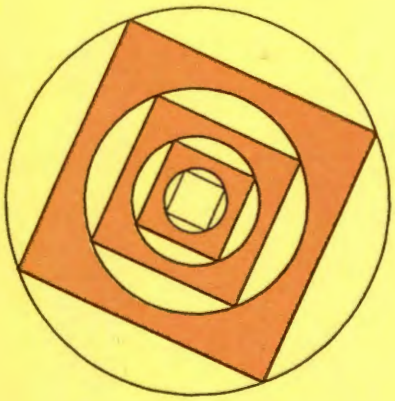
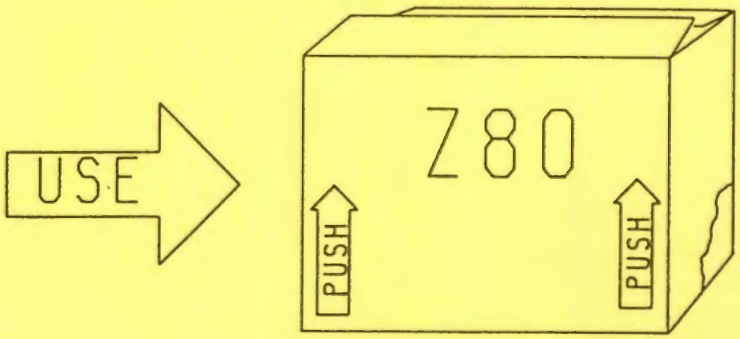
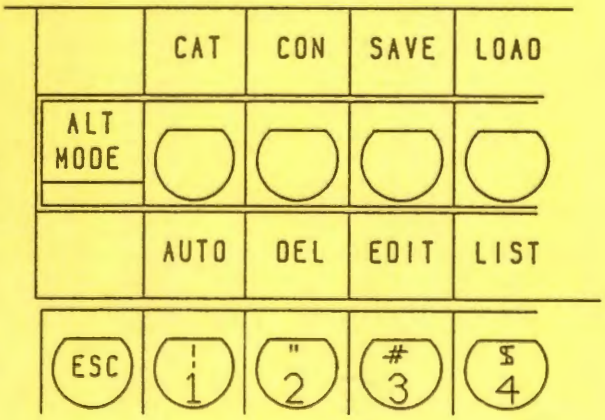
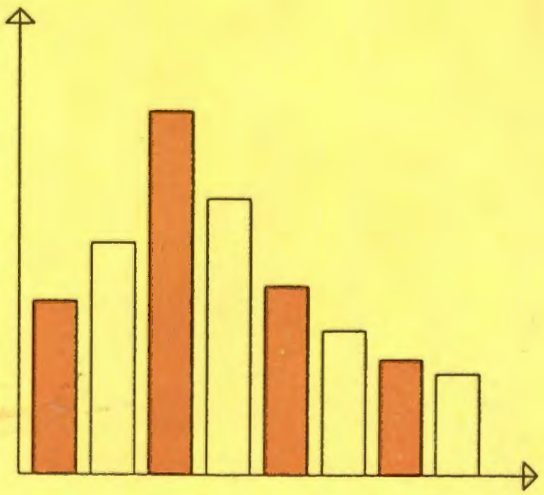
COMAL-80 / Z80

BRUGERVEJLEDNING - MARTS 86









ICL

545001
BRUNDLUNDSKOLEN
LADEGÅRDSVEJ 15
6200 AABENRAA

ICL

Brugervejledning COMAL-80/z80

MAR.86/THB

INDHOLDSFORTEGNELSE

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

Brugervejledning COMAL-80/z80

MAR.86/THB

TILLÆGSMANUAL CP/M-VERSIONEN AF COMAL-80/Z80

PROGRAMOVERSPILNING FRA METANIC-COMAL
KONTROLKODER VED PRINT
CP/M FILNAVNE
ADRESSER I COMAL-80/Z80
SYSTEMVARIABLE I CP/M COMAL-80/Z80

APPENDIX: (Bagest i manualen)

A: UDTRYK I COMAL-80/Z80.

B: FEJLHÅNDTERING I COMAL-80/Z80.

C: MASKINKODE I COMAL-80/Z80.

D: PAKKER I COMAL-80/Z80.

E: OVERSIGT OVER GRAFIKPROCEDURER.

F: OVERSIGT OVER FUNKTIONSTASTERS ANVENDELSE.

G: MINITEXT - ET MINITEKSTBEHANDLINGSSYSTEM OG ØVRIGE DEMO-PROGRAMMER PÅ COMAL-DISKETTTEN.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

BRUGER- OG OPSLAGS-MANUAL TIL COMAL-80/Z80.

INDLEDNING.

Datamater anvendes i stadig flere forskellige sammenhænge som et moderne hjælpeværktøj. Det er derfor af stor betydning, at mennesker har let adgang til at lære om datamater og om muligheder og begrænsninger for deres anvendelse inden for den enkeltes interesseområde.

Efterhånden som udviklingen på edb-sprogs-området går i retning af systemer, som er tilpasset til løsning af specielle opgaver, bliver det i almen og grundlæggende undervisningssammenhæng stadig mere værdifuldt at have et edb-sprog, der kan bruges både som indgang til og ved videre arbejde med så at sige alt.

Den her foreliggende COMAL-80/Z80 er i særlig grad brugervenlig og alsidig. Dertil kommer, at den i forhold til sine mange faciliteter levner god arbejdsplads i arbejdslageret, og at den skønt værende en fortolker arbejder særdeles hurtigt.

Nogle fordele og muligheder ved at anvende COMAL-80/Z80 i edb-undervisningen skal her kort antydes i ikke prioriteret rækkefølge:

Man kan uden det store besvær sætte elever uden forkundskaber i gang med lette opgaver, og man har mulighed for at lade dem arbejde sig frem til at kunne udvikle programmer til løsning af mere komplicerede opgaver.

Man kan ved at eksperimentere få kendskab til datamatens egenskaber.

Ved at studere færdige comalprogrammer - og dem har vi jo efterhånden mange af - eller dele af sådanne kan kursisterne hurtigere lære noget væsentligt om edb, herunder nemmere komme ind i tidssvarende og logiske tankegange og arbejdsformer.

Den kursist, som er meget matematisk og/eller logisk indstillet, har mulighed for at bruge sine interesser til at fremstille overskuelige og komprimerede systemer.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

Den, som prøver at sammenflikke noget helt på egne præmisser, har også muligheder - og kan oven i købet være heldig at få et "spagettiprogram" til at virke.

Den kreative såvel som den originale elev har begge muligheder for at lære og udvikle sig ved at anvende COMAL-80/z80.

COMAL-80/z80 gør det altså lettere for den enkelte på egne præmisser at komme igang med at styre og bruge datamaten.

Vil man lære om eller begynde at arbejde med processtyring kan det gøres i COMAL.

Ønsker man at bruge datamaten som et grafisk tegneværktøj, er COMAL hjælpeværktøjet, som gør, at man hurtigt kan komme i gang.

Erfaringen viser, at arbejde i COMAL gør det lettere at sætte sig ind i andre og mere specialiserede edb-sprog og edb-systemer. Men der er også set eksempler på, at edb-specialister har haft stort udbytte af at gå i gang med at udvikle noget i COMAL.

Man kan kort sagt bruge COMAL som et at værktøjerne i den elementære undervisning, og man kan bruge sproget i mere avanceret undervisning - også til at lave programmer af professionel karakter.

Manualen er opdelt i:

1. En emneorienterende del, hvor nogle hovedemner vedrørende programmering i COMAL-80/z80 tages op. Ved hvert emne findes oplysninger om de vigtigste kommandoer og sætninger set i relation til emnet, og der er eksempler på småprogrammer, som belyser emnet. Disse programmer medfølger på COMAL-80/z80-disketten, og de skulle gerne gøre det lettere at komme i gang med at studere emnerne.

2. En alfabetisk oversigt over reserverede COMAL-80/z80-ord, samt en kort og præcis beskrivelse af det enkelte ords syntaks og anvendelsesmuligheder som direkte kommando og/eller sætningskommando. Denne del er udarbejdet på grundlag af beskrivelsen udarbejdet af COMAL-80/z80's konstruktør: Freddy Dan Dalgas Kristiansen.

OPSTART AF COMAL-80/Z80.

COMAL-80/Z80 er lagret på den fremsendte COMAL-80/z80-diskette som en .COM fil under navnet: COMAL.COM.

I COMAL-80/Z80 indgår en maskinsprogspakke med GRAFIK-procedurer og funktioner. Den er lagret på disketten under navnet GRAPHICS.PCK.

Endvidere ligger på denne diskette følgende programmer:

AUTOLOAD.COM. Dette program kan bruges til at fremstille COMAL-disketter med automatisk opstart.

Grafikdemonstrationsprogrammet DEMO.SAV.

Program til udskrift af højopløsningsgrafikbilleder på printer: GRAFDUMP.COM.

Nogle demonstrationsprogrammer, som er omtalt i manualen.

Tag først en sikkerhedskopi af den tilsendte COMAL-diskette og arkiver originaldisketten.

Ved opstart UDEN autoload skal man først indlæse operativsystemet, og når dette melder klar ved at skrive 'A>' på skærmen, indtaster man: 'COMAL' og trykker på RETURN-tasten.

Herefter indlæses COMAL fra disketten med efterfølgende opstart af systemet til følge.

Ønsker man AUTOMATISK opstart, skal man først kopiere COMAL over på en ny diskette og derefter ved anvendelse af programmet AUTOLOAD lægge operativsystem med autoopstart over på denne diskette.

COMAL melder sig med en lille udskrift om version og COPYRIGHT øverst på skærmen, og COMAL er klar til brug, når den blinkende cursor viser sig på skærmen, hvilket er Comal's tilkendegivelse over for brugeren om, at systemet er klar til at modtage kommando. Hvis cursoren er slukket, skal man altså vente på klarmelding.

AT AFGIVE DIREKTE KOMMANDO(ER).

Man afgiver en direkte COMAL-kommando ved at skrive den på skærmen og derefter trykke på RETURN-tasten. Bemærk, at der ikke må stå andet end kommandoen i skrivelinjen, og at man hvis der er flere linjer med kommandoer på skærmen, kan gentage en kommando ved at flytte cursoren op til en linje med en kommando og trykke RETURN.

En alfabetisk oversigt over samtlige mulige COMAL-kommandoer findes bag i manualen.

Eksempler på direkte kommandoer:

AUTO (Giver automatisk linjenummerering).

CAT"DK1: (Giver catalogudskrift fra diskette i drev B: - eller diskettestation nr. 1.).

DIR (Giver en udskrift af filnavne på disketten).

PRINT "Prøvetekst" (Udskriver teksten: 'Prøvetekst' på skærmen).

PAGE (Renser skærmen og stiller cursoren i øverste venstre hjørne - eller skifter side på printeren, hvis den er udskriftsmedie). Denne kommando har i øvrigt samme virkning som kommandoen CLEAR.

RUN (Starter programafvikling, hvis der er et program i lageret.

LINK "GRAPHICS (Tilkobler grafikprocedure til program).

Bemærk:

Med funktionstasterne eller ved anvendelse af CONTROL-tasten + en anden tast kan man få skrevet en del COMALkommandoer. Se oversigt over FUNKTIONSTASTERS ANVENDELSE VED EDITERING bagest i manualen. Overlæg til funktionstaster leveres med COMAL-80/Z80.

AT SKRIVE PÅ SKÆRMEN

Da der er indbygget skærmeditering i COMAL-80/Z80, kan man frit skrive og rette i hele skærmbilledet.

Med de 4 piletaster (op-, ned-, højre- og venstre-) kan man flytte cursoren derhen på skærmen, hvor man ønsker at skrive, rette eller slette. Der er i alt plads til 25 linjer med hver 80 tegn på skærmen. Den linje, hvori cursoren befinder sig, kaldes den aktive linje.

Hvis man trykker på RETURN-tasten, opfatter systemet teksten i den aktive linje som en direkte COMAL-kommando.

Hvis den aktive linje begynder med et tal i intervallet 1 - 9999, opfatter systemet linjen som en direkte kommando til at indlæse linjen som en COMAL-sætning i programlageret.

I alle tilfælde vil systemet, når RETURN-tasten aktiveres, undersøge den aktive linje for syntaksfejl, og hvis der findes fejl, vil linjen ikke blive indlæst i computeren, og kommandoen derfor heller ikke blive udført. I stedet udskrives en fejlmelding, og man kan så med piletasterne gå op i linjen, rette og påny trykke RETURN.

Når man skriver på skærmen, overskrives evt. tidligere skrevet skærmtekst.

Den gamle tekst slettes altså og erstattes med den nye.

Bemærk:

1. Brugen af specialtasterne: DEL (slet) og INS (indsæt).

   Som det ses, kan man med DEL-tasten "æde" den efterfølgende tekst og med INS-tasten "skubbe" den efterfølgende tekst frem, så der skaffes plads til ny tekst.

2. Med funktionstasterne eller ved anvendelse af CONTROL-tasten + en anden tast kan man få skrevet en del COMALkommandoer. Se oversigt over FUNKTIONSTASTERS ANVENDELSE VED EDITERING bagest i manualen.

3. De specielle funktionstaster:

a) DOBB.LYS (^Ø). Når den er aktiveret, fremtræder skriften på skærmen med dobbelt lysstyrke.

b) INVERS SKRIFT (^Å). Når den er aktiveret, skrives der i invers skrift.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

c) UNDERSTREG (^^). Når den er aktiveret, understreges det skrevne.

Gentagelse af aktivering af ovennævnte specielle funktionstaster ophæver funktionen. Hveranden gang tasten bruges, aktiveres funktionen og de øvrige gange ophæves den.

d) CURSOR HOME (^P). Bevirker, at cursoren flyttes til øverste venstre hjørne af skærmen - uden at skærmen slettes.

e) DEL LINIE (^Q). Sletter i den linje, som cursoren står i. Der slettes fra cursoren og resten af linjen.

AT SKRIVE ET PROGRAM.

Et COMALprogram er opbygget af nummererede linjer. Linjenumrene skal ligge i intervallet 1 - 9999.

Hvis man lave et COMALprogram, skal man altså skrive de linjer på skærmen, som programmet skal bestå af. Når en linje er færdigskrevet eller rettet, afslutter man med at aktivere RETURN-tasten, hvorved linjen indlæses i programlageret i datamatens hukommelse.

For at undgå indlæsning af noget, som COMAL-fortolkeren ikke kan arbejde med, undersøges inden indlæsning, om der er syntaksfejl i den aktive linje.

Hvis det er tilfældet, indlæses linjen ikke i programlageret; men der udskrives i stedet en fejlmelding. Man kan så flytte cursoren op i linjen, rette den, og igen trykke RETURN.

Prøv at indskrive programmet, som står lidt fremme i teksten her.

Prøv også undervejs at lave fejl i nogle af de linjer, som du vil have indlæst, så du får afprøvet, hvordan fejlmeldingerne virker.

Afgiv evt. den direkte kommando PAGE først.

Og hvis du bruger kommandoen AUTO, kan du få systemet til selv at lave linjenumre!

Men så skal du vide, at man slipper ud af AUTO-tilstanden ved at aktivere ESC-tasten.

ESC-tasten bør du lægge mærke til fra starten. Når man arbejder i COMAL kan man normalt altid få maskinen til at stoppe og derved få mulighed for at afgive en ny kommando ved at aktivere ESC.

ESC betyder ecsape eller flygt eller stands.

Kun når cursoren blinker på skærmen, er det muligt at afgive en kommando.

Hvis et COMAL-program styrer datamaten, vil cursoren være slukket undtagen, når der skal indtastes noget. Og man kan altså kun standse programudførelse ved at aktivere ESC-tasten.

```
0010 // PROGRAM1
0020 PAGE
0030 PRINT "   ";CHR$(26);"Dette er mit første program,";
          CHR$(26) // BLINK
0040 PRINT
0050 PRINT CHR$(29);"- og det bliver nok ikke mit sidste.";
          CHR$(29) // INVERS
0060 PRINT
0070 PRINT
0080 PRINT CHR$(30);"Man kan aldrig vide, hvad datamater
          tænker.";CHR$(30)// UNDERSTREG
0090 PRINT
0100 PRINT
0110 PRINT "                    Venlig hilsen"
0120 PRINT
0130 FOR x:=1 TO 50 DO PRINT CHR$(7);// LYD
0160 PRINT
0170 PRINT "            ";CHR$(28);"Peter Plys.";CHR$(28)
          // DOBBELT LYS
```

Når programmet er færdigskrevet, kan du prøve at LIST'e og RUN'e programmet. Det gøres ved henholdsvis at afgive de direkte kommandoer:

```
LIST
```

og

```
RUN
```

Hvis du har en printer tilsluttet, kan du også afprøve kommandoen:

```
LIST "LP:
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

Bemærk, at programmet indeholder nogle specielle skrivekoder til systemet. De er angivet med koden: CHR$(X). Bag //'stregerne er i programmet angivet, hvad koderne bevirker. - Det skal også bemærkes, at disse koder kun virker på 4 MHz maskiner.

AT LAGRE ET PROGRAM PÅ DISKETTE.

Når foranstående program er indskrevet og indlæst i datamatens programlager, er det klar til at blive brugt. Men hvis der bliver slukket for datamaten, forsvinder det igen. Datamaten har et baggrundslager, oftest en eller flere disketter, hvorpå man kan gemme programmer m.m., så man kun behøver at skrive dem en gang.

Hvis du afgiver kommandoen:

```
SAVE "PROGRAM1
```

Bliver programmet udskrevet på disketten under programnavnet: PROGRAM1.SAV

Prøv det!

En mængde data udskrevet på datamatens baggrundslager kaldes også EN FIL eller EN DATAFIL. Og man taler i den forbindelse også om FILNAVN og FILTYPE. De sidste 3 tegn i programnavnet er filtypen, og tegnene før . er filnavnet.

Et programnavn må højst bestå af 8 tegn (bogstav- eller taltegn. Det første tegn skal dog være et bogstav).

Når man lagrer et program ved anvendelse af SAVE-kommandoen, skriver systemet selv .SAV bag navnet. I dette tilfælde har vi altså oprettet en fil af typen SAV, hvilket er en forkortelse for en SAVE-fil. Programmet er noteret i binær-kode på disketten.

Man kan også lagre et program ved anvendelse af LIST-kommandoen, f. eks. således:

```
LIST "PROGRAM1
```

I dette tilfælde noteres programmet i ASCII'kode i en såkaldt

PROGRAM1.LST

Med DISPLAY-kommandoen kan man også LIST'e. Men her bliver linjenumrene ikke udskrevet. Indrykningsstrukturer bevares. Programmet fremtræder derved i en tekst - lignende godt opstillede Pascal-programmer.

Udskrevet på denne måde vil et program eller del heraf være velegnet til at indlæse i en undervisningstekst i et tekstbehandlingssystem.

AT HENTE ET PROGRAM FRA DISKETTE.

Programmer, som er SAVE'et, kan hentes ind i datamatens programlager ved anvendelse af kommandoen LOAD.

Eks: Kommandoen

```
LOAD"PROGRAM1
```

Vil, om muligt, hente en kopi af programmet: PROGRAM1.SAV på det aktuelle diskettedrev.

Hvis der i forvejen er et program i programlageret, vil det blive slettet, inden det nye indlæses. Der kan altså ikke ske sammenblanding.

Programmer, som er LIST'et eller DISPLAY'et ud på disketten, kan læses ind igen ved brug af ENTER.

Eks: Kommandoen:

