMFEJL
279 SYSTEMFEJL
280 SYSTEMFEJL
281 SYSTEMFEJL
282 SYSTEMFEJL
283 SYSTEMFEJL

```
0010 // PRIMTALSPROGRAM
0020 //
0030 // BE OM ET NUMMER OG TEST DET
0040 //
0050 LOOP
0060   INPUT "POSITIV HELTAL SOM SKAL OMSÆTTES: ": HELTAL
0070   IF HELTAL)0 AND FRAC(HELTAL)=0 THEN EXIT //TEST FOR POSITIV
0080   //                                          HELTAL
0090   PRINT "JEG BAD OM ET POSITIVT HELTAL!"
0100 ENDLOOP
0110 PRINT "PRIMTALSFAKTORERNE ER: "
0120 //
0130 // PRIMTAL 2 OG 3 SKAL BEHANDLES SEPARAT
0140 //
0150 DIVISOR:=2
0160 EXEC TEST
0170 DIVISOR:=3
0180 EXEC TEST
0190 //
0200 //ALLE PRIMTAL KAN UDTRYKKES SOM
0210 //N*6+5 OG N*6+7
0220 //
0230 FOR N:=0 TO SQR(HELTAL)/6 DO
0240   DIVISOR:=6*N+5
0250   EXEC TEST
0260   DIVISOR:=6*N+7
0270   EXEC TEST
0280 NEXT N
0290 IF HELTAL()1 THEN PRINT HELTAL
0300 //
0310 PROC TEST
0320   WHILE HELTAL MOD DIVISOR=0 DO
0330     PRINT DIVISOR;
0340     HELTAL:=HELTAL DIV DIVISOR
0350   ENDWHILE
0360 ENDPROC TEST
```