```
ENTER "PROGRAM1
```

vil, hvis muligt, læse en kopi af programmet: PROGRAM1.LST fra den aktuelle diskette og ind i arbejdslageret.

Er der allerede et program i lageret, vil det blive slettet. Heller ikke her kan der ske sammenblanding.

Med kommandoen MERGE kan man koble et program til det, som er i programlageret.

Det nye program bliver koblet til slutningen af det gamle.

AT RETTE I ET PROGRAM.

Da der er skærmeditering i COMAL-80/Z80 er det let at rette i et program.

Med OP- og NED-piletasterne kan man bevæge sig hhv. op og ned i programmet til man er nået frem til den linje, som ønskes ændret.

Med TILHØJRE- og TILVENSTRE-piletasterne kan man herefter bevæge sig hhv. frem og tilbage i linjen til det sted, hvor ændringen skal foretages.

Hvis man nu skriver, vil der ske en overskrivning af den tekst, som evt. står på linjen i forvejen.

Når rettelserne i en linje er fuldført, trykkes RETURN, hvorved linjen kontrolleres. Hvis linjen godkendes som en direkte Comal-kommando, udføres kommandoen, og hvis linjen godkendes som en Comal-programsætning, indlæses denne i programlageret. I modsat fald udskrives fejlmelding, hvorefter man kan flytte cursoren op i linjen og rette igen - eller man kan opgive og lave noget andet.

BEM.: Hvis man ikke aktiverer RETURN-tasten er rettelsen ikke foretaget. Det betyder, at kommandoen i så tilfælde ikke bliver udført eller at programlinjen ikke bliver indlæst!

Der findes i COMAL-80/Z80 en del faciliteter, som kan bruges i forbindelse med programrettelser:

1. CURSOR-HOME funktionstasten (^P). Bruges til at flytte cursoren op i øverste venstre hjørne af skærmen uden at slette skærmteksten.

2. DEL-LINIE funktionstasten (^Q). Bruges til at slette resten af linjen - fra cursorposition og helt ud til højre. Er f. eks. også praktisk at bruge, hvis man midt i en programrettelse ønsker at afgive en direkte kommando. Man kan så blot skrive kommandoen t.v. oveni en programlinje, slette resten af linjen og trykke RETURN).

3. EDIT-kommandoen (^B) kan bruges til hurtigt at få lige netop den/de linjer, som ønskes rettet, frem på skærmen.

4. FIND"-kommandoen (^D) kan bruges til hurtigt at finde frem til et bestemt ord eller en bestemt tekst i programmet. Og man har mulighed for kun at søge i en med linjenumre afgrænset del af teksten. Søgningen fortsættes ved tryk på RETURN, og den afbrydes med ESC.

5. CHANGE-kommandoen bruges til at udskifte en tekststreng med en anden.

   F. eks. vil kommandoen:

   CHANGE "LP:","DS:"

   bevirke, at systemet fra cursorens position og fremefter vil søge efter teksten LP:. Hvis den findes, standser søgningen, og man får mulighed for at vælge, om den skal udskiftes med DS:, eller om man vil fortsætte, eller standse søgningen. Man kan også vælge at rette et eller andet i nærheden af LP: og derefter fortsætte søgningen. Som ved FIND"-kommandoen kan der ved angivelse af linjenumre søges i afgrænsede dele af et program.

6. Hvis man ønsker at ændre en bestemt procedure i et stort program, kan man starte med den direkte kommando:

   LIST "navn

   hvor "navn" er procedurens navn. Kommandoen bevirker, at proceduren udskrives på skærmen. Man kan så studere den og finde ud af, hvor man evt. vil rette.

AT PRØVEKØRE OG SPORE ET PROGRAM.

Det er altid nødvendigt at foretage prøve- og test-kørsler på nye programmer. I COMAL-80/Z80 er indbygget særlige faciliteter, som kan bruges i denne forbindelse.

Hvis der ved en prøvekørsel af et program findes fejl, standser systemet og udskriver fejlmelding, som består af et linjenummer og en fejlforklaring.

Man kan så v. hj. a. EDIT kommandoen forsøge at rette fejlsætningen samt evt. rette andre linjer, som måtte have samme skavank, og derefter prøvekøre igen.

Fejl fremkommer hyppigt som forkerte variabelværdier. Det kan være tekster, som søges læst ind i talvariable, eller heltalsvariable, som søges tildelt for store værdier eller decimaltalsværdier. o.s.v.. I sådanne tilfælde er det ofte en god ide - enten ved direkte kommandoer, eller ved at indsætte ekstra printsætninger i nærheden af fejllinjen - at få udskrevet nogle af de variabelværdier, som man har mistanke til.

Kommandoen SCAN, kan bruges inden egentlig prøvekørsel. Den undersøger programmet i programlageret for evt. strukturfejl og udskriver dem. Bem: SCAN run'er ikke programmet.

Men når et program er SCAN'et, kan enhver procedure og funktion i det kaldes ved en direkte kommando og altså på den måde prøvekøres.

F. Eks. vil efter en SCAN'ning af et program, som indeholder proceduren skærmbillede, kommandoen:

EXEC SKÆRMBILLEDE

bevirke at proceduren prøvekøres. Hvis der i proceduren optræder variable, som er erklæret andetsteds i programmet, bevirker det fejlmelding.

Kommandoen TRACE er velegnet både ved prøvekørsler og ved demonstrationskørsler i undervisningssammenhæng.

TRACE bruges til at spore et program. Hvis kommandoen er aktiv udskrives programlinjerne i den rækkefølge, som de bliver udført, ligesom evt. ind- og ud-læsninger af data udskrives.

Man kan vælge mellem at få een Comal-sætning udført ad gangen - så venter systemet på RETURN efter hver linje - eller at få hele programmet sporet uden stop. Og i sidste tilfælde kan man vælge mellem skærm- og printer-udskrift.

PROGRAMAFVIKLING.

Et Comal-program sættes i gang og afvikles ved anvendelse af den direkte kommando RUN.

Programmet skal være i programlageret. Det kan enten være skrevet direkte eller hentet ind fra disketten med kommandoerne LOAD" eller ENTER". (H.h.v. SAV' og LST'filer).

Når man henter et program ind fra disketten ved anvendelse af LOAD" eller ENTER" kommandoerne, tømmes programlageret før indlæsning. Man behøver altså ikke at frygte sammenblanding med evt. tidligere program.

Inden et program RUN'es bør man overveje, om printer og evt. andre ydre enheder, som programmet skal anvende, er tilsluttet og tændt. Påse også, at der er disketter i de diskettedrev, som programmet bruger.

AT SAMMENKOBLE PROGRAMMER.

Med kommandoen MERGE" kan man hente et program af LST'typen på disketten og få det skrevet i forlængelse af et allerede i programlageret værende program.

Man har også i forbindelse med MERGE-kommandoen mulighed for at angive hvilke linjenumre, det indlæste program skal have. Derved kan man indflette programmet på et ønsket sted i det gamle program.

Det kan anbefales, at man for at undgå dobbeltarbejde, gemmer f. eks. de procedurer og programmer, som man skønner at kunne bruge i andre forbindelser, som LST'filer på disketten. Så kan man altid ved anvendelse af MERGE let hente dem ind til genbrug i andre programmer. Oparbejdning af en diskette med et procedurebibliotek er en hensigtsmæssig arbejdsform, hvis man vil fremstille egne programmer.

AT ANVENDE HØJOPLØSNINGSGRAFIK.

Der er i COMAL-80/Z80 indbygget en højopløsningsgrafikdel. Men forudsætningen for, at den kan udnyttes, er, at der er indbygget et MPS-24 grafikmodul I COMETEN.

MPS-24 modulet indeholder et RAM-lager på 2 x 32 KB, som kan indeholde to grafikbilleder på hver 512 x 512 punkter. Den her foreliggende grafik har altså en opløselighed på skærmen på 512 såvel vandret som lodret, og den giver muligheder for at arbejde på/med to forskelllige grafikbilleder (+ det almindelige videobillede) i samme program.

Til at styre arbejdet med de to grafikbilleder og for at sætte tempoet i vejret er der i MPS-24 modulet indbygget en Thomson grafikprocessor.

Med COMAL-80/z80 medfølger en såkaldt GRAFIKPAKKE af procedurer og funktioner skrevet i maskinsprog. Disse rutiner ligger på disketten under navnet GRAPHICS.PCK.

Pakken gøres tilgængelig ved, at man afgiver den direkte kommando:

```
LINK "GRAPHICS
```

Systemet svarer, hvis grafikpakken er på disketten:

```
USE "GRAPHICS
```

Og sidstnævnte kommando skal altid anbringes som sætningskommando først i et program, som anvender grafikmodulet. Således:

```
0010  USE "GRAPHICS
```

Når grafikpakken anvendes er arbejdeslageret ca. 4 KB mindre (ca. 19 KB), end når der ikke bruges grafik.

Der er indbygget i alt 26 procedurer/funktioner i en grafikpakke.

I bilag bag i manualen kan man se, med hvilke navne og parametre grafikprocedurerne/funktionerne skal kaldes.

Man kan v. hj. a. grafikprocedurerne f. eks. let få tegnet linje-stykker, rektangler, punkter, cirkler, lagkagestykker og man kan udfylde og slette områder.

I grafikpakken er alle tastaturtegn til rådighed, og de kan forstørres i begge retninger og teksten kan skrives både lodret og vandret på skærmen.

På Comal-disketten medfølger i øvrigt et grafikdemonstrationsprogram. (DEMO.SAV). Nærmere studier heraf vil være en let indgang til at lære at bruge den.

Med programmet GRAFDUMP.COM kan grafikbillederne dumpes ud på Mikroline printere fra U182 og opefter. Mikroline U82A skal have udskiftet 3 EPROM's, hvis den skal kunne bruges hertil.

Det her efterfølgende eksempel på et grafikprogram, anvender begge grafikskærme i grafikmodulet, og det kan give nogle indgangsvink til den, som ønsker at gå i gang med at bruge modulet.

```
0010 // hartjul
0020 DIM b$(12) OF 40
0030 b$(2):="Med denne lille hilsen fra"
0040 b$(3):="SV Tønders INFORMATIKHOLD,"
0050 b$(4):="lærer og undertegnede en"
0060 b$(5):="stor tak for et særdeles"
0070 b$(6):="udbytterigt virksomheds-"
0080 b$(7):="besøg d. 10/12/85."
0090 b$(12):="ERLING MORTENSEN"
0100 USE "GRAPHICS"
0110 //
0120 FOR y:=1 TO 2 DO
0130   // TEGNING AF JULETRÆ.
0140   // gs = GRAFIKSKÆRM
0150   gs(y)
0160   box(20,450,470,60,0)
0170   fill(21,451)
0180   textstyle(3,3,0)
0190   // pc = PENCOLOR
0200   pc(0)
0210   plottext("Glædelig jul. 1985.",30,490)
0220   pc(1)
0230   FOR i:=360 DOWNTO 100 STEP 15 DO
0240     circle(255,i+30,(360-i)/3.5)
0250     IF i MOD 30 THEN fill(255,i+32)
0260   ENDFOR i
0270   FOR i:=70 TO 100 STEP 8 DO
0280     ellipse(252,i,160-i,8)
0290     arc(255,i+10,i-25,0,180)
```

```
0300  ENDFOR i
0310  FOR i:=16 TO 240 STEP 4 DO
0320   circle(255,i,100/i+4)
0330  ENDFOR i
0340  // TEGNING AF JULESTJERNE.
0350  moveto(230,400)
0360  drawto(255,443)
0370  drawto(280,400)
0380  drawto(230,400)
0390  moveto(230,422)
0400  drawto(280,422)
0410  drawto(255,379)
0420  drawto(230,422)
0430  box(20,1,470,55,0)
0440  //
0450  fill(21,2)
0460  pc(0)
0470  plottext("Og godt nytår 1986.",30,45)
0480  //ts = TEKSTSKÆRM
0490  ts(0)
0500  INPUT "Indtast VIRKSOMHED: ": b$(1)
0510  INPUT "Indtast ATT.NAVN: ": b#(10)
0520  gs(y)
0530  pc(1)
0540  textstyle(1,2,0)
0550  plottext("Til "+b$(1),20,400)
0560  plottext(b$(10),320,400)
0570  textstyle(1,2,0)
0580  plottext(b$(2),20,350)
0590  plottext(b$(3),295,350)
0600  plottext(b4(4),20,320)
0610  plottext(b$(5),310,320)
0620  plottext(b$(6),20,290)
0630  plottext(b$(7),320,290)
0640  textstyle(1,2,0)
0650  plottext("Venlig hilsen ",360,160)
0660  plottext("SV TØNDER ",20,160)
0670  plottext("19/12/85",20.130)
0680  plottext(b$(12),360,130)
0690 ENDFOR y
```

Ved hjælp af GRAFDUMP.COM giver programmet følgende udskrift af een af grafikskærmene:

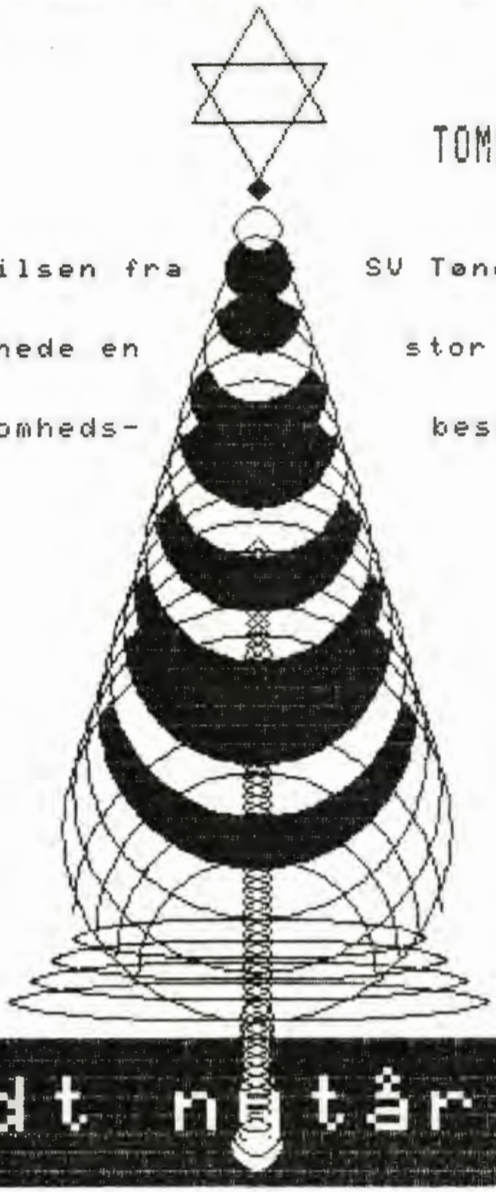
PAPIRUDSKRIFT AF GRAFIKSKÆRM 1.

Glædelig jul. 1985.

Til ICL A/S

TOMMY HELDING BENTZEN

Med denne lille hilsen fra SV Tønders INFORMATIKHOLD,
lærer og undertegnede en stor tak for et særdeles
udbytterigt virksomheds- besøg d. 10/12/85.



SV TØNDER
19/12/85

Venlig hilsen
ERLING MORTENSEN

Og godt nytår 1986.

International Computers Limited a/s
Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

INDDATA.

Skal en datamat bruges til noget fornuftigt, må man først have data - og helst mange - ind i den.

Det er naturligvis ikke ligegyldigt, hvilke data, der indkodes. Normalt planlægges først, hvad man ønsker, at et edb-program skal producere af UDDATA.

Herudfra kan man så finde ud af hvilke INDDATA, der er nødvendige, samt hvilke data systemet selv skal frembringe ved DATABEHANDLING.

Her behandles problemerne af overskuelighedshensyn i rækkefølgen:

INDDATA - DATABEHANDLING - UDDATA

De væsentligste muligheder for i COMAL-80/Z80 at få data ind i datamaten er følgende:

1. Med kommandoerne LOAD" og ENTER" kan man hente programfiler fra disketten og ind i programlageret.

2. I bl. a. PRINT- og DATA-sætminger kan man få såkaldte "dødtekster" ind i datamaten. Det vil sige tekster, som man ikke kan udøve direkte databehandling på.

   eks.: 0220 PRINT "Afsenderen er ikke hjemme."

   0460 DATA "LANGE PETER MADSEN","SØLYST"

   0050 INPUT "Indtast navn ": NAVN$

3. Ved anvendelse af TILDELINGSSÆTNINGER, kan man også få data ind i datamaten.

   Eks. på simple tildelingssætninger:

   0050 C:=13.4/1.5

   0060 A:=B+2*C

```
0070 NAVN$:="PETER PLYS"

0080 ADR$:=ADR$1+PDIST$
```

Tildelinger forekommer f. eks. også i: INPUT-, READ-, GET$-sætninger.

```
Eks. 0100 INPUT A,B,C#

     0110 INPUT "Indtast 3 tal, det sidste helt! ": A,B,C#

     0120 READ NAVN$,ADR$,PDIST$,NR

     0130 TLFNR$:=GET$(2,16)// Henter fra diskfil nr. 2.
                              16 bytes ind i TLFNR$.
```

Nedenstående programeksempel kan belyse INDDATA-problematikken.

```
0010 // INDDATA
0020 DIM navn$(3) OF 20,adr$(3) OF 20,pdist$(3) OF 20,
     tlfnr$ OF 16
0030 PRINT "Afsenderen er ikke hjemme."
0040 DATA "LANGE PETER MADSEN","SØLYST","6270 TØNDER",200
0050 INPUT "Indtast navn ": navn$(1)
0060 INPUT "Indtast adresse ": adr1$
0070 INPUT "Indtast postdistrikt ": pdist$(1)
0080 a:=13.4/1.5; b:=14
0090 c:=a+2*b
0100 navn$(2):="PETER PLYS"
0110 adr$(2):=adr$(1)+pdist$(1)
0120 INPUT d,e,f#
0130 INPUT "Indtast 3 tal, det sidste HELT! ": h,i,j#
0140 OPEN FILE 2,"TELEFON",READ
0150 READ navn$(3),adr$(3),pdist$(3),nr
0160 tlfnr$:=GET$(2,16)
```

Man kan også få data ind med direkte kommandoer, f. eks. tildelingskommandoer. Men det er normalt ikke særligt hensigtsmæssigt.

INPUTKONTROLRUTINER

HVORFOR? En vigtig forudsætning for at datamaten kan løse de opgaver, som den programmeres til, er at den kun fodres med relevante data. Mange af de fejl, som edb behandling er årsag til skyldes forkerte inddata. Hvis selve datamaten og programmerne til den interne databehandling og til udskrifter er i orden, arbejder datamaten helt korrekt, hvis den blot ikke får forkerte

input.

Ikke alene kan forkerte inddata medføre fejl i uddata. Systemet kan gå ned. Skærmbilledet kan blive ødelagt, og brugerens nervesystem kan måske på længere sigt tage skade.

Det kan derfor - især hvis andre end programmøren skal anvende programmet - betale sig nøje at overveje og planlægge, hvordan man "idiotsikrer" forbindelsen fra brugeren til maskinen, samt fra evt. andre inddataveje (datafiler, m.v.).

HVORDAN? Større systemer eller delsystemer styres som regel ud fra menuvalgrutiner, og allerede ved konstruktionen af disse er det vigtigt, at der kun kan vælges blandt de foreliggende muligheder. Al anden indkodning bør systemet ignorere.

Studer nedstående eksempel på en DEMO-menu.

```
0010 // VALGMENU
0020 REPEAT
0030   PAGE
0040   PRINT AT 2,1: " H O V E D M E N U - D E M O - E M O ->"
0050   PRINT AT 5,2: "  ÆNDRINGER i kartoteket --------->"
0060   PRINT AT 8,2: "  SKÆRMUDSKRIFTER ---------------->"
0070   PRINT AT 11,2: "  PAPIRUDSKRIFTER ---------------->"
0080   PRINT AT 14,2: "  SØGNINGER i kartoteket --------->"
0090   PRINT AT 17,2: "  STATISTISKE undersøgelser ------>"
0100   PRINT AT 20,2: "  <---------  P A U S E  --------->"
0110   PRINT AT 23,2: "  <- SYSTEM-BENYTTELSE AFSLUTTET ->"
0120   PRINT AT 2,41: " Brug piletaster og tryk RETURN "
0130   valg(7)
0140   IF job#<>6 THEN
0150     PRINT AT job#*3+2,48: " * ET ØJEBLIK * ";
0160   ENDIF
0170   CASE job# OF
0180   WHEN 1
0190     prøve(job#)
0200   WHEN 2
0210     prøve(job#)
0220   WHEN 3
0230     prøve(job#)
0240   WHEN 4
0250     prøve(job#)
0260   WHEN 5
0270     prøve(job#)
0280   WHEN 6
0290     prøve(job#)
0300   WHEN 7
0310     prøve(job#)
0320   OTHERWISE
0330   ENDCASE
0340 UNTIL job#=7
```

```
0350 PAGE
0360 CURSOR 23,1
0370 END
0380 //
0390 PROC valg(maxantal#)
0400   job#:=1
0410   PRINT AT job#*3+2,47: "<------";
0420   LOOP
0430     POKE 256,0
0440       REPEAT
0450       UNTIL PEEK(256)<>0
0460     PRINT AT job#*3+2,47: "       ";
0470     CASE PEEK(256) OF
0480     WHEN 10
0490       job#:+1
0500       IF job#>maxantal# THEN job#:=1
0510       PRINT AT job#*3+2,47: "<------";
0520     WHEN 11
0530       job#:-1
0540       IF job#<1 THEN job#:=maxantal#
0550       PRINT AT job#*3+2,47: "<------";
0560     WHEN 13
0570       EXIT
0580     OTHERWISE
0590       PRINT AT job#*3+2,47: "<------";
0600     ENDCASE
0610   ENDLOOP
0620 ENDPROC valg
0630 //
0640 PROC prøve(nr#)
0650   PAGE
0660   t#:=0
0670   FOR x#:=1 TO 400 DO
0680     CURSOR 12,25
0690     IF x# MOD 20 THEN t#:+1
0700     IF x# MOD 20=0 AND t# MOD 2 THEN
0710       PRINT SPC$(30);
0720     ENDIF
0730     IF x# MOD 20=0 AND NOT t# MOD 2 THEN
0740       PRINT USING "VALG NR # ER IKKE PROGRAMMERET": nr#,
0750     ENDIF
0760   ENDFOR x#
0770 ENDPROC prøve
```

Afprøv programmet, det ligger på disketten.

Hvilke inputdata accepteres?

Gør nøje rede for, hvor og hvordan programmet sikrer mod forkerte input.

Prøv at ændre procedureparameteren i linje 130.

Find ud af, afprøv og forklar, hvordan man kan ændre programmet, så der bliver flere eller færre menupunkter.

HVILKE sætningskommandoer er velegnede til tastaturinputkontrol?

I programmet VALGMENU udføres den væsentligste kontrol ved at POKE'e og PEEK'e til/fra datamatens tastaturinputlagercelle. (Celle nr. 256). Langt den overvejende del af programmet er skærmstyring, som jo også er nødvendigt, hvis tingene skal fungere overskueligt.

Udover POKE/PEEK sætninger findes en del andre, som kan bruges i inputkontrolsammenhæng.

Nedstående eksempel CPRKONTR tager udgangspunkt i cpr.nr. kontrol.

Samtidigt lægger programmet op til kontrol ved indkodning af navne. Hvis man i et navnekartotek ønsker at kunne: Søge, sortere, vende om på fornavn/efternavn, så er det dømt til at gå i fisk, hvis man ikke har sikret sig, at de indkodede navne overholder bestemte standarder og ikke indeholder fjollede tegn.

Prøvekør og studer programmet:

```
0010 // CPRKONTR
0020 DIM buf$ OF 50,tegn$ OF 80,t#(11),navn$ OF 30
0030 tegn$:="ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ/., -1234567890()"
0040 PAGE
0050 PRINT AT 2,3: "INDTAST cpr. nr. (XXXXXX-XXXX) --> "
0060 REPEAT
0070   PRINT AT 2,40: SPC$(40)
0080   PRINT AT 2,40: "<           >"
0090   CURSOR 2,41
0100   INPUT "": buf$
0110   indcheck
0120 UNTIL ok#
0130 PRINT AT 3,12: "Cpr. Nr. ";buf$;" er testet til OK."
0140 PRINT AT 4,3: "INDTAST navn (efternavn/fornavn) --> "
0150 REPEAT
0160   PRINT AT 4,40: SPC$(40)
0170   PRINT AT 4,40: "<                                  >"
0180   CURSOR 4,41
0190   INPUT "": buf$
0200   tegntest(tegn$(1:29)+tegn$(33:34))
0210   IF "  " IN buf$ OR NOT " " IN buf$ OR LEN(buf$)<6 THEN
```