```
0010 // KARAKTERSORTERINGSPROGRAM
0020 DIM STRENG$ OF 2000
0030 DIM KARAKTER$ OF 1
0040 DIM TÆLLER(ORD("A"):ORD("Å"))
0050 SPECIELLE_KARAKTERER:=0
0060 MELLEMRUM:=0
0070 TRAP ESC- // PAS PÅ. GEM PROGRAMMET
0080 //
0090 PRINT "INDTAST EN STRENG: ",
0100 LOOP
0110   EXEC HENT_KARAKTER(KARAKTER$) // HENT KARAKTERER EN FOR EN
0120   IF KARAKTER$=""27"" THEN EXIT
0130   PRINT KARAKTER$,
0140   STRENG$:+KARAKTER$ // SAMMENSÆT KARAKTERER
0150 ENDLOOP // "ESC" AFSLUTTER INPUT
0160 PRINT
0170 //
0180 FOR I:=1 TO LEN(STRENG$) DO
0190   KARAKTER$:=STRENG$(I)
0200   IF KARAKTER$=" " THEN MELLEMRUM:+1 // TEST FOR MELLEMRUM
0210   IF KARAKTER$)="A" AND KARAKTER$(="Å" THEN // BOGSTAV?
0220     TÆLLER(ORD(KARAKTER$)):+1 // TÆL BOGSTAVER
0230   ELSE
0240     SPECIELLE_KARAKTERER:+1 // TÆL ANDRE KARAKTERER
0250   ENDIF
0260 NEXT I // HENT NÆSTE KARAKTER
0270 // UDSKRIVNINGSFORMATET KLARGØRES
0280 FOR J:=ORD("A") TO ORD("Å") DO // SKRIV BOGSTAVERNE
0290   PRINT "  ",CHR$(J),
0300 NEXT J
0310 PRINT // TOM LINIE
0320 FOR K:=ORD("A") TO ORD("Å") DO // UDSKRIV ANTALLET
0330   PRINT USING " ##": TÆLLER(K),
0340 NEXT K
0350 PRINT
0360 PRINT
0370 PRINT "ANTAL KARAKTERER: ",LEN(STRENG$)
0380 PRINT
0390 PRINT "ANTAL SPECIELLE KARAKTERER INCLUSIVE MELLEMRUM: ",
0400 PRINT SPECIELLE_KARAKTERER
0410 PRINT
0420 PRINT "ANTAL SPECIELLE KARAKTERER EXCLUSIVE MELLEMRUM: ",
0430 PRINT SPECIELLE_KARAKTERER-MELLEMRUM
0440 PROC HENT_KARAKTER(REF A$) // BIBLIOTEKSRUTINE
0450   POKE 256, 255
0460   REPEAT
0470     IF ESC THEN POKE 256, 27
0480   UNTIL PEEK(256)()255
0490   A$:=CHR$(PEEK(256))
0500 ENDPROC HENT_KARAKTER
```

```
0010 // SKIFT GRUNDTAL
0020 // DETTE PROGRAM OVERSÆTTER ET POSITIVT HELTAL MED GRUNDTAL 10
0030 // TIL ET NYT GRUNDTAL MELLEM 2 OG 16
0040 DIM VÆRDI$(0:15) OF 1
0050 DIM DECIMAL(20)
0060 FOR I:=0 TO 15 DO
0070   //
0080   // KLARGØR KARAKTERSÆTTET DER SKAL BRUGES TIL UDSKRIVNING
0090   //
0100   READ VÆRDI$(I)
0110 NEXT I
0120 DATA "0", "1", "2", "3", "4", "5", "6", "7"
0130 DATA "8", "9", "A", "B", "C", "D", "E", "F"
0140 //
0150 // DET NYE GRUNDTAL HENTES OG TESTES
0160 //
0170 REPEAT
0180   INPUT "NYT GRUNDTAL: ": NY_BASE
0190 UNTIL 2(=NY_BASE AND NY_BASE(=16 AND FRAC(NY_BASE)=0
0200 //
0210 // BE OM NUMMERET SOM SKAL KONVERTERES
0220 //
0230 REPEAT
0240   INPUT "POSITIVT HELTAL SOM SKAL KONVERTERES: ": VÆRDI
0250   V:=VÆRDI
0260 UNTIL FRAC(VÆRDI)=0 AND VÆRDI)0
0270 //
0280 // OVERSÆT
0290 //
0300 I:=1
0310 REPEAT
0320   DECIMAL(I):=VÆRDI MOD NY_BASE; VÆRDI:=VÆRDI DIV NY_BASE
0330   I:+1
0340 UNTIL VÆRDI=0
0350 ANTAL_DECIMALER:=I-1
0360 //
0370 // UDSKRIV RESULTATET
0380 //
0390 PRINT V," GRUNDTAL 10 BLIVER I GRUNDTAL ",NY_BASE," TIL: ",
0400 FOR I:=ANTAL_DECIMALER DOWNTO 1 DO
0410   PRINT VÆRDI$(DECIMAL(I))," ",
0420 NEXT I
```

```
9978 // PROCEDURE TIL AT HENTE TASTATURINPUT
9979 // UDEN EKKO TIL SKÆRMEN
9980 // 'ESC' TASTEN VIRKER PÅ NORMAL MÅDE
9981 PROC HENT_KARAKTER(REF A$)
9982   POKE 256,255
9983   REPEAT
9984   UNTIL PEEK(256) <> 255
9985   A$:=CHR$(PEEK(256))
9986 ENDPROC HENT_KARAKTER
9987 //
9988 // PROCEDURE TIL AT HENTE TASTATURINPUT
9989 // UDEN EKKO TIL SKÆRMEN
9990 // 'ESC' TASTEN BEHANDLES SOM ENHVER ANDEN KARAKTER
9991 // 'TRAP ESC-' SKAL VÆRE UDFØRT INDEN DENNE
9992 // PROCEDURE KALDES
9993 PROC HENT_KARAKTER_ESC(REF A$)
9994   POKE 256,255
9995   REPEAT
9996     IF ESC THEN POKE 256,27
9997   UNTIL PEEK(256) <> 255
9998   A$:=CHR$(PEEK(256))
9999 ENDPROC HENT_KARAKTER_ESC
```

ASCII KARAKTERKODER

ASCII Kode	KARAKTER	ASCII Kode	KARAKTER	ASCII Kode	KARAKTER
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	Æ
006	ACK	049	1	092	Ø
007	BEL	050	2	093	Å
008	BS	051	3	094	^
009	HT	052	4	095	_
010	LF	053	5	096	'
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	[	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESC	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	VS	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	]	080	P	123	æ
038	&	081	Q	124	ø
039	'	082	R	125	å
040	(	083	S	126	
041	)	084	T	127	DEL
042	*	085	U		

ASCII koder er i decimal
LF=Line Feed, FF=Form Feed, CR=Carriage Return, DEL=Rubout

```
        NAME    ('EXAMPLE')
;
; This is an example of a file which defines "EXTENSIONS"
; to the METANIC COMAL-80 interpreter version 2.
; The file is intentioned for use with the MACRO-80 macro
; assembler from Microsoft.
;
; Version 1 written 830324
;                   by Arne Christensen
;          who is of Metanic ApS
;                    Byvej 11
;                    DK-3660 Stenlose
;                    Denmark
;

        .Z80
        .XLIST
;
; Macro definitions used when writing "EXTENSIONS" to the
; METANIC COMAL-80 interpreter version 2.
; The macro definitions are intentioned for use with the
; MACRO-80 macro assembler from Microsoft.

SETNAME MACRO   NAME
!&NAME  EQU     !NO
!NO     DEFL    !NO+1
        ENDM

SKIP    MACRO   AMOUNT
!NO     DEFL    !NO+AMOUNT
        ENDM

DEFO    MACRO   NAMES
        IRP     Y,<NAMES>
        SETNAME Y
Y       MACRO
        DB      !&Y
        ENDM
        ENDM
        ENDM

DEF1    MACRO   NAMES
        IRP     Y,<NAMES>
        SETNAME Y
Y       MACRO   PARAM
        DB      !&Y,PARAM
        ENDM
        ENDM
        ENDM
```

```
INO     DEFL    0
        DEFO    (ATN,COS,SIN,TAN,LOG,EXP,SQR,ESC,ERR,EOD)
        DEFO    (EOF,LEN,ORD,IVAL,VAL,INT,FRAC,TRUNC,ROUND)
        DEFO    (POS,BVAL,CHR,STR,I.STR,ERRTEXT,SGN,I.SGN)
        DEFO    (ABS,I.ABS,RND0,RND2,SPC,PEEK,INP,BSTR,VARPTR)
        SKIP    6
        DEFO    (FREEST)

        DEFO    (ENDEXPR,CONV1,CONV,REALINT,RLBL1,RLBL)
        SKIP    5
        DEF1    (INX)
        SKIP    2
        DEFO    (LDVAL)

        DEF1    (SYSVAR)
IX      DEFL    0
        IRP     Y,(ZONE,INDENT,PAGEWI,PAGELE,KWLOWER,IDLOWER)
Y       EQU     IX
IX      DEFL    IX+1
        ENDM

        SKIP    10

        SETNAME STRCON
STRCON  MACRO   STRING
        LOCAL   STREND
        DB      !STRCON
        DW      STREND-$-2
        DB      STRING
STREND:
        ENDM

        SKIP    1

        SETNAME INTCON
INTCON  MACRO   NUMBER
        DB      !INTCON
        DW      NUMBER
        ENDM

        DEFO    (TRUE,FALSE)
        SKIP    2
        DEFO    (CHS,POWER,TIMES,SLASH,DIV,MOD,PLUS,MINUS,IN)
        DEFO    (LEQ,LSS,GEQ,GTR,EQL,NEQ,B.AND,B.OR,B.NOT)
        DEFO    (S.PLUS,I.TIMES,I.CHS,I.DIV,I.MOD,I.PLUS,I.MINUS)
        DEFO    (I.LEQ,I.LSS,I.GEQ,I.GTR,I.EQL,I.NEQ)
        DEFO    (S.LEQ,S.LSS,S.GEQ,S.GTR,S.EQL,S.NEQ)
        DEFO    (UROUND)
        DEF1    (LOAD,STORE)
        DEFO    (STVAL)

EXPR    MACRO
        CALL    103H
        ENDM
```

```
INTREAL EQU     1
INT     EQU     2
REAL    EQU     3
STR     EQU     4
ANYTYPE EQU     9

ANYDIM  EQU     -3


DEFPRIORITY MACRO PRIORITY,NAMES
        IRP     Y,<NAMES>
II&Y    EQU     PRIORITY