```
          ok#:=FALSE
0220  UNTIL ok#
0230  PRINT AT 5,12: "Navnet ";buf$;" er testet til OK."
0240  END
0250  //
0260  PROC tegntest(tegn$)
0270    ok#:=TRUE
0280    IF buf$<>"" THEN
0290      FOR tegn#:=1 TO LEN(buf$) DO ok#:=ok# AND buf$(tegn#)
          IN tegn$ AND buf$(1)<>" "
0300    ENDIF
0310  ENDPROC tegntest
0320  //
0330  PROC indcheck
0340    tegntest(tegn$(34:44))
0350    IF buf$<>"" THEN
0360      ok#:=ok# AND LEN(buf$)=11
0370      IF ok# THEN ok#:=buf$(7)="-" AND (NOT "-" IN
          buf$(1:6) AND NOT "-" IN buf$(8:11))
0380      IF ok# THEN IF VAL(buf$(1:2))>31 OR
          VAL(buf$(3:4))>12 THEN ok#:=FALSE
0390      IF ok# THEN
0400        IF buf$(8:11)="0000" THEN
0410          ok#:=ok#
0420        ELSE
0430          pnrtest
0440        ENDIF
0450      ENDIF
0460    ELSE
0470      ok#:=TRUE
0480    ENDIF
0490  ENDPROC indcheck
0500  //
0510  PROC pnrtest// Modulo 11 testen.
0520    FOR y#:=1 TO 11 DO IF y#<>7 THEN t#(y#):=VAL(buf$(y#))
0530    ktal#:=t#(1)*4+t#(2)*3+t#(3)*2+t#(4)*7+t#(5)*6+t#(6)*5+
        t#(8)*4+t#(9)*3+t#(10)*2; m:=ktal# MOD 11
0540    IF m>1 THEN
0550      ktcf:=11-m
0560    ELSE
0570      ktcf:=m
0580    ENDIF
0590    IF ktcf<>t#(11) THEN ok#:=FALSE
0600  ENDPROC pnrtest
```

VARIABLE og VARIABELTYPER.

COMAL-80/Z80 arbejder med 3 variabeltyper:

1. HELTALS-variable - også kaldet: NUMMERISKE-variable. Angives med et #'tegn til slut i navnet.

Værdien må ikke overskride det lukkede interval: -32767 --- 32767.

2. REELTALS-variable - også kaldet DECIMALTALS-variable.

   Der regnes med 7 betydende cifre, hvis tallene bliver større skiftes automatisk til eksponentiel notation. Største numeriske værdi af eksponenten er 99.

3. TEKST-variable. Angives med et $'tegn til slut i navnet, og skal helst dimensioneres før brug.

   Ved dimensioneringen angives hvor mange tegn, som maximalt kan forekomme som variabelværdi. Og ved dimensionering af en tekstvariabel tildeles den værdien den tomme tekst ("").
   Hvis en tekstvariable ikke dimensioneres, bliver den første gang, den anvendes, automatisk dimensioneret til 80 tegn (=skærmlinjelængden).

Heltalsvariabelværdier fylder 2 byte, reeltalsvariabelværdier 5 byte og tekstvariabelværdier det antal byte, som svarer til dimensioneringen + yderligere 4 byte.

Nedstående program, VARIABLE, anvender alle 3 variabeltyper.

```
0010 // VARIABLE
0020 min#:=-32767; max#:=32767
0030 sum#:=min#+max#
0040 PRINT min#;max#;sum#//Heltalsvariabelnavne ender med
     en #
0050 //
0060 diff:=max#-min#
0070 facit:=diff/(1.22*max#)+1560
0080 PRINT diff;facit//Reeltalsvariabel navne slutter ikke med
                      specialtegn.
0090 //
0100 DIM titel$ OF 80 // Tekstvariable dimensioneres først.
0110 titel$:="BRUGER- og OPSLAGS-MANUAL TIL COMAL-80/Z80."
0120 PRINT titel$// Tekstvariabelnavne ender på $
```

Variabelbegrebet er meget fundamentalt, hvis man vil lære om edb. Det kan anbefales i den indledende undervisning at lave øvelser, som fokuserer på at skelne mellem og forstå betydningen af: VARIABELNAVN, VARIABELVÆRDI, VARIABELTILDELING og VARIABELTYPER.

Man kan i Comal oprette mange variable på en gang samt f. eks. tildele flere variable samme værdi på en gang.

Man skal i så tilfælde oprette (dimensionere) en VEKTOR eller en MATRICE.

Nedstående program, MATRICE, er ved indsættelse af REMARKS gjort selvforklarende:

```
0010 // MATRICE
0020 DIM navn$(20) OF 15// Opretter 20 navne af max 15. tegn.
0030 // navn$(1), navn$(2) ........navn$(20).
0040 // Alle navne tildeles værdien "".
0050
0060 DIM point#(-10:10)// Opretter 21 heltalsvariable.
0070 // point#(-10), point#(-9) ... 0 ...point#(9),point#(10)
0080
0090 DIM tal(10:13)// Opretter 4 reeltalsvariable.
0100 // tal(10),tal(11),tal(12) og tal(13)
0110 // Alle med værdien 0
0120
0130 FOR nr#:=-10 TO 10 DO point#(nr#):=13// Tildeler hele
       vektoren point# værdien 13.
0140
0150 PRINT "Udskrift af point#"
0160 FOR nr#:=-10 TO 10 DO PRINT point#(nr#);
     // Udskriver hele vektoren: point#.
0170 PRINT
0180 PRINT
0190 FOR nr#:=10 TO 13 DO tal(nr#):=1.5*point#(nr#-3)*1.22
0200 //Tildeler hele vektoren tal nogle beregnede værdier.
0210 PRINT "Udskrift af vektoren tal:"
0220 FOR nr#:=10 TO 13 DO PRINT tal(nr#);
     // Udskriver vektor:tal
0230 PRINT
0240 PRINT
0250 DIM matrix(2,3,4)// Opretter en 3-dimensionel
     reeltalsmatrice med plads til 2x3x4 tal = 24 tal,
     samt nulstiller matricen.
0260 // Nu beregnes og indlæses nogle værdier i matrix.
0270 FOR x:=1 TO 2 DO
0280   FOR y:=1 TO 3 DO
0290     FOR z:=1 TO 4 DO
0300       matrix(x,y,z):=(x+y)*z/2
0310     ENDFOR z
0320   ENDFOR y
0330 ENDFOR x
0340 PRINT "Udskrift af matricen matrix"
0350 FOR x:=1 TO 2 DO // Og her udskrives matrix
0360   FOR y:=1 TO 3 DO
0370     FOR z:=1 TO 4 DO
0380       PRINT matrix(x,y,z);
0390     ENDFOR z
0400   ENDFOR y
0410 ENDFOR x
```

TILDELINGSSÆTNINGER.

Med symbolet := kan man i såvel direkte- som sætnings-kommandoer tildele variable værdier.

Sætninger som f. eks.:

```
0200 TÆLLER:=123456
```

kaldes derfor tildelingssætninger.

Når man skriver en tildelingssætning, behøver man kun at skrive et ='tegn. Systemet oversætter det så selv til tildelingslighedstegnet := . Men logisk set er der jo stor forskel på de to begreber, så man bør ikke forfalde til at blande dem sammen.

Programmerne TILDEL1 og TILDEL2 giver nogle eksempler på brug af tildelingssætninger.

```
0010 // TILDEL1
0020 dage#:=7; uger#:=52; måneder#:=12
     // 3 Heltalsværditildelinger
0030 // i samme linje.
0040
0050 // Nedenfor tildeles varible værdier i inputsætninger.
0060 INPUT "dagens nummer (1 - 7).....": dagnr#
0070 INPUT "ugens nummer (1 - 52)......": ugenr#
0080 INPUT "Månedens nummer (1 - 12)...": månednr#
0090 INPUT "årstal (fire cifre)........": årstal#
0100 PAGE
0110 IF dagnr#>0 AND ugenr#>0 AND månednr#>0 AND årstal#>999
       THEN
0120   IF dagnr#<=dage# AND ugenr#<=uger# AND månednr#<=
       måneder# AND årstal#>999 THEN
0130     PRINT "dagens nr. er:.....";dagnr#
0140     PRINT "ugens nr. er.....";ugenr#
0150     PRINT "månedens nr. er:.....";månednr#
0160     PRINT "årstallet er:.....";årstal#
0170   ELSE
0180     PRINT "De indtastede data kan ikke godkendes."
0190   ENDIF
0200 ELSE
0210   PRINT "De indtastede data kan ikke godkendes."
0220 ENDIF


0010 // TILDEL2
0020 DIM dag$(7) OF 10,måned$(12) OF 12
0030
```

```
0040 // Nedenfor indlæses data fra DATASÆTNINGER.
0050 FOR nr#:=1 TO 7 DO READ dag$(nr#)
0060 FOR nr#:=1 TO 12 DO READ måned$(nr#)
0070
0080 PAGE
0090 // Nedenfor udskrives nogle af de indlæste data.
0100 // Nemlig de dage som er "større" end lørdag!
0110 PRINT "Dage, som er 'større end' lørdag:"
0120 FOR nr#:=1 TO 7 DO
0130   IF dag$(nr#)>"lørdag" THEN PRINT dag$(nr#);
0140 ENDFOR nr#
0150 PRINT
0160 PRINT
0170 // Nedenfor udskrives de måneder, som er "mindre"
     end juni.
0180 PRINT "Måneder, som er 'mindre end' juni er:"
0190 FOR nr#:=1 TO 12 DO
0200 IF måned$(nr#)<"juni" THEN PRINT måned$(nr#);
0210 ENDFOR nr#
0220 END
0230 DATA "mandag","tirsdag","onsdag","torsdag","fredag",
         "lørdag","søndag"
0240 DATA "januar","februar","marts","april","maj","juni",
         "juli"
0250 DATA "august","september","oktober","november","december"
```

Men tildelingssætninger kan bruges på mange andre måder i Comal. F. eks.

```
0300 TEKST$:=TEKST1$+" "+TEKST2$+TEKST3$(5:10)
```

Denne sætning vil under kørsel, hvis muligt, tildele TEKST$ en værdi, som er sammensat af: TEKST1$'s værdi + et mellemrum + TEKST2$'s værdi + værdien af 5,6,7,8,9 og 10'tegn i TEKST3$

```
0400 NR:=VAL(TEKST$(3:6)
```

Og denne sætning vil om muligt tildele reeltalsvariablen NR talværdien sammensat af tegnene fra og med nr 3 til og med nr. 6 i TEKST$.

DATABEHANDLING.

De væsentligste former for databehandling kan henføres til kombinationer af følgende elementære former for databehandling:

1. Beregninger.

2. Testninger. (Betingede sætninger).

3. Flytninger af data. (Internt i datamaten eller mellem dens enheder, eller til/fra ydre enheder.

- Oprettelse, rettelser og søgninger i databaser
- sortering af data
- udskrivning af f. eks. lagerregnskab

er eksempler herpå.

I COMAL-80/Z80 findes en lang række meget forskellige strukturer, som gør det muligt at udnytte disse for datamaten grundlæggende egenskaber til det yderste.

Der er i det følgende foretaget en hovedgruppering af disse strukturer, samt givet nogle eksempler på deres anvendelsesmuligheder.

BETINGEDE SÆTNINGER.

De basale betingede strukturer i COMAL-80/Z80 er:

IF-sætninger (herunder ELIF-sætninger) og CASE-sætninger (herunder WHEN-sætninger).

Betingede sætninger kan med de logiske operatorer: AND, OR, NOT, m.fl. kobles sammen til sammensatte betingede sætninger.

Ligeledes anvendes betingede strukturer i - og kan indbygges i - mange andre strukturer. (F. eks. i gentagelsesstrukturer).

En betinget sætning indeholder et UDSAGN (eller et UDTRYK), der også kan opfattes som en betingelse, der bliver testet, når sætningen udføres. Resultatet af testningen vil altid være enten TRUE eller FALSE.

I det følgende eksempel kan ses nogle typiske anvendelser af IF- og CASE-strukturer. Eksemplet viser, hvordan man kan indbygge datoinputkontrol i et program, og da selve programmet er skrevet i en lukket procedure, kan det let indbygges i andre programmer.

```
0010 // DATOKTR
0020 DIM dato$ OF 25
0025 //
0030 datoind(dato$)
0035 //
0040 PRINT AT 14,10: "Den godkendte dato er: ";dato$
0050 //
0060 PROC datoind(REF dato$) CLOSED
0070   DIM buf$ OF 10,månednavn$(12) OF 10,taltegn$ OF 10
0080   taltegn$:="0123456789"
0090   //
0100   PAGE
0110   FOR x#:=1 TO 12 DO
0120     READ månednavn$(x#)
0130   ENDFOR x#
0140   DATA "Januar","Februar","Marts","April","Maj","Juni"
0150   DATA "Juli","August","September","Oktober","November",
         "December"
0160   //
0170   PAGE
0180   PRINT AT 8,10: "INDTAST ÅRSTAL"
0190   REPEAT
0200     ok#:=TRUE
0210     PRINT AT 8,23: SPC$(57)
0220     PRINT AT 8,23: "<    >"
0230     CURSOR 8,24
0240     INPUT "": buf$
0250     IF buf$<>"" THEN
0260       FOR x#:=1 TO LEN(buf$) DO
0270         ok#:=ok# AND buf$(x#:x#) IN taltegn$
0280       ENDFOR x#
0290       ok#:=ok# AND LEN(buf$)=4
0300       IF ok# THEN
0310         IF VAL(buf$)>1985 THEN ok#:=FALSE
0320       ENDIF
0330     ELSE
0340       ok#:=FALSE
0350     ENDIF
0360   UNTIL ok#
0370   år#:=VAL(buf$)
0380   //
0390   PRINT AT 10,10: "INDTAST MÅNED"
0400   REPEAT
0410     PRINT AT 10,23: SPC$(10)
0420     PRINT AT 10,23: "<  >"
0430     CURSOR 10,24
0440     INPUT "": buf$
0450     IF buf$<>"" THEN
0460       IF LEN(buf$)=1 THEN
0470         ok#:=buf$ IN taltegn$
0480       ELIF LEN(buf$)=2 THEN
0490         ok#:=buf$(1:1) IN taltegn$ AND buf$(2:2) IN
               taltegn$
0500       ELSE
0510         ok#:=FALSE
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

```
0520        ENDIF
0530        IF ok# THEN
0540          IF VAL(buf$)<1 OR VAL(buf$)>12 THEN ok#:=FALSE
0550        ENDIF
0560      ELSE
0570        ok#:=FALSE
0580      ENDIF
0590    UNTIL ok#
0600    måned#:=VAL(buf$)
0610    //
0620    CASE måned# OF
0630      WHEN 1,3,5,7,8,10,12
0640      antaldage:=31
0650    WHEN 4,6,9,11
0660      antaldage:=30
0670    OTHERWISE
0680      IF år# MOD 100<>0   THEN
0690        CASE TRUE OF
0700        WHEN år# MOD 4=0
0710          antaldage:=29
0720        OTHERWISE
0730          antaldage:=28
0740        ENDCASE
0750      ELSE
0760        CASE TRUE OF
0770        WHEN år# MOD 400=0
0780          antaldage:=29
0790        OTHERWISE
0800          antaldage:=28
0810        ENDCASE
0820      ENDIF
0830    ENDCASE
0840    PRINT AT 12,10: "INDTAST DAG"
0850    REPEAT
0860      PRINT AT 12,23: SPC$(10)
0870      PRINT AT 12,23: "<  >"
0880      CURSOR 12,24
0890      INPUT "": buf$
0900      IF buf$<>"" THEN
0910        IF LEN(buf$)=1 THEN
0920          ok#:=buf$ IN taltegn$
0930        ELIF LEN(buf$)=2 THEN
0940          ok#:=buf$(1:1) IN taltegn$ AND buf$(2:2) IN
                taltegn$
0950        ELSE
0960          ok#:=FALSE
0970        ENDIF
0980        IF ok# THEN
0990          IF VAL(buf$)=0   THEN ok#:=FALSE
1000        ENDIF
1010      ELSE
1020        ok#:=FALSE
1030      ENDIF
1040      IF ok# THEN ok#:=ok# AND VAL(buf$)<=antaldage
1050    UNTIL ok#
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

```
1060    dag#:=VAL(buf$)
1070    //
1080    dato$:=STR$(dag#)+". "+månednavn$(måned#)+" "+
        STR$(år#)+"."
1090 ENDPROC datoind
```

Som ved alle andre sammensatte comalstrukturer, skrives programlisten automatisk med indrykninger, så det let kan konstateres, om stukturopbygningen er i orden.

Den præcise syntaks for alle betingede strukturer kan ses i den alfabetiske sætningsoversigt.

GENTAGELSESSÆTNINGER.

I COMAL-80/Z80 er det for at gøre det lettere, at bruge datamaten til at udføre den samme proces mange gange indlagt 4 forskellige LØKKE-strukturer:

1. FOR - ENDFOR

   Som principielt bruges til at gentage en programdel et FORUDKENDT antal gange.

2. REPEAT - UNTIL

   Som gentager en programdel, indtil en betingelsen i UNTIL-sætningen er opfyldt.

   En REPEAT-UNTIL-struktur gennemløbes - i modsætning til WHILE-ENDWHILE-strukturen - altid en gang.

3. WHILE - ENDWHILE

   Udfører en programdel, hvis og sålænge betingelsen i WHILE-sætningen er sand.

4. LOOP - ENDLOOP

   Udfører en programdel, indtil der i programdelen mødes en ordre (EXIT, EXIT WHEN eller GOTO label) om at forlade løkken.

Man bør være opmærksom på, at alle strukturer med undtagelse af REPEAT-UNTIL i mange tilfælde med fordel kan anvendes på såkaldt simpel form. D. v. s. i kun en programlinje.

Eksempler:

```
0100 FOR x:= 1 to 200 DO IF x MOD 10 = 0 THEN PRINT
```

```
0300 LOOP 4000 TIMES NULL
```

```
0500 WHILE svar$="J" DO udskriv
```

Fælles for alle 4 strukturer er, at de kan indeholde et program-afsnit, hvis størrelse kun begrænses af pladsen i programlageret.

Repeterende strukturer kan nestes ind i hinanden, men de skal i så fald anbringes helt inden i hinanden.

I den alfabetiske sætningsoversigt kan man se den nøjagtige syntaks, samt eksempler på forskellig anvendelse af de 4 strukturer.

UDDATA og UDSKRIFTER.

I dette afsnit nævnes de væsentligste kommandoer til brug ved udskrifter, herunder opstillinger/formatteringer af udskrifter.

Hensigten er, at gøre brugeren opmærksom på, at disse kommandoer specielt er anvendelige i denne forbindelse.

Den præcise syntaks for den enkelte kommando, kan man så om nødvendigt slå op i den alfabetiske kommandooversigt.

1. SELECT OUTPUT

   Bruges til at vælge udskriftsmedium.

   SELECT OUTPUT "DS:" giver udskrift på dataskærm.

   SELECT OUTPUT "LP:" giver udskrift på printer.

2. PRINT

   Kan bruges i følgende variationer:

   PRINT.....

   PRINT AT.....

   PRINT TAB.....

   PRINT USING.....

   PRINT FILE.....

   PRINT FILE USING.....

   En PRINT-kommando kan anføres efter betingelsen i nogle betingede sætninger.

3. PAGE (=CLEAR)

   Bevirker ved skærmudskrift, at skærmen renses og at cursoren placeres i position (1,1).

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

Ved papirudskrift skiftes side.

4. CURSOR l,p

Placeres cursoren i positionen (linie,position). Der er i alt 25 linier på skærmen og 80 tegn i hver linie.

5. SPC$

Er en funktion, som bruges til at skrive blanktegn, og den er velegnet til at slette et bestemt skærmområde, f. eks. et INPUT-felt.

Eks. PRINT AT 12,40: SPC$(40)

skriver 40 blanktegn i linie 12 og fra position 40 og fremefter.

Bem.: Med funktionerne: CURLIN og CURPOS kan man altid under programkørsel få oplyst, hvor cursoren aktuelt befinder sig.

6. ZONE

bruges til kolonneopstillinger.

7. WRITE FILE

bruges til filudskrift af data i binær-kode.

8. LIST FILE

bruges til filudskrift af data i ascii-kode.

9. UPPER$ og LOWER$

Bruges til hhv. at udskifte små bogstaver med store og omvendt.

I nogle sammenhænge kan det under programkørsel være formålstjenligt at have mulighed for at kunne få nogle linier fra skærmbilledet skrevet ud på papir. Nedstående program viser, hvordan det kan gøres. Programmet ligger på disketten og det kan let indbygges som en procedure i ethvert program.

```
0010 // SK_DUMP
0020 DIM linie$ OF 80
0030 PRINT AT 25,1: " "*79,
0040 INPUT AT 25,1: "Indtast nr. på 1.linje (1 - 25) ":
                     førsteli,
0050 PRINT AT 25,1: " "*79,
0060 INPUT AT 25,1: "Indtast nr. på antal linjer (1 - 25) ":
                     antalli,
0070 PRINT AT 25,1: " "*79,
0080 INPUT AT 25,1: "Indtast linjebredden (1 - 80) ": bredde,
0090 PRINT AT 25,1: " "*79,
0100 INPUT AT 25,1: "Indtast antal tomme afslutningslinjer
                     (1 - 72) ": ekstrali,
0110 dump_skærm(førsteli,antalli,bredde,ekstrali)
0120 //
0130 PROC dump_skærm(fl,al,bred,exl)
0140 OPEN FILE 1,"lp:",WRITE
0150 FOR y:=fl TO fl+al-1 DO
0160 IF bred<80 THEN PRINT FILE 1:
0170 linie$:=""
0180 FOR x:=DPEEK(3160)+y*80-80 TO DPEEK(3160)+y*80+bred-81
          DO
0190 nx:=x BITAND $07FF BITOR $F000
0200 IF PEEK(nx)>31 THEN
0210 linie$:=linie$+CHR$(PEEK(nx))
0220 ELSE
0230 linie$:=linie$+" "
0240 ENDIF
0250 ENDFOR x
0260 PUT FILE 1: linie$
0270 ENDFOR y
0280 FOR y:=1 TO exl DO PRINT FILE 1:
0290 CLOSE
0300 ENDPROC dump_skærm
```

FUNKTIONER og PROCEDURER.

Der findes i COMAL-80/Z80 indbygget en del STANDARDFUNKTIONER, og de kan deles op i:

Talbehandlingsfunktioner og Tekstbehandlingsfunktioner.

I den alfabetiske sætningsoversigt kan man se syntaksen for disse

funktioner. Se f. eks.: TRUE, FALSE, ACS, TRUNC, RND, PEEK, INP, HEX$, BIN$, COS, CHR$, KEY$, GET$, m.fl..

Det karakteristiske for en funktion er, at den ud fra en eller anden operation producerer et eller andet veldefineret resultat.

F. eks. bruges CHR$-funktionen til at omsætte en ascii-værdi til det tilsvarende tegn, ligesom ORD-funktionen virker omvendt.

En funktion aktiveres principielt blot ved at angive dens navn + evt. tilhørende parametre.

Desuden har brugeren mulighed for selv at definere funktioner og procedurer. (Se under FUNC og PROC i den alfabetiske del af manualen).

En SELVDEFINERET funktion skal begynde med en FUNC-sætning og slutte med en ENDFUNC-sætning.

En selvdefineret funktion aktiveres ved anvendelse af funktionsnavnet, og det særlige ved en selvdefineret funktion - i modsætning til en procedure - er, at den returnerer den frembragte værdi gennem procedurenavnet.

Der er mange lighedspunkter mellem funktioner og procedurer, f. eks. med hensyn til strukturopbygning, syntaks og parameteroverførsel. Men i hovedtræk kan man sige, at hvor funktioner som i matematik bruges til at behandle nogle data og derudfra frembringe andre data, så er hensigten med procedurer at opdele et større program i mindre programdele og derved gøre programopbygningen logisk, hensigtsmæssig og overskuelig.

Et PROCEDUREOPBYGGET program vil ofte være skåret over en læst som følgende:

1. En startdel med oplysninger, dimensioneringer, initialiseringer, konstanttildelinger o. lign.

2. Et hovedprogram, som kalder nogle procedurer. Ofte vil der blandt disse være en valgprocedure, som i form af en valgmenu giver brugeren mulighed for at vælge mellem de underprogrammer, som programmet er opdelt i.

3. En del underprogrammer, som er skrevet i hver sin procedure.

4. En del selvdefinerede funktioner og procedurer, som hver har sin opgave. Ofte vil disse opgaver være af kontrollerende og beregnende art.

Funktioner og procedurer kan kaldes fra ethvert sted i et program, og de kan være rekursive, d.v.s. at de også kan kalde sig selv.

Nedstående programeksempel belyser dele af, hvad her er omtalt.

```
0010 // HOTEL
0020 // Dimensioneringer m v.
0030 DIM svar$ OF 3,a$ OF 70
0040 a$:="        ##.stk. af ################### a' ####.##
     kr. = #####.## kr."
0050 DIM menu$(5,2,3) OF 40,pris(5,2,3),køb#(5,2,3)
0060 ialt:=0
0070 RESTORE menukort
0080 //
0090 // N U L S T I L L I N G af købstabel.
0100 FOR x:=1 TO 5 DO
0110   FOR y:=1 TO 2 DO
0120     FOR z:=1 TO 3 DO
0130     køb#(x,y,z):=0
0140     ENDFOR z
0150   ENDFOR y
0160 ENDFOR x
0170 //
0180 // I N D L Æ S N I N G fra datasætningerne i matricer.
0190 FOR x:=1 TO 5 DO
0200   FOR y:=1 TO 2 DO
0210     FOR z:=1 TO 3 DO
0220       READ menu$(x,y,z)
0230       READ pris(x,y,z)
0240     ENDFOR z
0250   ENDFOR y
0260 ENDFOR x
0270 //
0280 // Hovedprogram.
0290 REPEAT
0300   SELECT OUTPUT "DS:"
0310   valg
0320   tilbud
0330   PRINT
0340   PRINT "ØNSKES FLERE TILBUD? (J/N) ";
0350   janej
0360 UNTIL CHR$(tast#) IN "Nn"
0370 regning
0380 END
0390 //
0400 menukort:
0410 DATA "KLAR SUPPE MED BOLLER",15,"TOMATSUPPE",20,
```

```
            "ASPARGESSUPPE",25
0420 DATA "FISKESUPPE",27,"OKSEHALESUPPE",29,"",0
0430 DATA "FLÆSKESTEG",35,"OKSESTEG",47,"LAMMESTEG",56
0440 DATA "RÅDYRSTEG",60,"RENSDYRSTEG",75,"VILDSVINESTEG",105
0450 DATA "CITRONFROMAGE",15,"APPELSINFROMAGE",15,
           "ANNANASFROMAGE",23
0460 DATA "SHERRYFROMAGE",30,"ROMFROMAGE",36,"",0
0470 DATA "MOCCA u/",12,"",0,"",0
0480 DATA "MOCCA m/ småkager",18,"MOCCA m/ kransekage",24,
          "MOCCA m/ gode råd",36
0490 DATA "1. LYS TUBORG",8,"1. GRØN TUBORG",11,"1. HOF",11
0500 DATA "1. GAMMEL CARLSBERG",13,"1. GULD TUBORG",14,
          "1. ELEPHANTBEER",15
0510 //
0520 PROC valg
0530 // V A L G af tilbudstyper.
0540   PAGE
0550   PRINT "HOTEL DRONNINGENS SLOTSPARK har som altid mange
             velsmagende tilbud!"
0560   PRINT
0570   PRINT "Ønsker De tilbud på: SUPPE(1), STEG(2),
             DESSERT(3), KAFFE(4), DRIKKEVARER(5) ?
             TAST: 1,2,3,4 el.5)";
0580   ciffertest(1,5)
0590   valg#:=tast#-48
0600   PRINT
0610   PRINT "Angiv det ønskede prisniveau: Billigt(1),
             Dyrere(2) ";
0620   ciffertest(1,2)
0630   prisniveau#:=tast#-48
0640   PRINT
0650   ENDPROC valg
0660 //
0670 PROC tilbud
0680 // T I L B U D
0690   PAGE
0700   PRINT "HOTEL DRONNINGENS SLOTSPARK har den fornøjelse,
             at tilbyde Dem"
0710   PRINT
0720   IF prisniveau#=1 THEN
0730   PRINT "følgende fra Dagens MENU:"
0740   PRINT
0750   ELSE
0760   PRINT "følgende fra restaurantens eksklusive MENUKORT:"
0770   PRINT
0780   ENDIF
0790   FOR z:=1 TO 3 DO
0800     IF menu$(valg#,prisniveau#,z)<>"" THEN // Hvis der er
           noget i menukassen.
0810       PINT TAB (10): z;".";TAB (15): menu$(valg#,
           prisniveau#,z)
0820       PRINT TAB (15): "til en pris på ALT iberegnet: ";
           pris(valg#,prisniveau#,z);"kr."
0830       PRINT
0840     ENDIF
```

```
0850    ENDFOR z
0860  // K Ø B
0870    PRINT "Ønsker De et af ovenstående tilbud, bedes De
              indtaste tallet foran tilbuddet."
0880    PRINT "Hvis IKKE --->> Tast RETURN"
0890    tilbudtest
0900    IF tast#<>13 THEN
0910      købt#:=tast#-48
0920      PRINT
0930      PRINT AT 18,1: "HVOR MANGE ";menu$(valg#,prisniveau#,
                         købt#);" AF ";pris(valg#,prisniveau#,
                         købt#);" kr. ØNSKER DE";
0940      taltest
0950      køb#(valg#,prisniveau#,købt#):+antal#// Købskasser
              opsummeres.
0960    ENDIF
0970  ENDPROC tilbud
0980  //
0990  PROC regning
1000    PRINT
1010    PRINT "ØNSKES REGNINGEN UDSKREVET PÅ PRINTER? (J/N) ";
1020    janej
1030    printer
1040    IF NOT printer# THEN PAGE
1050    PRINT "         HOTEL DRONNINGENS SLOTSPARK"
1060    PRINT "         Leverandør til det kongelige danske hof"
1070    PRINT
1080    PRINT
1090    PRINT "    R E G N I N G"
1100    PRINT
1110    PRINT
1120    PRINT
1130    FOR x:=1 TO 5 DO
1140      FOR y:=1 TO 2 DO
1150        FOR z:=1 TO 3 DO
1160          IF køb#(x,y,z) AND pris(x,y,z) THEN // Hvis købt,
                og hvis der er noget at købe.
1170            beløb:=køb#(x,y,z)*pris(x,y,z)// Beløb for køb
                af enkelttilbud.
1180            PRINT USING a$: køb#(x,y,z),menu$(x,y,z),
                                pris(x,y,z),beløb
1190            ialt:+beløb// Opsummering af totalbeløb.
1200          ENDIF
1210        ENDFOR z
1220      ENDFOR y
1230    ENDFOR x
1240    PRINT "       ----------------------------------"
1250    PRINT
1260    PRINT USING "     I   A L T:....... #####.## Kr.": ialt
1270    PRINT "       =================================="
1280    PRINT
1290    PRINT
1300    PRINT "         Alt er incl.  Dankort kan anvendes."
1310    PRINT
1320    PRINT "         HOTEL DRONNINGENS SLOTSPARK ser Dem
                        gerne snarest igen!"
```

```
1330   IF printer# THEN PAGE
1340 ENDPROC regning
1350 //
1360 PROC printer
1370   IF CHR$(tast#) IN "Jj" THEN
1380     printer#:=TRUE
1390     SELECT OUTPUT "LP:"
1400   ELSE
1410     printer#:=FALSE
1420     SELECT OUTPUT "DS:"
1430   ENDIF
1440 ENDPROC printer
1445 //
1450 PROC tilbudtest
1460   POKE 256,0
1470   REPEAT
1480     tast#:=PEEK(256)
1490   UNTIL tast#>ORD("0") AND tast#<ORD("4") OR tast#=13
1500 ENDPROC tilbudtest
1505 //
1510 PROC janej
1520   POKE 256,0
1530   REPEAT
1540     tast#:=PEEK(256)
1550   UNTIL CHR$(tast#) IN "JjNn"
1560 ENDPROC janej
1565 //
1570 PROC ciffertest(min#,max#)
1580   POKE 256,0
1590   REPEAT
1600     tast#:=PEEK(256)
1610   UNTIL tast#>=min#+48 AND tast#<=max#+48
1620 ENDPROC ciffertest
1625 //
1630 PROC taltest
1640   REPEAT
1650     PRINT AT 20,1: "."*45
1660     INPUT AT 20,49,2: svar$
1670     IF svar$<>"" THEN
1680       ok#:=TRUE
1690       FOR x#:=1 TO LEN(svar$) DO
1700         ok#:=ok# AND svar$(x#) IN "0123456789"
1710       ENDFOR x#
1720       IF ok# THEN antal#:=VAL(svar$)
1730     ELSE
1740       ok#:=FALSE
1750     ENDIF
1760   UNTIL ok#
1770 ENDPROC taltest
```

PARAMETEROVERFØRSEL: I ovenstående procedure CIFFERTEST kan man se et eksempel på parameteroverførsel i forbindelse med procedurer, ligesom man kan se flere eksempler på anvendelse af procedurer uden parametre.

Under ordet PROC i den alfabetiske kommando- og sætningsoversigt er ud fra eksempler forklaret om: Parameteroverførsel, forskellige parametertyper, lukkede procedurer, lokale og globale variable, m.v.

FILBEHANDLING.

Ligesom det er muligt at skrive programfiler ud på baggrundslageret (og læse dem senere), kan man også udskrive alle mulige andre datamængder på f. eks. diskette og senere læse dem igen.

I den forbindelse er det vigtigt at have et baggrundslager med en rimelig lille tilgangstid (accesstid). Kassettebåndsbaggrundslager vil eksempelvis til mange praktiske formål være en umulig løsning, fordi det tager for lang tid, at finde frem til det sted på båndet, hvor de data, man søger, er placeret.

Der er også stor forskel på, hvor hurtigt forskellige maskiners diskettestationer kan finde frem til ønskede data.

I de senere år har de såkaldte WINCHESTERDISKE vundet stor anvendelse i forbindelse med microdatamater. De har en noget mindre accesstid end diskettestationer, og de har langt større kapacitet.

Men hvis de ikke er udskiftelige som disketter, medfører det problemer i forbindelse med filkopiering.

I det følgende skal vi se på forskellige muligheder for under programafvikling at skrive/læse data på/fra diskettestationer i COMAL-80/Z80.

SKRIVNING I SERIEL FIL

Før en datafil kan benyttes, skal den åbnes. Det gøres ved hjælp af en OPEN-sætning:

```
Eks.:  0010 OPEN FILE 2, "DK0:PRØVE", WRITE
```

Denne linje vil ved kørsel bevirke, at der oprettes en datafil, som der kan skrives serielt i (d.v.s. fra en ende af og i fortsat rækkefølge lige så længe, der er plads på disketten i DK0:.

Datafilen tildeles i diskettens katalog navnet: PRØVE.DTA.

Alle filer i diskettekataloget af typen: ????????.DTA er serielle datafiler.

Samtidigt med oprettelsen af den nye datafil etableres en forbindelseslinje med nummeret 2 mellem CPU'eren og disketten.

Når filen er åbnet, kan man under programafviklingen skrive i en WRITE sætning, som så med det etablerede linjenummer henviser til filen.

```
Eks.:  0020 WRITE FILE 2:  NAVN$, ADR$, TLFNR$, ELEVNR#, 22.45, "
           kr."
```

BEMÆRK:

1) Der kan åbnes og arbejdes med maximalt 8 filer ad gangen.

2) Tilladte filnumre er: 0,1,2,3,4,5,6,7,8,9.

3) 'DK0:' kan på normal vis udelades, hvis maskinen arbejder med denne station. Andre diskettestationer skal angives.

4) Hvis man genåbner en skrivefil, vil der ved skrivning til den ske overskrivning forfra.

5) Hvis man ønsker, at skrive i forlængelse af en lukket seriel fil, kan det gøres, hvis man i OPEN-sætningen erstatter WRITE med APPEND:

```
   Eks.:  OPEN FILE 2, "PRØVE", APPEND
```

6) Når man i et program er færdig med at bruge en fil, er det en god vane at lukke forbindelseslinjen til den, herefter kan linjen bruges til en anden fil. Man lukker en fil ved anvendelse af en CLOSE-sætning.

```
Eksempler: 300 CLOSE FILE 2 // lukker linje 2

           400 CLOSE // lukker alle åbne linjer
```

RANDOMFILER

En RANDOM-fil er en datafil, som man både kan skrive i og læse fra.

Og den er i modsætning til serielle filer delt op i nummererede blokke, således at man har direkte adgang til den enkelte blok - uden at skulle læse sig igennem alle de foranstående først. Herved opnås en væsentlig formindskelse af accesstid, især i situationer, hvor man har behov for at læse/ændre få data i en stor database. RANDOM-filer kaldes derfor også for DIREKTE-filer, eller for filer med direkte tilgang.

En RANDOM-fil åbnes således:

Eks.: 0010 OPEN FILE 2, "DK0:PRØVE", RANDOM XXX

Og man behøver kun at skrive:

10 OPEN #2,"DK0:PRØVE",XXX

systemet oversætter så selv til det, som er skrevet i eksemplet.

XXX er et helt positivt tal, som angiver bloklængden i byte.

Datafilen tildeles i diskettens katalog navnet: PRØVE.RND.

Alle filer i diskettekataloget af typen: ????????.RND er RANDOM-filer.

Bem: Åbning, lukning og filnummerering er stort set ens for alle filtyper.

Med CREATE-sætningen kan man afsætte plads til et bestemt antal blokke i en randomfil, og en RANDOM-fil skal CREATE's før den kan tages i brug.

Bem.: CREATE sletter en evt. eksisterende fil med samme navn!

Eks.: 0010 CREATE "PRØVE", 200, 100

Afsætter plads på disketten til 201 blokke (nummereret: 0 - 200) på hver 100 byte i en randomfil med navnet PRØVE.RND.

Den første blok får tildelt nummer 0!

Hvis man ønsker at udvide en RANDOM-fil, kan det gøres med en EXTEND-sætning, således:

Eks.: 0200 EXTEND "PRØVE", 20, 100

Vil udvide filen PRØVE.RND med yderligere 20 blokke a' 100 bytes.

Randomfiler kan åbnes og lukkes igen og igen, men de skal altid have samme bloklængde.

I hver enkelt af en randomfils blokke placeres dataene serielt og efter samme regler, som gælder for serielle filer. Man kan altså skrive både tekster, tal og heltal i samme blok. Bruger man f. eks. en randomfil til et kundekartotek, vil det være praktisk at tildele hver kunde sin blok. og dermed gemme samtlige oplysninger om vedkommende i en og samme blok.

Man bør altid nøje planlægge, hvilke data der skal være i en blok, hvor meget de hver især skal fylde, hvilken rækkefølge de skal stå i o. lign.. Ud fra denne planlægning dimensionerer man randomfilens bloklængde, så den er mindst mulig.

Blok nr. 0 kan bruges til at opbevare statusoplysninger om filen.

I en datafil fylder en heltalsvariabel 2 bytes, en talvariabel 5 bytes og en tekstvariabel (4 + strenglængden) bytes.

Med SEEK-kommandoen kan man søge en bestemt blok i en randomfil.

Eks.: 400 SEEK 2,345

vil, hvis muligt, få systemet til at pege på blok nr. 345 i den randomfil, som er åbnet med strømlinienummer 2.

I nedstående eksempel på et simpelt kartotek, kan man studere, hvordan kartoteket kan oprettes og vedligeholdes.

```
0010 // KUNDER // KUNDEKARTOTEK I RANDOMFILER.
0020 DIM fornavn$ OF 15,efternavn$ OF 15,gade$ OF 25,
     postdi$ OF 20,buf$ OF 4,tekst$(7) OF 25
0030 FOR x:=1 TO 7 DO READ tekst$(x)// Indlæs ledetekster
0040 TRAP
0050   OPEN FILE 2,"KUNDER",RANDOM 128
0060 HANDLER
0070   PAGE
0080   PRINT AT 10,20: "Vent Et ØJEBLIK, mens et tomt kartotek
       oprettes."
0090   CREATE "KUNDER",51,128
0100   maxtal#:=0; fornavn$:=""; efternavn$:=""; gade$:="";
       postdi$:=""; elevnr:=0; trukket:=0; betalt:=0
0110   OPEN FILE 2,"KUNDER",RANDOM 128
0120   FOR nr#:=1 TO 50 DO skrivfil
0130 ENDTRAP
0140 READ FILE 2,0: maxtal#
0150 //
0160 PAGE
0170 PRINT "DETTE PROGRAM GØR DET MULIGT AT SE, RETTE OG
          SLETTE I ET KUNDEKARTOTEK."
0180 PRINT
0190 PRINT "DER FINDES ";maxtal#;" KUNDER I KARTOTEKET, OG
          DATA OM DEM LIGGER I EN RANDOMFIL I"
0200 PRINT
0210 PRINT "POST/BLOKNUMRENE 1 - ";maxtal#;"."
0220 PRINT
0230 INPUT "Videre: Tast RETURN. ": buf$
0240 LOOP
0250 menu
0260 ENDLOOP
0270 //
0280 PROC menu
0290   PAGE
0300   PRINT AT 2,10: "M  E  N  U"
0310   PRINT AT 3,10: "=========="
0320   PRINT AT 5,4: "1. INDKOD nye kunder."
0330   PRINT AT 7,4: "2. UDSKRIFT af alle kunder."
0340   PRINT AT 9,4: "3. UDSKRIFT enkelte kunder."
0350   PRINT AT 11,4: "4. ÆNDRINGER i kundedata."
0360   PRINT AT 13,4: "5. SLETNING af kunder."
0370   PRINT AT 15,4: "6. PAUSE i kartoteksarbejdet."
0380   PRINT AT 17,4: "7. AFSLUT kartoteksarbejdet."
0390   PRINT AT 21,4: "VÆLG: 1,2,3,4,5,6 ELLER 7: ";
0400   ciffertest(1,7)
0410   CASE tast# OF
0420   WHEN 1
0430     indkunde
0440   WHEN 2
0450     IF maxtal# THEN udkunder
0460   WHEN 3
0470     IF maxtal# THEN søgkunde
0480   WHEN 4
0490     IF maxtal# THEN retkunde
0500   WHEN 5
```

```
0510      IF maxtal# THEN sletkunde
0520    WHEN 6
0530      pause
0540    WHEN 7
0550      slut
0560    OTHERWISE
0570    ENDCASE
0580  ENDPROC menu
0590  //
0600  PROC indkunde
0610    nr#:=0
0620    REPEAT
0630      nr#:+1
0640      læsfil
0650    UNTIL fornavn$=""
0660    IF nr#>maxtal# THEN maxtal#:=nr#
0670    skærmbillede
0680    FOR tast#:=1 TO 7 DO ret
0690    slet
0700    PRINT "SKAL DER FORETAGES RETTELSER I OVENSTÅENDE
              (J/RETURN)?";
0710    janej
0720    IF PEEK(256)=74 THEN retpost
0730    skrivfil
0740  ENDPROC indkunde
0750  //
0760  PROC udkunder
0770    FOR nr#:=1 TO maxtal# DO
0780      PAGE
0790      læsfil
0800      skærmbillede
0810      slet
0820      tast
0830    ENDFOR nr#
0840  ENDPROC udkunder
0850  //
0860  PROC søgkunde
0870    PAGE
0880    slet
0890    PRINT "HVILKET KUNDENUMMER ØNSKER DU AT SE
              (maxnr =";maxtal#,")";
0900    taltest(maxtal#)
0910    nr#:=VAL(buf$)
0920    IF nr# THEN
0930      læsfil
0940      skærmbillede
0950    ENDIF
0960    slet
0970    PRINT "FLERE SKÆRMBILLEDER? (J/RETURN) ";
0980    janej
0990    slet
1000    IF PEEK(256)=74 THEN søgkunde
1010  ENDPROC søgkunde
1020  //
1030  PROC retkunde
```