        ENDM
        ENDM

        DEFPRIORITY 7,<ARROW>
        DEFPRIORITY 6,<TIMES,SLASH,DIV,MOD>
        DEFPRIORITY 5,<PLUS,MINUS,CHS>
        DEFPRIORITY 4,<LEQ,LSS,GEQ,GTR,EQL,NEQ,IN>
        DEFPRIORITY 3,<B.NOT>
        DEFPRIORITY 2,<B.AND>
        DEFPRIORITY 1,<B.OR>

EXTENSION MACRO NAME,USENAM
        LOCAL   NAMEEND
        IFB     <USENAM>
         DW     ?&NAME
         DS     1
         DW     NAME
        ELSE
         DW     ?&USENAM
         DS     1
         DW     USENAM
        ENDIF
        DB      NAMEEND-$-1
        DB      '&NAME'
NAMEEND:
        ENDM

OPERATOR MACRO  X1,X2,X3,X4
        IFB     <X4>    ;; UNARY OPERATOR
         DEFB   80H+X1,0,X2,II&X3
        ELSE
         DEFB   80H+X1,X2,X3,II&X4
        ENDIF
IOPER   DEFL    1
        ENDM

FUNCTION MACRO  TYPE
        DB      TYPE
IOPER   DEFL    0
        ENDM

STATEMENT MACRO
        DB      0
IOPER   DEFL    0
        ENDM
```

```
PARAMETER MACRO X1,X2
        IFB     <X2> ;; VALUE
         DB     -1,X1
        ELSE
         DB     X1,X2
        ENDIF
        ENDM

ENDEXT  MACRO   NAME,USENAM
        IFF     !OPER
         DB      -2
        ENDIF
        IFB     <USENAM>
?&NAME:
        ELSE
?&USENAM:
        ENDIF
        ENDM

ENDALLEXT MACRO NAME,USENAM
        IFF     !OPER
         DB     -2
        ENDIF
        IFB     <USENAM>
?&NAME   EQU    0
        ELSE
?&USENAM EQU    0
        ENDIF
        ENDM

        .LIST
        .SALL

;
; Comments are in a COMAL-80-like style, while
; routine headings are in a Pascal-like style.
;

;
; Interface descriptions for the "EXTENSIONS" defined in
; this file:
;
        EXTENSION BIT,BIT00
        OPERATOR  INT,INT,INTREAL,IN
        ENDEXT    BIT,BIT00

        EXTENSION SETBIT
        STATEMENT
        PARAMETER 0,INT
        PARAMETER INTREAL
        PARAMETER INTREAL
        ENDEXT    SETBIT

        EXTENSION COMPL
        OPERATOR  INT,INT,CHS
        ENDEXT    COMPL
```

```
        EXTENSION HEX
        FUNCTION  STR
        PARAMETER REAL
        ENDEXT    HEX

        EXTENSION LOWBOUND
        FUNCTION  INT
        PARAMETER ANYDIM,ANYTYPE
        PARAMETER INTREAL
        ENDEXT    LOWBOUND

        EXTENSION HIGHBOUND
        FUNCTION  INT
        PARAMETER ANYDIM,ANYTYPE
        PARAMETER INTREAL
        ENDEXT    HIGHBOUND

        EXTENSION MAXLENGTH
        FUNCTION  INT
        PARAMETER 0,STR
        ENDALLEXT MAXLENGTH


;
; OPERATOR BIT(NUMBER : INT; POS : REALINT) : INT
;
; Returns the value of bit number POS in NUMBER.