```
1040   PAGE
1050   slet
1060   PRINT "HVILKET KUNDENUMMER ØNSKER DU AT RETTE
             (maxnr =";maxtal#,")";
1070   taltest(maxtal#)
1080   nr#:=VAL(buf$)
1090   IF nr# THEN
1100     læsfil
1110     skærmbillede
1120     PRINT "(8) INGEN RETTELSER"
1130     retpost
1140     skrivfil
1150   ENDIF
1160   slet
1170   PRINT "SKAL DER RETTES FLERE KUNDER? (J/RETURN)";
1180   janej
1190   IF PEEK(256)=74 THEN retkunde
1200 ENDPROC retkunde
1210 //
1220 PROC retpost
1230   REPEAT
1240     slet
1250     PRINT "Indtast talkode for det som skal rettes
               (1,2,3,4,5,6,7,8) ";
1260     ciffertest(1,8)
1270     slet
1280     ret
1290     slet
1300     PRINT "FLERE RETTELSER? (J/RETURN) ";
1310     janej
1320     slet
1330   UNTIL PEEK(256)=78 OR PEEK(256)=13
1340 ENDPROC retpost
1350 //
1360 PROC sletkunde
1370   PAGE
1380   slet
1390   PRINT "HVILKET KUNDENUMMER ØNSKER DU AT SLETTE
             (maxnr =";maxtal#,")";
1400   taltest(maxtal#)
1410   nr#:=VAL(buf$)
1420   IF nr# THEN
1430     læsfil
1440     skærmbillede
1450   ENDIF
1460   slet
1470   PRINT "SKAL DENNE KUNDE SLETTES? (J/RETURN) ";
1480   janej
1490   IF PEEK(256)=74 THEN
1500     fornavn$:=""; efternavn$:=""; gade$:=""
1510     postdi$:=""; elevnr:=0; kursus:=0; trukket:=0
1520     IF nr#=maxtal# THEN maxtal#:-1
1530     skrivfil
1540     skærmbillede
1550   ENDIF
```

```
1560   slet
1570   PRINT "SKAL DER SLETTES FLERE KUNDER? (J/RETURN)";
1580   janej
1590   IF PEEK(256)=74 THEN sletkunde
1600 ENDPROC sletkunde
1610 //
1620 PROC pause
1630   PAGE
1640   udfyld(""27""3"",7,13,10,70)
1650   skrive(" P A U S E   P A U S E  P A U S E
                P A U S E ",10,17)
1660   tast
1670 ENDPROC pause
1680 //
1690 PROC slut
1700   PAGE
1710   CURSOR 20,10
1720   PRINT "Arbejdet afsluttes nu og filer lukkes."
1730   CLOSE
1740   END
1750 ENDPROC slut
1760 //
1770 PROC ret
1780   IF tast#<8 THEN PRINT AT 20,1: "Indtast ";
                       tekst$(tast#);" ";
1790   IF tast#=1 THEN
1800     INPUT "": fornavn$
1810     ajour(fornavn$)
1820   ELIF tast#=2 THEN
1830     INPUT "": efternavn$
1840     ajour(efternavn$)
1850   ELIF tast#=3 THEN
1860     INPUT "": gade$
1870     ajour(gade$)
1880   ELIF tast#=4 THEN
1890     INPUT "": postdi$
1900     ajour(postdi$)
1910   ELIF tast#=5 THEN
1920     taltest(999)
1930     ajour(buf$)
1940     elevnr:=VAL(buf$)
1950   ELIF tast#=6 THEN
1960     taltest(4)
1970     ajour(buf$)
1980     betalt:=VAL(buf$)
1990   ELIF tast#=7 THEN
2000     taltest(2)
2010     ajour(buf$)
2020     trukket:=VAL(buf$)
2030   ELSE
2040   ENDIF
2050 ENDPROC ret
2060 //
2070 PROC ajour(al$)
2080   PRINT AT tast#*2+1,5: SPC$(75)
```

```
2090    IF tast#>4 THEN
2100      PRINT AT tast#*2+1,5: tekst$(tast#)+":";al$
2110    ELSE
2120      PRINT AT tast#*2+1,5: al$
2130    ENDIF
2140      slet
2150  ENDPROC ajour
2160  //
2170  PROC skrivfil
2180    WRITE FILE 2,0: maxtal#
2190    WRITE FILE 2,nr#: fornavn$,efternavn$,gade$,postdi$,
                          elevnr,betalt,trukket
2200  ENDPROC skrivfil
2210  //
2220  PROC læsfil
2230    READ FILE 2,0: maxtal#
2240    READ FILE 2,nr#: fornavn$,efternavn$,gade$,postdi$,
                          elevnr,betalt,trukket
2250  ENDPROC læsfil
2260  //
2270  PROC ciffertest(min#,max#)
2280    POKE 256,0
2290    REPEAT
2300      tast#:=PEEK(256)
2310    UNTIL tast#>=min#+48 AND tast#<=max#+48//
            Indtil der tastes et af de ønskede cifre
2320    tast#:=tast#-48
2330  ENDPROC ciffertest
2340  //
2350  PROC tast
2360    POKE 256,0
2370    slet
2380    PRINT "Videre aktiver en tast."
2390    REPEAT
2400    UNTIL PEEK(256)<>0
2410  ENDPROC tast
2420  //
2430  PROC janej
2440    POKE 256,0
2450    REPEAT
2460    UNTIL PEEK(256)=74 OR PEEK(256)=78 OR PEEK(256)=13
2470  ENDPROC janej
2480  //
2490  PROC slet
2500    PRINT AT 20,1: SPC$(80)
2510    CURSOR 20,1
2520  ENDPROC slet
2530  //
2540  PROC skærmbillede
2550    PAGE
2560    PRINT "POST/BLOKNUMMER I RANDOMFILEN.......";nr#
2570    PRINT
2580    PRINT "(1)";fornavn$
2590    PRINT
2600    PRINT "(2)";efternavn$
```

```
2610    PRINT
2620    PRINT "(3)";gade$
2630    PRINT
2640    PRINT "(4)";postdi$
2650    PRINT
2660    PRINT "(5) ELEVNUMMER: ";elevnr
2670    PRINT
2680    PRINT "(6) KODE FOR BETALT: ";betalt
2690    PRINT
2700    PRINT "(7) UDTRUKKET: ";trukket
2710    PRINT
2720  ENDPROC skærmbillede
2730  //
2740  DATA "FORNAVN","EFTERNAVN","GADE OG NR.","POSTNUMMER OG
      BY","ELEVNUMMER","KODE FOR BETALT","KODE FOR UDTRUKKET"
2750  //
2760  PROC udfyld(karakter$,y_pos_1,y_pos_2,x_pos_1,x_pos_2)
          CLOSED //
2770    FOR printlinje:=y_pos_1 TO y_pos_2 DO
2780      FOR printpos:=x_pos_1 TO x_pos_2 DO
2790        PRINT AT printlinje,printpos: karakter$
2800      ENDFOR printpos
2810    ENDFOR printlinje
2820  ENDPROC udfyld
2830  //
2840  PROC skrive(txt$,y_pos,x_pos)//
2850    PRINT AT y_pos,x_pos: txt$;
2860  ENDPROC skrive
2870  //
2880  PROC taltest(højst#)
2890    REPEAT
2900      ok#:=TRUE
2910      PRINT AT 20,55: SPC$(25)
2920      PRINT AT 20,55: "<    >"
2930      CURSOR 20,56
2940      INPUT "": buf$
2950      IF buf$<>"" THEN
2960        FOR x#:=1 TO LEN(buf$) DO ok#:=ok# AND buf$(x#:x#)
                        IN "0123456789"
2970        IF ok# THEN IF VAL(buf$)<0   OR VAL(buf$)>højst#
                       THEN ok#:=FALSE
2980      ELSE
2990        ok#:=FALSE
3000      ENDIF
3010    UNTIL ok#
3020  ENDPROC taltest
```

AUTOMATISK PROGRAMSKIFT

Ved hjælp af en CHAIN-sætning kan man få et comalprogram til at LOAD'e et andet program og RUN'e det.

Når datamaten under programkørsel støder på en CHAIN-sætning, undersøges det, om den ønskede SAV-fil er på disketten, og hvis det er tilfældet, slettes programlageret, og det nye program hentes ind og køres.

Fordelene herved er, at programdele, som er så store, at de ikke kan afvikles under et, sammenkobles, og derved kommer til at optræde som en helhed.

Man kan i høj grad sammenligne, det der sker, med procedurekald. Blot skal man være opmærksom på, at variabelnavne og værdier ikke overføres. Det program, som kaldes (CHAIN'es ind), RUN'es forfra, hvorved alle variable skal erklæres, dimensioneres, tildeles værdier, etc.

EKSEMPEL:

```
0010 // S
0020 CHAIN "EKS7"
```

Hvis dette program køres, vil det om muligt LOAD'e programmet EKS7.SAV fra DK0:.

Bemærk:

1) Hvis det ønskede program ligger på DK1: skal det angives således:

```
0020 CHAIN "DK1:EKS7"
```

2) CHAIN-sætninger kan kun håndtere SAV-filer.

3) CHAIN-sætninger lukker ikke filer, nulstiller ikke ZONE, ændrer ikke på aktuel enhedsstatus for logaritmefunktionen eller for vinkelfunktionen.

Alfabetisk oversigt

ALFABETISK OVERSIGT.

Her gives en liste over alle de ord, COMAL-80/Z80 forstår. De er listet i den rækkefølge, som de forklares i manualen.

//	ABS	ACS	AF	AND	APPEND
ASN	AT	ATN	AUTO	BC	BIN$
BITAND	BITOR	BITXOR	CALL	CASE	CAT
CHAIN	CHANGE	CHR$	CLEAR	CLOSE	CLOSED
CODE	CON	COS	CREATE	CURLIN	CURPOS
CURSOR	DATA	DE	DEL	DELETE	DIM
DIR	DISCARD	DISPLAY	DIV	DO	DOWNTO
DPEEK	DPOKE	DS	EDIT	ELIF	ELSE
EMPTY$	END	ENDCASE	ENDFOR	ENDFUNC	ENDIF
ENDLOOP	ENDPROC	ENDTRAP	ENDWHILE	ENTER	EOD
EOF	ERR	ERRTEXT$	ESC	EXEC	EXIT
EXP	EXTEND	FALSE	FILE	FIND	FLOAT
FOR	FRAC	FREE	FUNC	GET$	GOTO
HANDLER	HEX$	HL	IF	IMPORT	IN
INP	INPUT	INT	KEY$	LABEL	LEN
LET	LINK	LIST	LN	LOAD	LOG
LOGBASE	LOOP	LOWER$	MAXMEM	MERGE	MOD
MOUNT	NEW	NEXT	NOT	NULL	OF
OPEN	OR	ORD	OFF	OTHERWISE	OUT
OUTPUT	PAGE	PEEK	PI	POKE	PRINT
PROC	PROTECT	PUT	QUIT	RAND	RANDOM
RANDOMIZE	READ	REF	RENAME	RENUM	REPEAT
REPORT	RESTORE	RETURN	RND	ROUND	RUN
SAVE	SCAN	SEEK	SELECT	SETDEG	SETEXEC
SETGRAD	SETLABEL	SETLET	SETRAD	SGN	SIN
SIZE	SPC$	SQR	STEP	STOP	STR$
SYMBOLS	TAB	TAN	THEN	TIME$	TIMER
TIMES	TO	TRACE	TRAP	TRUE	TRUNC
UNIT	UNIT$	UNTIL	UPPER$	USE	USER
USING	VAL	WHEN	WHILE	WRITE	ZONE

//

SYMBOLET

// angiver, at den efterfølgende tekst skal opfattes som en kommentar, dvs. en meddelelse til brugeren af programmet. En kommentar berører ikke på nogen måde udførelsen af programmet, men tager dog plads i lageret. Ved at vælge passende navne på procedurer og variable kan man i vidt omfang spare kommentarer.

En kommentar kan naturligvis indeholde så mange foranstillede blanktegn, det skal være, men efterstillede blanketegn vil blive slettet, da de jo alligevel ikke er synlige i en udlistning.

ABS

er en indbygget funktion. Dens syntaks er:

ABS(<numerisk udtryk>)

Funktionen giver den numeriske værdi af <numerisk udtryk>.

ACS

Er en indbygget funktion. Dens syntaks er:

ACS(<numerisk udtryk>)

hvor det numeriske udtryk skal have en værdi i det lukkede interval fra -1 til 1. Den returnerer arcus cosinus til <numerisk udtryk>. Ved hjælp af sætningerne/kommandoerne SETDEG, SETGRAD og SETRAD (se disse) kan man få resultatet i hhv. grader, nygrader eller radianer.

Se også SETDEG, SETGRAD og SETRAD.

AF

kan bruges på to måder. Som sætning/kommando med følgende syntaks:

AF <numerisk udtryk>

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Værdien af <numerisk udtryk> fortolkes som to bytes, af hvilke den høje byte tildeles Z80'erens A register, og den lave byte tildeles F registret.

Man kan også benytte AF som funktion. Den giver da indholdet af registerparret AF.

Eksempel. Når denne sætning/kommando:

AF515

udføres, får A værdie 2 og F værdien 3. Følgende program:

```
0010 AF $FFFE
0020 PRINT AF
```

giver udskriften: 65534

Se også appendix om maskinkode i COMAL-80/Z80.

AND

er en logisk operator, som udtrykker "logisk og".

Se iøvrigt appendix om udtryk i COMAL-80/Z80.

APPEND

bruges til at angive fortsat skrivning i en sekventiel fil. Se nærmere under OPEN.

ASN

er en indbygget funktion. Dens syntaks er:

ASN (<numerisk udtryk>)

hvor det numeriske udtryk skal have en værdi i det lukkede interval fra -1 til 1. Den giver arcus sinus til <numerisk udtryk>. Ved hjælp af sætningerne/kommandoerne SETDEG, SETGRAD og SETRAD (se disse) kan man få resultatet i hhv. grader, nygrader eller radianer.

Se også SETDEG, SETGRAD og SETRAD.

AT

bruges til at angive skrivepositionen i en PRINT sætning/kommando. Tillige bruges den til at angive en position, fra hvilken indtastning af data skal finde sted i en INPUT sætning.

Se nærmere under PRINT og INPUT.

ATN

er en indbygget funktion. Dens syntaks er:

ATN(<numerisk udtryk>)

hvor det numeriske udtryk skal have en værdi i det lukkede interval fra -1 til 1. Den giver arcus tangens til <numerisk udtryk>. Ved hjælp af sætningerne/kommandoerne SETDEG, SETGRAD og SETRAD (se disse) kan man få resultatet i hhv. grader, nygrader eller radianer.

Se også SETDEG, SETGRAD og SETRAD.

AUTO

er en kommando, som får COMAL -80 systemet til at sætte linienumre automatisk under programindtastning. Efter at AUTO kommandoen er givet, skriver systemet:

0010

og venter på, at der bliver indtastet en linie. Når linien er blevet afsluttet med et tryk på RETURN tasten, svarer systemet med:

0020

og afventer endnu en linie. Man afmelder AUTO ved at trykke på <ESC> tasten.

Hvis ordet AUTO efterfølges af et lovligt linienummer, starter listen af de frembragte linienumre med dette. F.eks. får kommandoen

```
AUTO 110
```

systemet til at frembringe følgende linienumre:

```
0110, 0120, 0130, ...
```

Tilføjer man et komma, efterfulgt af endnu et heltal, bruger systemet dette som tilvækst for de frembragte linienumre. Således får kommandoen:

```
AUTO, 2
```

systemet til at frembringe denne række linienumre:

```
0010, 0012, 0014, ...
```

og

```
AUTO 110,2
```

får systemet til at nummerere de indtastede linier med:

```
0110, 0112, 0114, ....
```

Hvis der allerede findes programlinier, når man bruger AUTO kommadoen, starter den automatiske linienummer plus tilvæksten. Hvis altså den hidtil sidste linies nummer 138, giver fortsat indtastning med AUTO følgende numre:

```
0148, 0158, 0168, ....
```

Såfremt den automatiske nummerering under indtastningen rammer nummeret på en linie, som findes i forvejen, gør systemet - ved at skrive linjenummer i invers skrift (kun på 4. MHz maskiner) - brugeren opmærksom på, at linien findes.

Den automatiske linienummerering afbrydes med et tryk på <ESC> tasten.

BC

kan bruges på to måder. Som sætning/kommando med følgende syntaks:

```
BC <numerisk udtryk>
```

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Værdien af <numerisk udtryk> fortolkes som to bytes, af hvilke den høje byte tildeles Z80'erens B register, og den lave byte tildeles C registret.

Man kan også benytte BC som funktion. Den giver da indholdet af registerparet BC.

Eksempel. Når denne sætning/kommando:

```
BC 1541
```

udføres, får B værdien 6 og C værdien 5. Følgende program

```
0010 BC %0001001111
0020 PRINT BC
```

giver udskriften: 79.

Se også appendix om maskinkode i COMAL-80/Z80.

BIN$

er en indbygget tekstfunktion. Dens syntaks er:

```
BIN$(<numerisk udtryk>)
```

hvor <numerisk udtryk> ligger i det lukkede interval fra 0 til 65535. Den giver en tekst, som er 16 tegn lang, og indeholder en binær repræsentation af <numerisk udtryk>.

BITAND

er en logisk operator, som udtrykker "bitvis og".

Se iøvrigt appendix om udtryk i COMAL-80/Z80.

BITOR

er en logisk operator, som udtrykker "bitvis eller".

Se iøvrigt appendix om udtryk i COMAL-80/Z80.

BITXOR

er en logisk operator, som udtrykker "bitvis eksklusivt eller".

Se iøvrigt appendix om udtryk i COMAL-80/Z80.

CALL

er en sætning/kommando med følgende syntaks:

```
CALL <numerisk udtryk>
```

hvor <numerisk udtryk> skal have en værdi i det lukkede interval fra 0 til 65535. Sætningen/kommandoen bevirker, at systemet laver et kald af en maskinkode subrutine, der starter på den adresse <numerisk udtryk> angiver.

Se også appendix om maskinkode i COMAL-80/Z80.

CASE, WHEN, OTHERWISE, ENDCASE

CASE - sætningen bruges som indledning til en CASE struktur, der angiver en flerforgrening. Syntaksen for en CASE sætning er:

```
CASE <udtryk> OF
```

En CASE sætning kan kun anvendes, hvis det sker sammen med en ENDCASE sætning. Syntaksen for ENDCASE sætningen er:

```
ENDCASE
```

Mellem CASE sætninger og ENDCASE sætningen kan der være et vilkårligt antal WHEN sætninger, og til slut kan der angives en OTHERWISE sætning. Synstaksen for en WHEN sætning er:

```
WHEN< udtryk>, <udtryk>....
```

Udtrykkene på denne liste skal være af samme type, som udtrykket i den tilhærende CASE sætning.

Synstaksen for OTHERWISE sætningen er:

```
OTHERWISE
```

CASE strukturen består af en sammensætning af disse sætninger:

```
CASE <udtryk> OF
WHEN <udtryk>,<udtryk>....
  (Programafsnit 1)
WHEN <udtryk>,<udtryk>....
  (Programafsnit 2)
...
WHEN <udtryk>,<udtryk>....
  (Programafsnit n)
OTHERWISE
  (Programafsnit n+1)
ENDCASE
```

Efter at udtrykket i CASE sætningen er blevet udregnet, bliver listen efter den første WHEN sætning undersøgt. Hvis et af udtrykkene giver samme værdi som udtrykket i CASE sætningen, ud-

føres programafsnit 1, og derefter fortsætter programudførelsen med sætningerne efter ENDCASE. Hvis værdien af CASE udtrykket ikke findes blandt udtrykkene i den første WHEN sætning, undersøges listen efter den anden WHEN sætning, findes værdien af udtrykket der, udføres programafsnit 2, og der fortsættes igen efter ENDCASE sætningen.

Er værdien endnu ikke fundet, fortsættes med tredie WHEN liste, etc. indtil den sidste WHEN sætning er testet. Hvis værdien af CASE udtrykket endnu ikke er fundet, bliver programafsnittet efter OTHERWISE sætningen udført. Hvis der ikke er nogen OTHERWISE del, frembringer systemet en fejlmelding, når værdien ikke kan findes blandt nogen af WHEN sætningerne.

Bemærk: at højst et af programafsnitene 1 til (n+1) bliver udført. Skulle det ske, at CASE udtrykkets værdi kan findes på mere end en af WHEN listerne, får kun den første liste sit programafsnit udført.

Programteksten i programafsnittene 1 til n+1), bliver indrykket under udlistning af programmet. Dette søger systemet automatisk for.

CAT

er en kommando, der bruges til at udskrive en fortegnelse over diskettens indhold eller dele af denne. Syntaksen er:

```
CAT
```

eller

```
CAT "<filnavn>"
```

Hvis den første form anvendes får man en fortegnelse over hele disketten. Hvis man tilføjer et filnavn, udskrives kun den del af kataloget, som indeholder det pågældende filnavn.

Man kan også få katalog-udskriften dirigeret til printeren eller til en fil på disketten, ved at bruge SELECT sætningen før brug af CAT (se SELECT).

Eksempel. Kommandoen:

```
CAT "OPG*.*"
```

giver en fortegnelse over alle de filer, hvis navn begynder med 'OPG'.

Se også SELECT.

CHAIN

sætningen/kommandoen har følgende syntaks:

CHAIN <tekst udtryk>

hvor <tekst udtryk> angiver et filnavn (som skal være med filtypebetegnelsen SAV - altså et program, som er SAVE'et). Sætningen/kommandoen bevirker, at programmet i den angivne fil bliver hentet ind i maskinens lager, hvorpå der iværksættes en udførelse af det.

Et program med tilhørende variable, der befinder sig i arbejdslageret, før CHAIN udføres, bliver slettet.

I stedet for kommandoen CHAIN kan man også benytte RUN, men sidstnævnte kan ikke bruges som sætning, og efter RUN skal der stå en <tekst konstant>.

Bemærk: RUN kommandoen lukker filer, sætter ZONE 0, LOGBASE EXP(1) og SETRAD, disse initialiseringer sker IKKE ved CHAIN sætningen/kommandoen.

Se også RUN.

CHANGE

er en kommando med følgende syntaks:

CHANGE <tekst1>,<tekst2>

hvor <tekst1> og <tekst2> er tekstkonstanter. Kommandoen bevirker, at programmet gennemsøges efter et sted, hvor der står tekstsekvensen <tekst1>. Hvis der findes et sted i programmet, hvor dette er tilfældet, listes den pågældende linie ud på skærmen, og <tekst1> blinker. Systemet venter nu på, at brugeren skal afgøre, hvorvidt tekstsekvensen skal udskiftes eller ej. Man kan trykke på en af følgende taster:

<RETURN>

Bevirker at systemet udskrifter <tekst1> med <tekst2>, og fortsætter søgningen efter flere forekomster af <tekst1>.

<SPACE>

Bevirker at systemet IKKE foretager udskiftningen, men fortsætter blot søgningen efter næste forekomst af <tekstl>.

<ESC>

Bevirker at systemet stopper søgning.

<J>

Bevirker at systemet foretager udskiftningen, og giver brugeren mulighed for at foretage yderligere ændringer i linien, ved brug af den normale skærm-editor.

<N>

Bevirker at systemet IKKE foretager udskiftningen, men giver blot brugeren mulighed for at foretage andre ændringer i linien.

Hvis man ønsker, at søgningen skal foretages over hele programmet, kan man angive en programdel, i hvilken søgning og udskiftning skal foretages. Denne facilitet belyses med følgende eksempler. Kommandoen:

```
CHANGE 200 "tekst", "nytekst"
```

søger kun i linie 200. Kommandoen:

```
CHANGE -200 "først", "sidst"
```

søger i alle linier til og med 200. Kommandoen:

```
CHANGE 200- "CLOSED". ""
```

søger i alle linie fra og med linie 200. Og til sidst

```
CHANGE procedure "a", "b"
```

søger i proceduren ved navn procedure.

CHR$

er en indbygget tekstfunktion. Dens syntaks er:

CHR$(<numerisk udtryk>)

hvor <numerisk udtryk> ligger i det lukkede interval fra 0 til 255. Funktionen giver det til <numerisk udtryk> svarede tegn i systemets tegnsæt.

CLEAR

er et nøgleord, der kan benyttes i stedet for PAGE, systemet vil så automatisk ændre det til PAGE.

CLOSE

eller

CLOSE FILE <numerisk udtryk>

Hvis den første form benyttes lukkes alle filer. Tilføjer man et numerisk udtryk lukkes kun til, hvis strømnummer svarer til værdien af det numeriske udtryk.

Nøgleordet FILE kan udelades. Systemet indsætter det da selv.

CLOSED

bruges til at angive, at en procedure eller en funktion er lukket.

Se nærmere under PROC OG FUNC.

CODE

sætningen/kommandoen har følgende syntaks:

CODE <numerisk udtryk>, <numrisk udtryk>,....

hvor <numerisk udtryk> skal have en værdi i det lukkede interval fra 0 til 255.

Hvert enkelt numerisk udtryk bliver fortolket som en byte. Disse bytes bliver opfattet som Z80 maskinkode og udført så snart systemet møder CODE sætningen/kommandoen.

Bemærk: Brugeren behøver ikke at give en kode, svarende til RET instruktionen (201) i hver linie, idet denne automatisk indsættes af systemet. Brugeren skal altså sørge for, at maskinstakkens indhold ved kodens slutning er den samme, som ved kodens start.

Eksempel. Sætningen/kommandoen:

```
CODE $11,x MOD 256, x DIV 256
```

sætter DE registret lig med værdien af variablen x.

CON

er en kommando med følgende syntaks:

```
CON
```

Kommandoen bevirker, at udførelsen af et program genoptages. Udførelsen kan kun genoptages, hvis den blev stoppet med en STOP sætning, eller ved et tryk på <ESC> tasten. På grund af den interne lagring af variable etc. kan denne kommando ikke udføres efter, at man har tilføjet eller fjernet programlinier.

COS

er en indbygget funktion. Dens syntaks er:

```
COS(<numerisk udtryk>)
```

Funktionen giver cosinus af <numerisk udtryk>. Ved hjælp af sætningerne/kommandoerne SETDEG, SETGRAD og SETRAD kan brugeren bestemme, hvorvidt værdien af <numerisk udtryk> angiver hhv. grader, nygrader eller radianer.

Se også SETDEG, SETGRAD og SETRAD.

CREATE

er en sætning/kommando med syntaksen:

```
CREATE <tekst udtryk>, <udtryk1>, <udtryk2>
```

hvor <tekst udtryk> angiver et filnavn, og <udtryk1> og <udtryk-2>angiver hele positive tal.

Sætningen/kommandoen bevirker, at der oprettes en fil med navnet <tekst udtryk> på disketten. I filen reserveres der plads til <udtryk1> poster på hver <udtryk2> tegn med direkte tilgang til hver post (RANDOM fil). Hver af disse poster, svarer til en sekventiel fil, således at når filen er åbnet (se OPEN), kan man søge (se SEEK) hen til post nr. 5, og skrive sekventielt i denne post. Når posten er fyldt op, vil systemet give en fejlmelding. Hvis man ønsker at udvide eller WRITE sætningerne/kommandoerne.

Se også OPEN, SEEK, EXTEND, WRITE, READ.

CURLIN

er en funktion. Dens syntaks er:

```
CURLIN
```

Funktionen angiver i hvilken linie markøren befinder sig.

CURPOS

er en funktion. Syntaksen er:

```
CURPOS
```

Funktionen angiver i hvilken position på linien markøren befinder sig.

CURSOR

er en sætning/kommando med syntaksen:

```
CURSOR <udtryk1>, <udtryk2>
```

hvor <udtryk1> skal være et helt tal, der angiver en linie på skærmen, og <udtryk2> skal være et helt tal, der angiver en positionen på de pågældende linie.

Sætningen/kommandoen bevirker, at markøren bliver flyttet fra sin nuværende position til linie nr. <udtryk1> og position nr. <udtryk2> i denne linie.

Hvis linien eller positionen angives til 0, bruger systemet hhv. nuværende linie eller position til placering af markøren.

Se også PRINT AT og INPUT AT.

DATA

er en sætning, der har syntaksen:

```
DATA <tekst/talkonstant>, <tekst/talkonstant>,....
```

Sætningen i sig selv springes over vil fejludførsel, og den bruges til at anføre konstanter, som skal læses fra READ sætninger (Se READ). Konstanterne adskilles med kommaer. DATA sætningerne kan placeres hvor som helst i programmet, og organiseres ved programmets start (RUN/SCAN) i en datakø, hvis elementer bliver læst i den rækkefølge, i hvilken de optræder i programmet. Talkonstanter kan skrives med fortegn. Tekstkonstanter skal være omsluttet af anførselstegn. Således vil:

```
DATA "Freddy Kristiansen",673529, "Peter Plys",722525
```

Give anledning til en datakø startende med en konstant.

Data, som er anført i en kø indenfor en lukket procedure, er lokale for proceduren. De pågældende data kan altså ikke læses af READ sætninger udenfor proceduren. Man kan dog godt inde i en lukket procedure, IMPORT'ere en label, bruge RESTORE til denne label, og derved læse globale data ind i proceduren.

Hvis man ønsker at begrænse en DATA-kø, dvs. afslutte den efter en bestemt konstant, kan man anføre et END i data sætningen. Når datapilen kommer til at pege på dette END vil EOD blive sat.

Eksempel. En række datakøer, der er organiseret på følgende måde:

```
brik'nr1:
DATA "B", "O", "N", "D", "E", END
brik'nr2:
DATA "T", "Å", "R", "N", END
...
```

Kan således læses på følgende måde:

```
RESTORE brik'nr1
læsdata
RESTORE brik'nr2
læsdata
...
PROC læsdata
  WHILE NOT EOD DO
    READ A$
    PRINT A$,
  ENDWHILE
ENDPROC læsdata
```

DE

kan bruges på to måder. Som sætning/kommando med følgende syntaks:

DE <numerisk udtryk>

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Værdien af <numerisk udtryk> fortolkes som to bytes, af hvilken den høje byte tildeles Z80'erens D register, og den lave byte tildeles E registret.

Man kan også benytte DE som funktion. Den giver da indholdet af registerparret DE.

Eksempel. Når denne sætning/kommando:

DE 61265

udføres, får D værdien 239 og E værdien 81.

Se også appendix om maskinkode i COMAL-80/Z80.

DEL

kommandoen bruges til at fjerne en eller flere linier fra programmet i maskinens lager. F.eks. fjerner kommandoen:

DEL 100

linie 0100. Kommandoen:

DEL 20-30

fjerner alle linierne mellem 0020 og 0030 incl. Kommandoen:

DEL -300

fjerner alle linier til og med linie 0030, og kommandoen:

DEL 300-

fjerner alle linierne fra og med linie 0300. Til slut vil kommandoen:

```
DEL udskrift
```

fjerne hele proceduren/funktionen ved navn udskrift.

Man kan desuden kæde flere linieangivelser sammen med kommaer. F.eks.

```
DEL dummy,20,300-340
```

fjerner proceduren/funktionen dummy, linie 0020, og alle linier mellem 0300 og 0340 incl.

DEL kommandoen kan også bruges til at slette TRACE sætninger, der er indsat i et program. Kommandoen:

```
DEL TRACE
```

vil fjerne alle TRACE sætninger i det indeværende program.

DELETE

er en sætning/kommando med følgende syntaks:

```
DELETE "<filnavn>"
```

sætningen/kommandoen bevirker, at filen med navnet <filnavn> bliver slettet fra disketten.

DIM

er en sætning/kommando, der bruges til at erklære tabeller og tekstvariable.

Tekstvariable, der benyttes uden først at blive dimensioneret, får automatisk en dimensionering på et antal tegn, der svarer til bredden af skærmen. Hvis man ønsker at spare plads, eller ønsker at bruge en tekstvariabel, der er længere end skærmbredden, skal denne dimensioneres:

```
DIM tekst$ OF 1
```

bevirker således at tekst$ maksimalt kan indeholde 1 tegn.

```
DIM nytekst$ OF 1000
```

I COMAL-80/Z80 kan man erklære tabeller med et hvilket som helst antal dimensioner og indeksnummereringen behøver ikke nødvendigvis at begynde med 0 eller 1. Følgende eksempler skal belyse brugen af DIM sætningen/kommandoen:

```
DIM tabel(100)
```

erklærer en tabel af reele tal med indices gående fra 1 til 100. Ønsker man at nedre grænse skal være 0, skal man skrive:

```
DIM tabel(0:100)
```

Man kan bruge et hvilket som helst heltal som nedre eller øvre grænse, når blot det første er mindre end eller lig med det andet.

```
DIM felter(-7:7)
```

erklærer en taltabel med indices fra -7 til 7, incl.

```
DIM elev$(30:100,8:10)
```

erklærer en todimensional teksttabel med indices fra 30 til 100 og 8 til 10.

Hver komponent kan indeholde et antal tegn, svarende til skærmbredden.

Hvis man ønsker at ændre på antal tegn pr. komponent, skal man som før nævnt blot tilføje et OF og et numerisk udtryk.

```
DIM navn$(30:100,8:10) OF 30
```

DIR

er en sætning/kommando der har følgende syntaks:

```
DIR
```

eller som sætning:

```
0010 a$:="EKS*.*"
0020 DIR a$
```

hvor a$ angiver del af et filnavn. Hvis den første form anvendes, får man en fortegnelse over hele disketten. I det andet tilfælde udskrives kun den del af kataloget, som indeholder den pågældende del af filnavn.

Denne sætning virker præcis som kommandoen CAT, bortset fra at CAT ikke kan bruges som sætning, og at CAT kun må efterfølges af en tekstkonstant.

Se iøvrigt CAT.

DISCARD

er en kommando med syntaksen:

```
DISCARD
```

eller

```
DISCARD "<pakkenavn>"
```

Hvis man anvender den første form, bliver alle maskinkode-pakker, der findes i hukommelsen fjernet. Hvis DISCARD efterfølges af et pakkenavn fjernes kun den pakke, der har det pågældende navn.

Se også appendix om pakker i COMAL-80/Z80.

DISPLAY

kommandoen giver en programlistning (se LIST) uden linienumre, men med den sædvanlige strukttur-indrykning.

Kan f. eks. anvendes til udskrift af programtekst i tekst i tekstbehandlingssystem.

Se nærmere under LIST.

DIV

er en operator. Dens syntaks er:

<udtryk1> DIV <udtryk2>

hvor begge udtryk er reele udtryk. Resultatet er heltalsdivisionen af <udtryk1> med <udtryk2>. Den er defineret ved:

<udtryk1> DIV <udtryk2> = INT (<udtryk1>/<udtryk2>)

Se også MOD og appendix om udtryk i COMAL-80/Z80.

DO

er et nøgleord, der er beregnet til at afslutte en FOR eller en WHILE sætning. Brugeren behøver aldrig at indtaste DO. Hvis det mangler vil systemet selv sætte det.

Se også FOR og WHILE.

DOWNTO

er et nøgleord, der bruges i forbindelse med FOR løkker.

Se FOR.

DPEEK

er en indbygget funktion med syntaksen:

DPEEK (<numerisk udtryk>)

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Funktionen bruges til at hente en adresse fra lageret. DPEEK er defineret ved:

DPEEK(<udtryk>) = PEEK (<udtryk>) + 256*PEEK (<udtryk>+1)

Se også DPOKE og PEEK

DPOKE

er en sætning/kommando. Syntaksen er:

DPOKE <udtryk1>,<udtryk2>

hvor <udtryk1> og <udtryk2> skal være hele tal i det lukkede interval fra 0 til 65535. Sætningen/kommandoen anvendes til at lagre en 2 bytes adresse i hukommelsen, efter standard z80 metode (lav byte først).

DPOKE er defineret ved:

DPOKE <udtryk1>,<udtryk2> = POKE <udtryk1>,<udtryk2> MOD 256
POKE <udtryk1>+1,<udtryk2> DIV 256

DPOKE er således den omvendte funktion til DPEEK, præcis som POKE og PEEK er hinandens omvendte funktioner.

Se også POKE og DPEEK.

DS

er en sætning. Dens syntaks er:

DS -

eller

DS +

Hvis man anvender den første form vil al udskrivning til skærmen blive slået fra. Skrivning til skærmen kan genoptages med den anden form af DS sætningen eller ved at systemet bliver bragt ud i skærm-editoren.

EDIT

kommandoen bruges til at udskrive programlinierne fra maskinens lager, men uden den indrykning, som ellers laves af LIST kommandoen. Når EDIT har udskrevet en linie, placeres markøren i begyndelsen af denne linie, og brugeren får mulighed for at lave rettelser i linien. Når RETURN tasten aktiveres, vil maskinen kontroller linien, og hvis alt er i orden indlæses linien.

Afbrydelse af EDIT kommandoen sker ved at trykke på <ESC> tasten.

EDIT kan også bruges til at editer enkeltlinier eller programafsnit.

F. eks.: EDIT 345

eller EDIT 2000-

ELIF

er en sætning, der har følgende syntaks:

ELIF <numerisk udtryk> THEN

hvor <numerisk udtryk> angiver en sandhedsværdi,TRUE eller FALSE.

Sætning, bruges som en del af en IF-struktur.

Se nærmere under IF.

ELSE

er en sætning, der har syntaksen:

ELSE

Sætningen bruges som en del af en IF-struktur.

Se nærmere under IF.

EMPTY$

er en tekstfunktion. Dens syntaks er:

```
EMPTY$
```

Funktionen giver den tomme tekst ("") som resultat.

END

er en sætning med syntaksen:

```
END
```

eller END <tekst udtryk>

Hvis man benytter den første form, standses udførelsen af programmet, men uden den sædvanlige meddelelse:

```
Programudførelse afsluttet.
```

Hvis man sætter et tekst udtryk efter END, bliver dette udtryk udskrevet fremfor den normale meddelelse.

Nøgleordet END kan også anvendes i forbindelse med DATA, i dette tilfælde bruges det til at afslutte en data-kø.

Se også DATA.

ENDCASE

er en sætning. Dens syntaks er:

```
ENDCASE
```

Sætningen bruges til at afslutte en CASE-struktur..

Se også CASE

ENDFOR

er en sætning med syntaksen:

```
ENDFOR <variabel>
```

Sætningen bruges til at afslutte en FOR-struktur.

Se også FOR.

ENDFUNC

er en sætning med følgende syntaks:

```
ENDFUNC <funktionnavn>
```

Sætningen bruges til at afslutte en brugerdefineret funktion, der er indledt med FUNC sætningen.

Se også FUNC

ENDIF

er en sætning med følgende syntaks:

```
ENDIF
```

Sætningen bruges til at afslutte en IF-struktur.

Se også IF.

ENDLOOP

er en sætning med syntaksen:

ENDLOOP

Sætningen bruges til at afslutte en LOOP-struktur.

Se også LOOP.

ENDPROC

er en sætning, der har følgende syntaks:

ENDPROC <procedurenavn>

Sætningen bruges til at afslutte en brugerdefineret procedure, der er indledt med PROC sætningen.

Se også PROC.

ENDTRAP

er en sætning, der har syntaksen:

ENDTRAP

Sætningen bruges til at afslutte en TRAP-struktur.

Se også appendix om fejlhåndtering i COMAL-80/Z80.

ENDWHILE

er en sætning. Dens syntaks er:

ENDWHILE

Sætningen bruges til at afslutte en WHILE-struktur.

Se også WHILE.

ENTER

kommandoen har syntaksen:

```
ENTER "<filnavn>"
```

Kommandoen bevirker at programmet i filen <filnavn> bliver hentet ind i arbejdsområdet. Med ENTER kan man kun indhente programmer skrevet i ASCII-kode. D.v.s. programmer, der er gemt med LIST- eller DISPLAY-kommandoer, som .LST-filer.

Bem.: "Gamle" .CML programmer kan også hentes ind, hvis man angiver filtypen, og hvis de er skrevet i ren ASCII-kode.

Bemærk: Eventuelle programmer, der findes i arbejdsområdet før ENTER kommandoen udføres, bliver slettet når det nye program hentes ind. Det program, som hentes ind får linienumre fra 0010 og opefter med tilvækst på 0010. Hvis man ønsker at kæde programmer sammen eller at give programmet andre linienumre kan man bruge MERGE kommandoen.

Se også MERGE.

EOD

er en indbygget funktion. Dens syntaks er:

```
EOD
```

EOD (End-Of-Data) er en logisk funktion, der giver TRUE, hvis der ikke er flere elementer i datakøen, ellers FALSE.

EOF

er en indbygget funktion, der har syntaksen:

```
EOF (<strømnummer>)
```

EOF (End-Of-File) er en logisk funktion, der giver TRUE, hvis det sidste element i filen, angivet ved <strømnummer> er læst, ellers FALSE.

ERR

er en indbygget funktion. Dens syntaks er:

```
ERR
```

Funktionen giver fejlnummeret, hvis der opstår en fejl, mens fejlhåndteringsstruktur er aktiv.

Se appendix om fejlhåndtering i COMAL-80/Z80.

ERRTEXT

er en tekstfunktion. Syntaksen er:

```
ERRTEXT$(<numerisk udtryk>)
```

Funktionen giver fejlteksten, svarende til fejlen med nummeret som det numeriske udtryk angiver.

Se også appendix om fejlhåndtering i COMAL-80/Z80.

ESC

er en logisk funktion, der kun er brugbar når <ESC> tastens værdi er sat ud af funktion (Se TRAP ESC). Syntaks:

```
ESC
```

Hvis <ESC> tasten har været nedtrykket vil værdien være TRUE, ellers FALSE.

Bemærk: Når ESC funktionen bruges bliver den nulstillet.

EXEC

er et nøgleord, der kan sættes foran et procedurenavn i et kald til proceduren. Hvis nøgleordet anvendes, får procedurekaldet følgende syntaks:

EXEC <procedure navn> (< liste over aktuelle parametre>)

Normalt bruges nøgleordet ikke i COMAL-80/Z80, men af hensyn til fordrageligheden med ældre versioner af COMAL, accepteres ordet af systemet og kan under listningen gengives, hvis kommandoen SETEXEC (se SETEXEC) bruges.

En nærmere beskrivelse af procedurer og parametre findes under PROC.

Se også PROC og SETEXEC.

EXIT, EXIT WHEN

er to sætninger. Syntaksen for den første form er:

EXIT

og syntaksen for den anden form er:

EXIT WHEN <numerisk udtryk>

hvor <numerisk udtryk> angiver en sandhedsværdi. Hvis den første form bruges, foretager systemet et udhop fra den af de følgende løkker, systemet befinder sig i:

FOR - ENDFOR

LOOP - ENDLOOP

REPEAT - UNTIL

WHILE - ENDWHILE

Den anden form benyttes hvis man ønsker at lave et betinget udhop fra en af løkkerne. Hvis <numerisk udtryk> udregnes til TRUE foretages udhoppet, hvis ikke fortsættes programudførelsen med den efterfølgende linie.

EXP

er en indbygget funktion. Dens syntaks er:

```
EXP(<numersik udtryk>)
```

Funktionen beregner grundeksponentialfunktionen.

Man har altså "e i x'te", hvor x er det numeriske udtryk.

EXTEND

sætningen/kommandoen har syntaksen:

```
EXTEND <tekst udtryk>,<udtryk1>,<udtryk2>
```

hvor <tekst udtryk> angiver et filnavn, og <udtryk1> og <udtryk2> er hele positive tal. Sætningen/kommandoen bruges, hvis man opererer med en RANDOM fil og ønsker at udvide denne fil med et antal poster.

Som ved CREATE sætningen/kommandoen (se denne) angiver <udtryk1> antallet af poster, filen skal udvides med, og <udtryk2> angiver postlængden.

Eksempel.Kommandoen.