;

BIT00:
        LD      A,(IX+1)        ; IF POS > 15
        AND     0F0H
        OR      (IX+2)          ; OR POS < 0
        OR      (IX+0)          ; OR POS HAS OVERFLOWED
        JR      Z,BIT10         ; THEN
        LD      A,84            ;     A := 84; CY := 0; Z := 0
        AND     A
        JR      BIT90           ; ELSE
BIT10:  LD      A,(IX+1)        ;   IF POS < 8 THEN
        CP      8
        JR      NC,BIT20
        LD      B,A             ;     B := POS
        LD      A,(IX+3)        ;     A := LOW BYTE OF NUMBER
        JR      BIT30           ;   ELSE
BIT20:  SUB     8
        LD      B,A             ;     B := POS-8
        LD      A,(IX+3+1)      ;     A := HIGH BYTE OF NUMBER
BIT30:                          ;   ENDIF
        INC     B               ;   ROTATE A RIGHT B+1 TIMES
BIT40:  RRCA                    ;   CY := LAST BIT SHIFTED OUT
        DJNZ    BIT40
        LD      A,0
        ADC     A,A
        LD      B,A             ;   B := CY
        XOR     A               ;   A := 0; CY := 0; Z := 1
                                ; ENDIF
```

```
BIT90:
        INC     IX              ; // ADJUST IX
        INC     IX
        INC     IX
        LD      (IX+0),B        ; RETURN B,   A,CY,Z
        LD      (IX+1),0
        RET

;
; STATEMENT SETBIT(VAR VARIABLE : INT; POS, VALUE : INTREAL)
;
; Sets bit number POS in VARIABLE to VALUE. If VALUE <> 0
; the bit will be set; if VALUE = 0 the bit will be reset.
;
SETBIT:
        LD      A,(IX+3+1)      ; IF POS > 15
        AND     0F0H
        OR      (IX+3+2)        ; OR POS < 0
        OR      (IX+3)          ; OR POS HAS OVERFLOWED
                                ; THEN
        LD      A,84            ;   A := 84; CY := 0; Z := 0
        JR      NZ,SETB90       ; ELSE
        LD      A,(IX+0)        ;   SET := (VALUE HAS OVERFLOWED)
        OR      (IX+1)          ;           OR (VALUE <> 0)
        OR      (IX+2)
        PUSH    AF
        LD      L,(IX+3+3+2)    ;   HL := VARPTR(VARIABLE)
        LD      H,(IX+3+3+2+1)
        LD      A,(IX+3+1)      ;   A := POS
        CP      8               ;   IF A >= 8 THEN
        JR      C,SETB10
        SUB     8               ;     A :- 8; HL :+ 1
        INC     HL
SETB10:                         ;   ENDIF
        LD      B,A
        INC     B
        LD      A,80H
SETB20: RLCA
        DJNZ    SETB20
        LD      B,A             ;   B := 2^A
        POP     AF              ;   IF NOT SET THEN // CLEAR
        JR      NZ,SETB30
        LD      A,B
        CPL
        AND     (HL)
        LD      (HL),A          ;     (HL) := (HL) AND NOT B
        JR      SETB40          ;   ELSE
SETB30: LD      A,B
        OR      (HL)
        LD      (HL),A          ;     (HL) := (HL) OR B
SETB40:                         ;   ENDIF
        XOR     A               ;   A := 0; CY := 0; Z := 1
SETB90:                         ; ENDIF
        LD      DE,3+3+4        ; // ADJUST IX
        ADD     IX,DE
        RET
```

```
;
; OPERATOR COMPL(VALUE : INT) : INT
;
; Returns the one's complement of VALUE
;
COMPL:
        LD      A,(IX+0)        ; VALUE := ONE'S COMPLEMENT
        CPL                     ;          OF VALUE
        LD      (IX+0),A
        LD      A,(IX+1)
        CPL
        LD      (IX+1),A
        XOR     A               ; A := 0; CY := 0; Z := 1
        RET                     ; RETURN VALUE,  A,CY,Z

;
; FUNCTION HEX(VALUE : REAL) : STR
;
; Converts VALUE, which must be in the range 0 to 65535
; after rounding, to a four-digit hexadecimal number.
;
HEX:
        EXPR
        UROUND                  ; NUMBER := UROUND(VALUE)
        ENDEXPR                 ; A,CY,Z := ACCORDINGLY
        JR      Z,HEX10         ; IF Z=1 THEN
        LD      (IX+0),0
        LD      (IX+1),0        ;   RETURN "",  A,CY,Z
        RET                     ; ELSE
HEX10:  LD      HL,RESULT+2     ;   HL := VARPTR(RESULT)
        LD      A,(IX+1)
        CALL    HEX50           ;   EXEC HEX50(HIGH BYTE OF NUMBER)
        LD      A,(IX+0)
        CALL    HEX50           ;   EXEC HEX50(LOW BYTE OF NUMBER)
        INC     IX              ;   // ADJUST IX
        INC     IX
        EXPR
        INTCON  RESULT
        LOAD    STR             ;   RETURN RESULT,  A,CY,Z SET
        ENDEXPR                 ;                   ACCORDINGLY
        RET

;
; PROCEDURE HEX50(BYT : BYTE)
;
HEX50:
        PUSH    AF
        AND     0F0H
        RRCA
        RRCA
        RRCA
        RRCA
        CALL    HEX60           ; EXEC HEX60(BYT DIV 16)
        POP     AF
        AND     0FH             ; EXEC HEX60(BYT MOD 16)
        CALL    HEX60
        RET
```

```
;
; PROCEDURE HEX60(A : BYTE)
;
HEX60:  ADD     A,'0'           ; A :+ '0'
        CP      '9'+1           ; IF A ) '9' THEN A :+ -'0'-10+'A'
        JR      C,HEX70
        ADD     A,-'0'-10+'A'
HEX70:  LD      (HL),A          ; (HL) := A
        INC     HL              ; HL :+ 1
        RET

RESULT: DW      4               ; RESULT IS A STRING VARIABLE
        DS      4               ; WITH LENGTH 4


;
; FUNCTION
; LOWBOUND(VAR VARIABLE : ANYDIM&ANYTYPE ; INDEX : INTREAL) : INT
;
; Returns the lower bound on index number INDEX (1st, 2nd, 3rd etc.
; index) of the array VARIABLE.
;
LOWBOUND:
        LD      DE,-2
        JR      LOWHIGH         ; RETURN LOWHIGH(VARIABLE, INDEX, -2)

;
; FUNCTION
; HIGHBOUND(VAR VARIABLE : ANYDIM&ANYTYPE ; INDEX : INTREAL) : INT
;
; Returns the upper bound on index number INDEX (1st, 2nd, 3rd etc.
; index) of the array VARIABLE.
;
HIGHBOUND:
        LD      DE,-4
        JR      LOWHIGH         ; RETURN LOWHIGH(VARIABLE, INDEX, -4)


;
; FUNCTION LOWHIGH(VAR VARIABLE : ANYDIM&ANYTYPE : INDEX : INTREAL;
;                  OFFSET : INT) : INT
;
; UTILITY FUNCTION FOR LOWBOUND AND HIGHBOUND. OFFSET IS IN DE.
;
LOWHIGH:
        LD      L,(IX+3)        ; B := THE FIRST BYTE OF VARIABLE'S
        LD      H,(IX+3+1)      ;      DESCRIPTOR
        LD      B,(HL)
        BIT     7,B
        JR      Z,LOW10         ; IF B ( 0 THEN
        LD      A,74            ;   RETURN A = 74, CY = 1, Z = 0
        OR      A
        SCF
        RET
```

```
LOW10:  LD      A,(IX+0)        ; ELIF IF INDEX HAS OVERFLOWED
        OR      (IX+2)          ;   OR INDEX ( 0 OR INDEX ) 255
        JR      NZ,LOW20
        LD      A,(IX+1)        ;   OR INDEX = 0
        OR      A
        JR      Z,LOW20
        DEC     A               ;   OR INDEX-1 ) B
        CP      B
        JR      C,LOW30
        JR      Z,LOW30
LOW20:                          ; THEN
        LD      A,67            ;   RETURN A = 67, CY = 1, Z = 0
        OR      A
        SCF
        RET                     ; ELSE
LOW30:  LD      L,(IX+3)        ;   HL := POINTER TO
        LD      H,(IX+3+1)      ;       VARIABLE'S DESCRIPTOR
        LD      BC,-7
LOW40:  OR      A               ;   HL :- (INDEX-1)*7
        JR      Z,LOW50
        ADD     HL,BC
        DEC     A
        JR      LOW40
LOW50:  ADD     HL,DE           ;   HL :+ OFFSET
        LD      E,(HL)
        INC     HL
        LD      D,(HL)
        LD      BC,3+4-2        ;   // ADJUST IX
        ADD     IX,BC
        LD      (IX+0),E        ;   RETURN (HL),  A=0, CY=0, Z=1
        LD      (IX+1),D
        XOR     A
        RET                     ; ENDIF

;
; FUNCTION MAXLENGTH(VAR STRING : STR) : INT
;
; Returns the maximum (DIMensioned) length of the string variable
; STR.