```
EXTEND "elever",100,80
```

Udvider filen under navnet 'elever' på disketten, med 100 poster, der har postlængden 80.

Forskellen på EXTEND og CREATE er; at CREATE sletter en eventuelt eksisterende fil og udfører en EXTEND på en tom fil.

FALSE

er en indbygget funktion. Dens simple syntaks er:

```
FALSE
```

Funktionen giver altid værdien 0, hvilket svaret til logisk falsk.

Numeriske udtryk der udregnes til 0 vil have sandhedsværdien FALSE, eller TRUE.

Eksempel. Efter tildelingen

```
slut:=FALSE
```

har det logiske udtryk:

```
tal>max AND slut
```

værdien FALSE. I virkeligheden er 'slut' ikke andet end en numerisk variabel med værdien 0 og kan i øvrigt anvendes som sådan, hvis man ønsker det.

FILE

er et nøgleord, der anvendes i OPEN/CLOSE, READ/WRITE og INPUT /PRINT sætninger/kommandoer i forbindelse med filbehandling. I OPEN, CLOSE og WRITE kan man undlade nøgleordet FILE, hvorefter det automatisk indsættes af systemet.

Se også OPEN, CLOSE, READ, WRITE, INPUT og PRINT.

FIND

er en kommando, der benyttes til søgning af en tekstsekvens i en programliste. Hvis man f.eks. anvender kommandoen:

```
FIND "antal"
```

gennemsøges programlisten, og hver gang tekstsekvensen 'antal' bliver fundet, standses søgningen, og den linie, i hvilken teksten er fundet, bliver udlistet på skærmen med markøren placeret på det første 'a' i 'antal'. Man kan derpå foretage eventuelle rettelser og fortsætte søgning ved at trykke på RETURN tasten.

Søgning afbrydes ved tryk på <ESC> tasten.

Hvis man ønsker at afgrænse et søgeområde i programmet, gøres dette på samme måde som det gøres i CHANGE.

Se også CHANGE.

FLOAT

funktionen har syntaksen:

FLOAT (<numerisk udtryk>)

Funktionen giver <numerisk udtryk>, og er mest brugbar i WRITE sætningen, hvor man jo kan bruge:

WRITE FILE 1:45+x

Den værdi der skrives ud i filen fylder 2 bytes, hvis tallet ligger i det lukkede interval fra 0 til 65535, ellers 5 bytes. Ved at bruge FLOAT funktionen sikrer man sig at tallet fylder 5 bytes.

FOR

sætningen/kommandoen styrer en FOR-løkke. I sin simpleste form er sætningen opbygget således.

FOR <variabel>:=<start> TO <grænse> DO <simpel sætning>
hvor <variabel> er en numeriskvariabel.

Sætningen virker således: <variabel> sættes lig med værdien af <start>. Derpå gentages følgende process: Hvis værdien af <variabel> er mindre end eller lig med værdien af <grænse>, udføres <simpel sætning> og værdien af <variabel> øges med 1. Den gentagne udførelse af <simpel sætning> standser altså, så snart <variabel> antager en værdi, der er større end værdien af <grænse>. Bemærk, at udtrykkene <start> og <grænse> kun udregnes en gang. Udførelsen af løkken har altså ingen sideeffekt på værdien af disse udtryk.

Eksempler:

```
FOR i:=1 TO 5 DO PRINT i

FOR i:=1 TO 5 DO FOR j:=1 TO i DO PRINT i:j
```

FOR-sætningen kan udvides til FOR-struktur, der ser sådan ud:

```
FOR <variabel>:=<start> TO <slut> STEP <tilvækst> DO
    <programafsnit>
ENDFOR <variabel>
```

Strukturen kører igennem på præcis samme måde som FOR-løkken, når den er i en enkelt linie. Bemærk at brugeren ikke behøver at

skrive variabelnavnet efter ENDFOR, dette navn sættes af systemet, når der gennemføres et prepass af programmet (SCAN eller RUN).

FOR-sætningen kan udvides til følgende:

```
FOR <variabel>:=<start> TO <slut> STEP <tilvækst> DO
```

hvor <tilvækst> er et numerisk udtryk. Udførelsen af FOR løkken er som beskrevet ovenfor, men efter hvert gennemløb øges <variabel> med værdien af <tilvækst> i stedet for 1, og hvis <tilvækst> er negativ, standses udførelsen af løkken, når <variabel> antager en værdi, der er mindre end værdien af udtrykket <grænse>.

FOR-sætningen kan også se sådan ud:

```
FOR <variabel>:=<start> DOWNTO <grænse> DO
```

Når DOWNTO bruges, i stedet for TO, bliver <tilvækst> sat til -1 i stedet for 1. Hvis DOWNTO benyttes sammen med STEP < tilvækst>, og <tilvækst> er negativ, vil FOR-løkken gøre præcis det samme, som hvis TO blev brugt, sammen med den numeriske værdi af <tilvækst>!

Eksempler:

```
FOR i:=5 DOWNTO -2 DO PRINT i

FOR j:=65 DOWNTO 72 STEP -1 DO PRINT CHR$(j);

FOR tæller:=10 TO 2 STEP -2 DO udskriv (tæller)
```

Den variabel, der bruges som tæller i FOR-løkken, er LOKAL med hensyn til løkken. Følgende eksempler skal belyse, hvad det betyder:

```
FOR i:=1 TO 10 DO
  FOR i:=1 TO 3 DO PRINT
  PRINT post$ (i)
ENDFOR i
```

Selv om begge tællere har navnet 'i', behandles de af systemet som om de var forskellige. Man får altså udskrevet værdien af post$(1), post$(2), ...,post$(10) hver med tre tomme linier mellem.

```
FOR i:=1 TO 10 DO
  FOR i:=1 TO i DO
    PRINT b (i);
  ENDFOR i
  PRINT post$(i)
ENDFOR i
```

Først får man udskrevet værdien b(1), da den inderste FOR-løkke kun går til 1 i første gennemløb. Derefter udskrives post$(1), og når så systemet når den indre FOR-løkke næste gang, vil den ydre styrevariabel være lig 2. Så udskrives b(1), b(2) og post$(2), etc. Når systemet er færdig med den ydre For-løkke, vil tæller-variablen være udefineret.

FRAC

funktionen har syntaksen:

```
FRAC(<numerisk udtryk>)
```

Funktionen giver brøkdelen af udtrykket. FRAC er defineret som:

```
FRAC(<udtryk>) = <udtryk>-INT(<udtryk>)
```

FREE

er en indbygget funktion med syntaksen:

```
FREE
```

Funktionen giver hvor mange bytes, brugeren har til sin rådighed. Ved udførelsen af en SIZE kommando kan man også se, hvor mange bytes man har tilbage.

FUNC, ENDFUNC, RETURN

FUNC sætningen har denne syntaks:

```
FUNC <funktionens navn> ( <liste med formelle parametre> )
```

ENDFUNC sætningen har denne syntaks:

```
ENDFUNC <funktionens navn>
```

RETURN sætningen har denne syntaks:

```
RETURN <udtryk>
```

En funktion er opbygget efter følgende skema:

```
FUNC <funktionens navn> ( <liste med formelle parametre> )
...
...
RETURN <udtryk>
...
ENDFUNC <funktionens navn>
```

Listen med formelle parametre, kan godt være tom, i så fald får man en parameterløs funktion. Der kan godt være flere RETURN sætninger i funktionens krop, men der skal være mindst een, som kan tilbagegive funktionens værdi.

Funktioner kan være lukkede ligesom procedurer.

Eksempel:

```
FUNC sfd(x,y)
   IF x MOD y=0 THEN RETURN y //hvis y går op i x, er sfd=y
   RETURN sfd(y,xx MOD y) //    brug Euklids algoritme
ENDFUNC
```

Denne funktion udregner den største fælles divisor for x og y, og kan f.eks. kaldes med sætningen/kommandoen:

```
IF sfd(a,b)=1 THEN PRINT "Tallene er indbyrdes primiske."
```

Følgende funktion er af tekst-type:

```
FUNC bagfra$(tekst$)
  længde:=LEN(tekst$)
  IF længde=1 THEN RETURN tekst$
  RETURN tekst$(længde)+bagfra$(tekst$(1:længde-1))
ENDFUNC bagfra$
```

kommandoen:

```
PRINT bagfra$("BIKSEMAD")
```

giver udskriften: 'DAMESKIB'.

Se PROC for nærmere forklaring om parametre.

GET$

er en indbygget funktion, hvis syntaks er:

```
GET$ (<udtrykl>,<udtryk2>)
```

hvor begge udtryk skal være positive hele tal.<udtrykl> skal angive et strømnummer, hvorfra der skal hentes en række bytes. Dette antal er angivet af værdien af <udtryk2>.

Eksempel. Udtrykket:

```
GET$(2,5)
```

giver en tekst på 5 tegn hentet fra den fil, der er åbnet med strømnummeret 2.

Tastaturet åbnes med filnavnet "KB:" (KeyBoard).

Eksempler:

```
OPEN FILE 3,"adresser",READ
PRINT GET$(3,20)
```

Fra den sekventielle fil "adresser" hentes 20 tegn. Den deraf sammensatte tekst udskrives derpå.

```
OPEN FILE 5, "kb:",READ
kommando$:=GET$(5,3)
PRINT kommando$
```

Når der er indtastet 3 tegn, tildeles denne tekst til kommando$.

Man kan også bruge GET$ i RANDOM filer:

```
OPEN FILE 0,"kartotek",RANDOM %_
SEEK 0,postnummer
PRINT GET$(0,10)
```

udskriver en tekst, bestående af de første 10 tegn i posten

'postnummer' i filen "kartotek", med direkte tilgang.

GOTO

sætningen/kommandoen har syntaksen:

GOTO <label>

hvor <label> er defineret et andet sted i programmet med en LABEL sætning (se LABEL).
Sætningen/kommandoen bevirker, at udførelsen fortsætter med linien efter denne LABEL sætning.

Bemærk: GOTO kan IKKE bruges i følgende tilfælde:

Ind/ud af en PROC-ENDPROC.

Ind/ud af en FUNC-ENDFUNC.

Ind/ud af en TRAP-ENDTRAP.

Ind i en FOR-ENDFOR.

Ind i en LOOP-ENDLOOP.

Se også LABEL.

HANDLER

er en sætning med syntaksen:

HANDLER

Sætningen bruges til at indlede en fejlhåndteringsrutine i COMAL-80/Z80.

Se i øvrigt appendix om fejlhåndtering i COMAL-80/Z80.

HEX$

er en indbygget funktion. Dens syntaks er:

HEX$(<numerisk udtryk>)

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Funktionen giver en tekst, der er 4 tegn lang og indeholder en hexamal repræsentation af <numerisk udtryk>.

HL

kan bruges på to måder. Som sætning/kommando med følgende syntaks:

HL <numerisk udtryk>

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Værdien af <numerisk udtryk> fortolkes som to bytes, af hvilke den høje byte tildeles Z80'erens H register, og den lave byte tildeles L registret.

Man kan også benytte HL som funktion. Den giver da indholdet af registerparret HL.

Eksempel. Når denne sætning/kommando:

HL USER

udføres, kommer HL til at pege på adressen angivet af USER.

Se også appendix om maskinkode i COMAL-80/Z80.

IF, ELIF, ELSE, ENDIF

Syntaksen for en IF sætning/kommando er:

IF <numerisk udtryk> THEN <simpel sætning>

eller kun som sætning:

IF <numerisk udtryk> THEN

Den første form er den simpleste form, en IF sætning kan have. Den virker på følgende måde:

Hvis <numerisk udtryk> er TRUE, (forskellig fra 0) så bliver <simpel sætning> udført. Hvis udtrykket derimod er FALSE, (lig 0) bliver <simpel sætning> IKKE udført.

Den anden form indleder en IF-struktur. En sådan struktur skal afsluttes med en ENDIF sætning. ENDIF sætningen har syntaksen:

```
ENDIF
```

En IF struktur kan se således ud:

```
IF <numerisk udtryk> THEN
  <programafsnit>
ENDIF
```

Denne struktur virker på samme måde som IF sætningen i en enkelt linie: <programafsnit> bliver kun indført, hvis <numerisk udtryk> udregnes til en værdi, der ikke er FALSE.

IF strukturen kan udvides med en ELSE sætning, der har følgende syntaks:

```
ELSE
```

Sammensat ser strukturen nu sådan ud:

```
IF <numerisk udtryk> THEN
  <programafsnit 1>
ELSE
  <programafsnit 2>
ENDIF
```

Denne struktur virker på følgende måde:
Hvis <numersik udtryk> er TRUE, bliver <programafsnit 1>, hvis derimod <numerisk udtryk> er FALSE bliver <programafsnit 2> udført.

Endnu en udvidelse kan laves i IF strukturen:

```
        ELIF <numerisk udtryk> THEN
```

Den helt store IF struktur ser sådan ud:

```
        IF <numerisk udtryk 1> THEN
          <programafsnit 1>
        ELIF <numerisk udtryk 2> THEN
          <programafsnit 2>
        ELIF <numerisk udtryk 3> THEN
          <programafsnit 3>
        ....
        ELIF <numerisk udtryk n> THEN
          <programafsnit n>
        ELSE <programafsnit n+1>
        ENDIF
```

ELSE delen kan evt. udelades. Denne struktur virker på følgende måde:

Hvis <numerisk udtryk1> udregnes til TRUE, udføres kun det ene <programafsnit 1>, og herefter fortsættes der efter ENDIF. Hvis <numersk udtryk1> er FALSE udregnes <nummerisk udtryk2>, og hvis det er TRUE udføres <programafsnit2>. Sådan fortsætter udregningen af alle numeriske udtryk, indtil:

Systemet finder et udtryk, der er TRUE, i så fald udføres det afsnit, der kommer umiddelbart efter den pågældende ELIF.

eller: Systemet møder en ELSE sætning, i så fald udføres <programfsnit n+1>.

eller: Systemet møder en ENDIF sætning, i så fald udføres ingen afsnit.

IMPORT

er en sætning. Syntaksen for denne sætning er:

```
IMPORT <navn>,<navn>,...
```

Hvor IMPORT sætningen SKAL stå inde i en lukket procedure, og de navne, der angives på listen må ikke være kendt inde i proceduren, men de skal være kendt udenfor proceduren.

Sætningen bruges til at gøre variable, procedurer, funktioner eller labels udenfor proceduren, tilgængelig inde i proceduren.

Eksempel:

```
PROC udskriv(tekst$9 CLOSED
  IMPORT ryd'skærm,ryd'linie
  ...
ENDPROC udskriv
```

Her er ryd'skærm og ryd'linie to procdurer, som er defineret udenfor den lukkede procedure 'udskriv' men ønskes brugt inde i denne.

```
PROC quicksort(l,r) CLOSED
  IMPORT a ()
  i=l: j:=r: x:=a((l+r) DIV 2)
  REPEAT
    WHILE a(i)<x DO i:+1
    WHILE x<a(j) DO j:-1
    IF i<=j THEN w:=a(i): a(i):=a(j): a(j):=w: i:+1: j:-1
  UNTIL i>j
  IF l<j THEN quicksort(l,j)
  IF i<r THEN quicksort(i,r)
ENDPROC quicksort
```

Her er et eksempel på en lukket procedure, der importerer en tabel, med een dimension, og proceduren sorterer elementerne i

a() i nummerorden.

IN

er en tekstoperator. Dens syntaks er:

<tekstl> IN <tekst2>

Operatoren giver værdien FALSE, hvis <tekstl> ikke findes som en sammenhængende deltekst i <tekst2>.

Hvis <tekstl> findes i <tekst2> giver operatoren en værdi, der svarer til positionen af <tekstl> i <tekst2>.

Hvis <tekstl> er den tomme tekst bliver resultatet længden af <tekst2> plus en.

Eksempel. Udtrykket:

"lot" IN "Charlotte"

giver resultatet 5.

INP

er en funktion. Dens syntaks er:

INP(<numerisk udtryk>)

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Funktionen henter en værdi fra en z80 I/O port. Dette gøres på den måde at BC bliver sat lig <numerisk udtryk> og herefter benyttes instruktionen:

IN A, (C)

til at finde resultatet.

INPUT

er en sætning/kommando. Den simpleste syntaks er:

INPUT <variabelliste>

Syntaksen kan også være:

INPUT <vink>:<variabelliste>

hvor <vink> er en tekstkomstant.

Sætningen/kommandoen kan også have syntaksen:

INPUT AT <udtryk1>,<udtryk2>:<som ovenfor>

hvor <udtryk1> skal være et helt tal, der angiver en linie på skærmen, og <udtryk2> skal være et helt tal, der angiver en positionen på den pågældende linie nr. <udtryk1> og på position nr. <udtryk2> på denne linie.

Sætningen/kommandoen kan yderligere udvides til:

INPUT AT <udtryk1>,<udtryk2>,<udtryk3>:<som overfor>

Hvor <udtryk3> er et helt positivt tal, den sidste talværdi definere et indtastningsfelt, som har en længde på <udtryk3> tegn. Skrivning ud over dette antal tegn er spærret.

Afsluttes sætningen/kommandoen med et semikolon eller et komma, bevirker det, at markøren bliver stående på linien efter indlæsningen.

Et semikolon vil (som i PRINT) bevirke udskrivning af et mellemrum.

INPUT FILE

er en sætning/kommando, der har syntaksen:

INPUT FILE <numerisk udtryk>:<som ved INPUT>

hvor <numerisk udtryk> angiver et strømnummer. Sætningen er faktisk kun en udvidelse af INPUT sætningen/kommandoen. Den eneste forskel er:

INPUT FILE henter sine data fra filen med det angivne strømnummer

Eksempel. Disse to sætninger:

```
OPEN FILE 7,"KB:",READ

INPUT FILE 7:AT 10,10:a$
```

virker på præcis sammen måde som sætningen:

```
INPUT AT 10,10:A$
```

Hvis man gerne vil hente data ind fra en fil og ikke ønsker disse skrevet ud på skærmen, kan man bruge:

```
DS -

INPUT FILE 7:variabel

DS+
```

(se DS) den værdi, der kommer ind fra filen vil i dette tilfælde ikke blive skrevet ud på skærmen.

Eksempel. Sætningerne:

```
OPEN FILE 4,"ascii",READ

DS-

INPUT FILE 4:a$,b$,c$

DS+

PRINT a$,b$,c$
```

bruges til at læse en sekventiel ascii fil:

Filen 'ascii' skal forekomme på disketten og den skal være oprettet og skrevet med PRINT FILE sætningen/kommandoen (se denne).

Hvis man ønsker at læse en fil med direkte tilgang (RANDOM), gøres dette ved at bruge SEEK før input file, eller at addere et postnummer til efter strømnummeret.

Eksempel. Sætningen:

```
INPUT FILE 4,52:linie$
```

gør det samme som sætningerne:

```
        SEEK 4,52
        INPUT FILE 4:linie$
```

Se iøvrigt DS og PRINT FILE.

INT

er en standardfunktion. Syntaksen er:

```
        INT (<numerisk udtryk>)
```

Funktionen giver heldelen af værdien af <numerisk udtryk>, dvs. det største hele tal, der er mindre end eller lig med udtrykkets værdi.

KEY$

er en tekstfunktion. Syntaksen er:

```
        KEY$
```

Funktionen giver den sidst trykkede tast. Hvis der ikke har været trykket nogen tast giver KEY$ den tomme tekst (EMPTY$) som værdi.

Eksempel. Følgende procedure:

```
        PROC vent'tast
          WHILE KEY$=EMPTY$ DO NULL
        ENDPROC vent'tast
```

venter på at du trykker en tast.

LABEL

er en sætning. Syntaksen er:

```
        LABEL <navn>:
```

Nøgleordet LABEL kan udelades. Man får da syntaksen:

```
<navn>:
```

En LABEL kan bruges af enten GOTO eller RESTORE sætningen.

Eksempel.

```
LOOP
  INPUT navn$
  IF navn$=EMPTY$ THEN GOTO halt
  ...
  ...
ENDLOOP
halt:
...
```

Nøgleordet LABEL, er normalt usynligt (som EXEC og LET), men hvis man ønsker, at det skal udskrives ved udlistning, skal man skrive:

```
SETLABEL+
```

Ved RESTORE sætningen har labels en anden betydning. Dette belyses lettest med et eksempel. I et program findes følgende sætninger:

```
telefonnumre:

DATA 673529, 722574, 722525
```

Hvis sætningen/kommandoen:

```
RESTORE telefonnumre
```

Udføres, sættes datapilen til at pege på det er det første element i køen efter sætningen:

telefonnumre

uanset hvilke data-sætninger der måtte findes tidligere i programmet.

I dette tilfælde peger pilen altså på 673529.

Bemærk: DATA sætninger, der står inde i en lukket procedure er lokale, for den pågældende procedure.

Se også GOTO, RESTORE og SETLABEL.

LEN

er en standardfunktion. Dens syntaks er:

LEN(<tekst udtryk>)

Funktionen giver længden af <tekst udtryk>.

LET

I COMAL-80/Z80 udtrykkes en tildeling ved hjælp af tegnet:

:=

For at lette programmeringen er det dog muligt at bruge et almindeligt lighedstegn under programindtastning. Dette ændrer systemet automatisk til ':='. Nøgleordet LET bliver ignoreret af systemet, hvis det bliver indtastet. Hvis man ønsker nøgleordet udskrevet under udlistning, kan man bruge:

SETLET+

Det er muligt at have flere tildelinger på en linie. I så fald skal de adskilles med et semikonol ';'.

Foruden almindelige tildelinger ':=' findes der også mulighed for direkte at addere til eller subtrahere fra en variabel. Dette angives med tegnene ':+' og ':-'.

Eksempler. Sætningen:

```
variabel:=6
```

bevirker at variabel sættes til 6. Sætningern:

```
tekst$:+"test"
```

bevirker at teksten 'test' lægges til indholdet af tekst$, og:

```
counter:-1
```

bevirker at der trækkes 1 fra variablen counter.

LINK

er en kommando. Syntaksen for LINK er:

```
LINK "<filnavn>"
```

Kommandoen bruges til at hente en maskinkodepakke ind fra disketten.

Eksempel. Kommandoen:

```
LINK "OVERFRÆS.PCK"
```

henter maskinkodepakken i filen "OVERFRÆS.PCK" ind i datamatens lager.

Se nærmere i appendix om pakker i COMAL-80/Z80.

LIST

er en kommando som bruges, når man vil udskrive hele eller en del af det program, der i øjeblikket befinder sig i maskinens lager.

Kommandoen:

```
LIST
```

får systemet til at skrive en liste over hele programmet. Kommandoen:

```
LIST 100
```

udskriver linie 100. Kommandoen:

```
LIST 100-200
```

bevirker en udskrift af linierne mellem 100 og 200 incl. Kommandoen:

```
LIST -300
```

udskriver alle linier til og med linie 300, og kommandoen:

```
LIST 300-
```

udlister alle linier fra og med linie 300.

LIST kan også efterfølges af et navn på en procedure. F.eks. vil:

```
LIST udskrift
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

bevirker, at en liste over proceduren med navnet udskrift (kun denne!) bliver vist på skærmen.