;
MAXLENGTH:
        LD      L,(IX+0)        ; HL := POINTER TO STRING'S
        LD      H,(IX+1)        ;       DESCRIPTOR
        DEC     HL
        LD      D,(HL)
        DEC     HL
        LD      E,(HL)
        INC     IX              ; // ADJUST IX
        INC     IX
        XOR     A
        LD      (IX+0),E        ; RETURN (HL-2),  A=0,CY=0,Z=1
        LD      (IX+1),D
        RET

        END
```

```
0010 // This is the CONVERT program used in conjunction with
0020 // "EXTENSIONS" to the METANIC COMAL-80 interpreter.
0030 // The program is intentioned for use with the MACRO-80 macro
0040 // assembler from Microsoft.
0050 //
0060 // The program reads an object file produced by the MACRO-80
0070 // assembler and produces a corresponding relocatable file
0080 // which conforms to the format expected by COMAL-80.
0090 //
0100 // Version 1 written 830324
0110 //                  by Arne Christensen
0120 //          who is of Metanic ApS
0130 //                    Byvej 11
0140 //                    DK-3660 Stenlose
0150 //                    Denmark
0160 //
0170 FUNC STREAM#(N#) CLOSED // N# <= 8
0180   // Reads N# bits from the input file and returns them
0190   // as function value
0200   IMPORT BUFFER$
0210   DIM RESULT$ OF 8
0220   WHILE LEN(BUFFER$)<=N# DO
0230     READ FILE 0: B#
0240     BUFFER$:=BSTR$(PEEK(VARPTR(B#)))+BSTR$(PEEK(VARPTR(B#)+1))
0250   ENDWHILE
0260   RESULT$:="00000000"; RESULT$(8-N#+1:8):=BUFFER$(1:N#)
0270   BUFFER$:=BUFFER$(N#+1:LEN(BUFFER$))
0280   RETURN BVAL(RESULT$)
0290 ENDFUNC STREAM#
0300 //
0310 PROC PUT(BYTE#, FLAG#) CLOSED
0320   // Writes out BYTE# with relocation status FLAG#
0330   // to the output file
0340   IMPORT BYTES$, FLAGS#, LOC_MOD_8#, LOCATION#
0350   BYTES$(LOC_MOD_8#+1):=CHR$(BYTE#); FLAGS#:+FLAGS#+FLAG#
0360   LOCATION#:+1; LOC_MOD_8#:+1
0370   IF LOC_MOD_8#=8 THEN
0380     LOC_MOD_8#:=0
0390     PRINT FILE 1: BYTES$;CHR$(FLAGS#);
0400     FLAGS#:=0
0410   ENDIF
0420   PRINT ".";
0430 ENDPROC PUT
0440 //
0450 PROC MESSAGE(N#) CLOSED
0460   // Prints an error message
0470   PRINT
0480   PRINT
0490   PRINT "***** ERROR : ";
0500   CASE N# OF
0510   WHEN 1
0520     PRINT "THE DATA AREA HAS BEEN USED"
0530   WHEN 2
0540     PRINT "IMPROPER FILE FORMAT"
0550   ENDCASE
0560 ENDPROC MESSAGE
0570 //
```

```
0580 DIM BUFFER$ OF 40
0590 DIM FILENAME$ OF 20
0600 INPUT "FILE: ": FILENAME$
0610 PRINT
0620 IF NOT "." IN FILENAME$ THEN FILENAME$:+".REL"
0630 OPEN FILE 0, FILENAME$+"/C/B", READ
0640 FILENAME$:=FILENAME$(1:POS(".",FILENAME$)-1)+".EXT"
0650 DELETE FILENAME$
0660 OPEN FILE 1, FILENAME$, WRITE
0670 //
0680 LOCATION#:=0; LOC_MOD_8#:=0
0690 DIM BYTES$ OF 8
0700 FLAGS#:=0
0710 //
0720 PROG_SIZE#:=-1 // NOT YET DEFINED
0730 DIM PROGRAM_NAME$ OF 10
0740 //
0750 LOOP
0760   CASE STREAM#(1) OF
0770   WHEN 0 // ONE ABSOLUTE BYTE
0780     EXEC PUT(STREAM#(8),FALSE)
0790   WHEN 1 // MUST BE FURTHER DISTINGUISHED
0800     CASE STREAM#(2) OF
0810     WHEN 0 // SPECIAL LINK ITEM. FURHTER CHECK
0820       CASE STREAM#(4) OF
0830       WHEN 2 // PROGRAM NAME
0840         PROGRAM_NAME$:=""
0850         FOR I#:=1 TO STREAM#(3) DO // FETCH NAME
0860           PROGRAM_NAME$:+CHR$(STREAM#(8))
0870         NEXT I#
0880         PRINT "PROGRAM NAME - ",PROGRAM_NAME$
0890         PRINT
0900       WHEN 10 // DEFINE SIZE OF DATA AREA
0910         IF STREAM#(2)+STREAM#(8)+STREAM#(8)<>0 THEN
0920           EXEC MESSAGE(1)
0930           GOTO ERROR
0940         ENDIF
0950       WHEN 13 // DEFINE SIZE OF PROGRAM
0960         IF STREAM#(2)<>1 OR PROG_SIZE#<>-1 THEN
0970           EXEC MESSAGE(2)
0980           GOTO ERROR
0990         ENDIF
1000         P1#:=STREAM#(8); P2#:=STREAM#(8)
1010         PROG_SIZE#:=P1#+P2#*256
1020         PRINT FILE 1: CHR$(P1#);CHR$(P2#);
1030       WHEN 11 // SET LOCATION COUNTER
1040         IF STREAM#(2)<>1 THEN
1050           EXEC MESSAGE(2)
1060           GOTO ERROR
1070         ENDIF
1080         FOR I:=LOCATION# TO STREAM#(8)+STREAM#(8)*256-1 DO
1090           EXEC PUT(0,FALSE)
1100         NEXT I
1110       WHEN 14 // END PROGRAM
1120         EXIT
```

```
1130        OTHERWISE
1140          EXEC MESSAGE(2)
1150          GOTO ERROR
1160        ENDCASE
1170      WHEN 1 // PROGRAM RELATIVE DATA
1180        EXEC PUT(STREAM#(8),FALSE)
1190        EXEC PUT(STREAM#(8),TRUE)
1200      OTHERWISE
1210        EXEC MESSAGE(2)
1220        GOTO ERROR
1230      ENDCASE
1240    ENDCASE
1250 ENDLOOP
1260 IF LOCATION#()PROG_SIZE# THEN
1270    EXEC MESSAGE(2)
1280    GOTO ERROR
1290 ENDIF
1300 WHILE LOC_MOD_8#()0 DO // EMPTY BUFFER
1310    EXEC PUT(0,FALSE)
1320 ENDWHILE
1330 CLOSE
1340 PRINT
1350 PRINT
1360 PRINT "Conversion complete. No errors detected."
1370 PRINT "Program size ";PROG_SIZE#;"bytes."
1380 PRINT
1390 END
1400 //
1410 LABEL ERROR
1420 CLOSE
1430 DELETE FILENAME$
1440 PRINT
1450 PRINT
1460 PRINT "Conversion incomplete. Error(s) in the file."
1470 PRINT
1480 END
```