Normalt vil en programliste udskrives med en skærmside af gangen. Når en side er udskrevet, venter systemet på at der trykkes en tast. Man kan trykke på en af følgende taster:

<RETURN>:

bevirker at systemet udlister endnu en linie på skærmen.

<SPACE>:

bevirker at systemet udlister en ny side på skærmen.

<ESC>:

bevirker at systemet går ud af list-kommandoen, og giver mulighed for at editere på skærmen.

Kommandoen kan også bruges til at gemme programmer som ASCII-filer på disketten. Kommandoen:

```
LIST "mitprog"
```

gemmer det program, der for øjeblikket befinder sig i maskinens lager, som en programfil, med navnet "mitprog". Programmet gemmes som kildetekst, og kan derfor flettes sammen med et andet i maskinens lager (se MERGE). Da LIST kommandoen arbejder direkte på kildeteksten, er denne form også tilladt:

```
LIST "ditprog" 100-200
```

I dette tilfælde gemmes linierne 100-200 i programfilen "ditprog".

LiST kommandoen kan også bruges til at gemme procedurer, kommandoen:

```
LIST "udskrift.l" udskrift
```

gemmer en liste over proceduren udskrift i filen "udskrift.l.".

En programliste, der er gemt med LIST eller DISPLAY på disketten, kan læses med kommandoerne:

```
ENTER
MERGE
```

eller sætningen:

```
INPUT FILE
```

Se yderligere DISPLAY, ENTER, MERGE og INPUT FILE.

LN

er en indbygget funktion. Syntaksen er:

```
LN(<numerisk udtryk>)
```

hvor <numerisk udtryk> er et positivt tal, forskellig fra 0. Funktionen giver den naturlige logaritme til <numerisk udtryk>.

Den naturlige logaritme har grundtallet:

```
EXP(1) = 2.7182818279
```

Se også LOG og LOGBASE.

LOAD

er en kommando. Dens syntaks er:

```
LOAD "<filnavn>"
```

Kommandoen bruges til at hente programmer fra disketten ind i maskinens lager.

Eksempel. Kommandoen:

```
LOAD "monitor"
```

bevirker at programmet i filen "monitor" hentes ind i maskinens lager. LOAD kommandoen kan kun bruges i forbindelse med programmer, som er gemt på disketten ved brug af SAVE kommandoen (se SAVE).

Hvis et COMAL-80/Z80 program anvender maskinkodepakker, og programmet gemmes med SAVE kommandoen, bliver maskinkoden gemt på disketten sammen med COMAL programmet. Systemet opfatter altså i den henseende maskinkoden som en del af COMAL programmet. Hvis programmet senere hentes med LOAD kommandoen følger maskinkodepakken med ind i lageret.

LOG

er en funktion. Dens syntaks er:

```
LOG (<numerisk udtryk>)
```

hvor <numerisk udtryk> er et positivt tal, forskellig fra 0.

Funktionen er en logaritmefunktion, hvis grundtal brugeren selv kan definere. Ved opstart af COMAL-80/Z80, sættes LOG til den naturlige logaritme (som LN), men ved brug af LOGBASE, kan funktionens grundtal let ændres.

Se også LN og LOGBASE.

LOGBASE

kan bruges på to måder. Som sætning/kommando med følgende syn-

taks:

LOGBASE <numerisk udtryk>

Hvor <numerisk udtryk> er et positivt tal, forskellig fra 0.

I denne form bruges den til at definere grundtallet for LOG-funktionen med.

Eksempel. Sætningen:

LOGBASE 10

sætter LOG-funktionen til at være 10-talslogaritmen.

LOGBASE kan også bruges som funktion. Syntaksen er:

LOGBASE

Funktionen giver det nuværende grundtal som resultat.

Eksempel. Sætningen/kommandoen:

PRINT LOGBASE

vil (hvis ikke andet er defineret) give udskriften: 2.71828279

Se også LOG og LN.

LOOP, TIMES, ENDLOOP

Den simpleste form af LOOP sætningen se således ud:

LOOP <simpel sætning>

Denne form af LOOP er i princippet en uendelig løkke, men kan dog stoppes ved at trykke på <ESC> tasten. Eller hvis systemet møder

en STOP-sætning.

Eksempel. Programmet:

```
LOOP udfør
//

PROC udfør
  ...
ENDPROC udfør
```

vil kalde proceduren 'udfør' i en uendelig løkke.

En lille udvidelse af LOOP sætningen vil være:

```
LOOP <numerisk udtryk> TIMES <simpel sætning>
```

Hvor TIMES kan indskrives som et kolon (:).

Denne sætning/kommando vil udføre <simpel sætning> et antal gange, der er nærmere angivet ved værdien af <numerisk udtryk>.

LOOP kan også benyttes til at indlede en LOOP-struktur:

```
LOOP
  <programafsnit>
ENDLOOP
```

eller:

```
LOOP <numerisk udtryk> TIMES
  <programafsnit>
ENDLOOP
```

Her vil <programafsnit> blive udført i stedet for <simpel sætning>.

Med en af følgende sætninger kan man hoppe ud af en LOOP-struktur:

En GOTO sætning, til en LABEL udenfor LOOP-ENDLOOP

En EXIT/EXIT WHEN sætning

Se også EXIT, EXIT WHEN.

LOWER$

er en tekstfunktion. Dens syntaks er:

LOWER$(<tekst udtryk>)

Funktionen giver <tekst udtryk>, men alle store bogstaver er blevet lavet om til små bogstaver.

Eksempel. Efter sætningen:

PRINT LOWER$("Charlotte JENSEN")

vil systemet svarer med: 'charlotte jensen'.

MAXMEM

er en funktion. Syntaks:

MAXMEM

Funktionen giver en adresse på systemvariablene i COMAL-80/Z80. Samtidigt angiver MAXMEM den maksimale adresse for data, der er lagret i USER området.

Se også USER og NEW.

MERGE

er en kommando. Syntaksen er:

```
MERGE "<filnavn>"
```

Kommandoen bevirker at et programafsnit, som er gemt på disketten med LIST eller DISPLAY under navnet <filnavn>, flettes sammen med det program, som ligger i lageret.

Dette belyses lettest ved hjælp af et eksempel. Lad os tænke os, at programmet, som ligger i lageret, har 210 som højeste linienummer. Man ønsker at føje et afsnit til og dette afsnit ligger på disketten under navnet "udskrift.l". Man kan da bruge kommandoen:

```
MERGE "udskrift.l"
```

Det nævnte programafsnit bliver nu hentet ind fra disketten og får automatisk linienumre, der starter med 220 og videre med trin på 10.

Kommandoen kan også have denne form:

```
MERGE 5000 "udskrift"
```

I dette tilfælde får det programafsnit, som hentes ind fra disketten, linienumrene: 5000, 5010, 5020, osv. Bruger man kommandoen:

```
MERGE 4000,2 "udskrift"
```

for afsnittet linienumrene: 4000, 4002, 4004, osv.

MOD

er en operator med syntaksen:

```
<udtryk1> MOD <udtryk2>
```

Operatoren giver resten ved division af <udtryk1> med <udtryk2>.

Se også DIV og appendix om udtryk i COMAL-80/Z80.

MOUNT

er en sætning/kommando. Dens syntaks er:

```
MOUNT
```

eller:

```
MOUNT "<unit>"
```

Sætningen/kommandoen anvendes til at initialisere en diskette.

Den første form af MOUNT initialiserer disketten i drev 0 og sætter UNIT til dette drev. Den anden form gør det muligt at initialisere en diskette i et andet drev.

Eksempel. Sætningen/kommandoen:

```
MOUNT "DK:"
```

Initialiserer disketten i drev 1, og sætter UNIT til dette drev.

Se iøvrigt UNIT.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

NEW

er en kommando med følgende syntaks:

NEW

NEW <udtrykl>

eller:

NEW <udtrykl>,<udtryk2>

Hvor <udtrykl> og <udtryk2> er hele positive tal i det lukkede interval fra 0 til 65535. Den første form af NEW lukker alle åbne filer, fjerner alle pakker og sletter det nuværende program.
<udtrykl> angiver et antal bytes, som brugeren kan få reserveret til maskinkode eller andet data. Når programmet slettes vil der blive gjort plads til dette antal bytes et sted i hukommelsen. Adressen på dette sted kan hentes med funktionen:

USER

<udtryk2> angiver længden på Z80 maskinstakken. Denne værdi skal være større end 512. Maskinstakken er fra opstart sat til en længde af 1024 bytes.

Bemærk: Efter at have reserveret noget af hukommelsen på denne måde, vil den simple form:

NEW

ikke fjerne de reserverede bytes eller ændre på maskinstakkens længde.

Se også USER og appendix om maskinkode i COMAL-80/Z80.

NEXT

er et nøgleord, der kan indskrives i programmet i stedet for ENDFOR, systemet vil automatisk ændre det til ENDFOR.

Se ENDFOR.

NOT

er en logisk operator, som udtrykker "logisk ikke".

Se iøvrigt appendix om udtryk i COMAL-80/Z80

NULL

er en sætning/kommando. Syntaks:

```
NULL
```

Sætningen/kommandoen udfører den tomme process, dvs absolut ingenting.

Ligesom den tomme mængde i matematikken kan den dog være såre nyttig af syntaktiske grunde.

OF

er et nøgleord, som bruges i COMAL-80/Z80 til at afslutte en CASE sætning eller under erklæring af en tekstvariabel eller en tekst-tabel.

Se nærmere under CASE og DIM.

OPEN

er en sætning/kommando. Syntaksen for OPEN er:

```
OPEN FILE <udtryk>,<tekst>,<retning>
```

hvor <udtryk> er strømnummeret, <tekst> er filnavnet og <retning> er en af følgende muligheder:

```
READ
WRITE
APPEND
RANDOM <numerisk udtryk>
```

Sætningen/kommandoen bruges til at åbne data filer på en ydre enhed.

Eksempel. Sætningen/kommandoen:

```
OPEN FILE 2,"karakter",WRITE
```

åbner filen "karakter". Strømnummeret 2 bruges i resten af programmet til at referere til denne fil, og nøgleordet WRITE betyder, at filen åbnes for gemning af data.

Hvis man ønsker at åbne en fil, med direkte tilgang (RANDOM fil) skal denne åbnes med <retning> RANDOM.

Eksempel:

```
OPEN FILE 0,"kunder",RANDOM 100
```

Konstanten 100 efter ordet RANDOM angiver, at hver post må være på op til 100 bytes. Brugeren behøver ikke skrive ordene FILE og RANDOM, dem sætter systemet, hvis de mangler.

OR

er en logisk operator, som udtrykker "logisk eller".

Se også appendix om udtryk i COMAL-80/Z80.

ORD

er en standardfunktion. Dens syntaks er:

```
ORD (<tekst udtryk>)
```

Funktionen giver ascii-værdien af det første tegn i <tekst udtryk>.

OFF

er et nøgleord, der bruges i forbindelse med TRACE.

Se nærmere under TRACE.

OTHERWISE

er en sætning, der indleder en alternativ del i en CASE-struktur.

Se under CASE.

OUT

er en sætning/kommando. Syntaksen er:

```
OUT <udtrykl>,<udtryk2>
```

hvor <udtrykl> er et helt tal i det lukkede interval fra 0 til 65535, og <udtryk2> er et helt tal i det lukkede interval fra 0 til 255.

Sætningen/kommandoen sender værdien af <udtryk2> til den port, som værdien af <udtrykl> angiver. Internt gøres det på den måde, at værdien af <udtrykl> placeres i BC registret og A registeret sættes lig værdien af <udtryk2> hvorefter Z80 instruktionen:

```
OUT (C),A
```

udføres.

OUTPUT

er et nøgleord, der bruges i SELECT sætningen. Brugeren behøver ikke indtaste dette nøgleord, da systemet selv vil sætte det, hvis det mangler.

Se også under SELECT.

PAGE

er en sætning/kommando med følgende syntaks:

```
PAGE
```

Sætningen/kommandoen bevirker, at skærmen renses. Hvis printeren eller andet medie er valgt som uddatamedie, sender COMAL-80/Z80 blot et signal om sideskift (form feed = 12) til dette.

PEEK

er en standardfunktion. Syntaks:

PEEK(<numerisk udtryk>)

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Funktionen henter værdien af den byte i maskinens lager, der angives af <numerisk udtryk>.

PI

er en konstant.

Værdien for PI er: 3.1415927464.

POKE

er en sætning/kommando. Dens syntaks er:

POKE <udtryk1>,<udtryk2>

hvor <udtryk1> er et helt tal i det lukkede interval fra 0 til 65535, og <udtryk2> er et helt tal i det lukkede interval fra 0 til 255. Sætningen/kommandoen lagrer værdien af <udtryk2> i den adresse, som værdien af <udtryk1> angiver.

PRINT

sætningen/kommandoen er opbygget således:

PRINT <udskriftliste>

Hvor <udskriftliste> består af en liste med udtryk. De enkelte elementer på listen skal adskilles med komma eller semikolon. Listen kan også afsluttes med et komma eller et semikolon.

Eksempel. I sætningen:

```
      Print "Arealet er:"; højde*grundlinie/2
```

består udskriftlisten af et tekstudtryk og et numerisk udtryk.

Hvis skilletegnet semikolon (;) anvendes, skriver systemet altid eet mellemrum mellem hvert element. Et komma (,) har derimod normalt ikke nogen som helst virkning på udskriften, men tjener blot til at adskille de enkelte elementer på listen. Man kan dog definere en såkaldt skrivezone ved hjælp af system-variablen ZONE>

Betydningen og brugen af ZONE belyses i følgende eksempler:

```
        ZONE 10
        PRINT 1, 2, 3
```

Giver udskriften:

```
1       2       3
12345678901234567890123...   <-- kolonne nummer.
```

hvorimod disse to linier:

```
        ZONE 5
        PRINT 1, 2, 3
```

giver udskriften:

```
1       2       3
```

Efter opstart har ZONE værdien 0, hvilket betyder, at hvis ingen andre værdier af ZONE er blevet specificeret, giver PRINT sætningen fra de to forrige eksempler følgende udskrift:

```
12345678901234...            <-- kolonne nummer
123
```

Semikolon giver altid et ekstra mellemrum. For eksempel:

```
PRINT 1;2;3
```

udskriver:

```
1 2 3
```

Normalt udføres et linieskift, når PRINT sætningen er udført, men hvis en udskriftsliste afsluttes med komma eller semikolon, erstattes dette linieskift med hhv, et zone-tab eller et mellemrum.

Hvis man ønsker at springe frem på linien, til en bestemt kolonne, kan man bruge TAB funktionen:

```
PRINT 1,TAB 6,2
```

vil udskrive:

```
1     2
```

forudsat at ZONE er lig 0.

Hvis ikke ZONE er 0, vil dette have effekt på kommaet FØR TAB funktionen, men ikke på kommaet efter TAB værdien.

Hvis man har anført en TAB funktion i en linie, og markøren er forbi den position der ønskes, vil markøren blive positioneret på den pågældende kolonne i den næste linie.

En anden funktion, som man kan benytte i en PRINT sætning er AT funktionen:

```
PRINT AT 3,3:"Dette er linie 3",AT 10,3:"Dette er linie 10"
```

Når fortolkeren støder på en AT funktion, læses de to parametre, på samme måde som ved CURSOR sætningen/kommandoen (se denne), og markøren placeres på dette sted, før fortolkeren fortsætter udførelsen af linien.

Endelig kan man til sidst have en USING del i PRINT sætningen:

```
PRINT USING <format>:<udskriftliste>
```

hvor <format> er et tekstudtryk, mens <udskriftliste> er den samme som i den simple PRINT sætning.

Dette lille program skulle belyse brugen af USING:

```
0010 format$:="-#####,##"
0020 REPEAT
0030   INPUT x
0040   PRINT "Udformatteret:";x;TAB 30
0050   PRINT USING "Formatteret:"+format$:x
0060 UNTIL x=0
```

```
Uformatteret: 3                 Formatteret:       3.00
Uformatteret: -3                Formatteret:      -3.00
Uformatteret: -0.3              Formatteret:      -0.30
Uformatteret: 0.3               Formatteret:       0.30
Uformatteret: 234.4567          Formatteret:     234.46
Uformatteret: -234.5678         Formatteret:    -234.57
Uformatteret: 12345678          Formatteret: *********
Uformatteret: -0.123456         Formatteret:      -0.12
Uformatteret: 3.4E-04           Formatteret:       0.00
Uformatteret: 0                 Formatteret:       0.00
```

Der kan kun forekomme een USING-del i en PRINT sætning/kommando.

Se også ZONE og CURSOR.

PRINT FILE

er en sætning/kommando, som bruges til at gemme data i sekventielle eller direkte filer. For eksempel vil:

```
PRINT FILE 2:navn$, telefonnummer
```

gemme værdierne af de variable på listen sekventielt i filen med strømnummeret 2. (Se også OPEN) sætningen:

```
PRINT FILE 4,i:kunde$,regning$
```

gør det samme som:

```
SEEK 4,i
PRINT FILE 4:kunde$,regning$
```

altså, den skriver variablene på udskriftlisten ud i post nummer i, i den direkte fil, der er åbnet med strømnummeret 4.

Data, der er gemt med PRINT FILE skal hentes med INPUT FILE.

Se også INPUT FILE.

PROC

Hvis en blok af sætninger indledes med sætningen:

```
PROC <navn>
```

hvor <navn> er et gyldigt navn, og afsluttes med sætningen:

```
ENDPROC <navn>
```

kan denne blot kaldes som et underprogram ved hjælp af sætningen:

```
<navn>
```

Navnet efter ENDPROC behøver brugeren ikke selv at indtaste, systemet tilføjer det selv, hvis det mangler.

Efter at underprogrammet er udført, fortsættes programudførelsen med linien efter den sætning, som kaldte underprogrammet (proceduren).

Programteksten imellem PROC og ENDPROC bliver indrykket under udlistningen af programmet.

Eksempel:

```
PROC udskriv
```

```
    PRINT USING "##.##":kroner
  ENDPROC udskriv
  //
  REPEAT
    READ kroner
    udskriv
  UNTIL EOD
  DATA 1,2,3,45
```

PROCEDURER MED PARAMETRE.

I COMAL-80/Z80 kan PROC sætningen udvides til:

```
  PROC <navn> ( <liste over formelle parametre> )
```

og nu er der tale om et regulært procedurehoved. En procedure med parametre skal kaldes således:

<navn> (<liste over aktuelle parametre>)

Følgende eksempler bruger parametre til procedurer.

Eksempel nr. 1:

En procedure er indledt med sætningen:

```
  PROC prøv(i,j)
```

og bliver kaldt af sætningen:

```
  prøv(først,sidst)
```

I dette tilfælde er 'i' og 'j' i procedurehovedet formelle parametre, og disse skal tildeles en værdi, når proceduren bliver kaldt. Navnene 'først' og 'sidst' i kaldet er de aktuelle parametre, og de skal være tildelt en værdi, inden kaldet bliver udført. Under procedure kaldet, får 'i' tildelt værdien af 'først' og 'j' tildeles værdien af 'sidst'.

Man kan sige, at 'i' og 'j' overtager værdierne fra 'først' og 'sidst'. Derfor kalder man også 'i' og 'j' for værdiparametre (calledby-value).

'i' og 'j' bliver behandlet som lokale variable for proceduren 'prøv' og de vil således ikke være kendt for verden udenfor proceduren og vil ikke blive forvekslet med eventuelle andre

variable 'i' og 'j' et andet sted i programmet.

De aktuelle parametre til en værdiparameter kan være numerisk udtryk.

Proceduren 'prøv' kan således også kaldes med sætningen:

```
prøv(1,8)
```

eller:

```
prøv(j+1,i+1)
```

Et kald til en procedure kan endog foretages inden i proceduren selv, hvorved den kalder sig selv rekursivt. I dette tilfælde opretter systemet nye pladser for 'i' og 'j', som om den aldrig havde mødt dem før, og den vil aldrig forveksle værdierne fra det ene kald til det næste.

Eksempel nr. 2:

En procedure er indledt med:

```
PROC skrivpost(r,n$,REF m())
```

og kaldt med:

```
skrivpost(elevnr,navn$,karakter())
```

I dette tilfælde er 'r' og 'n$' formelle værdiparametre og under kaldet overtager de værdien fra hhv. 'elevnr' og 'navn$'.

Tegnfølgen 'REF m()' betyder en formel parameter 'm', som er reference-parameter (called-by-reference). Tegnene '()' efter 'm' betyder, at 'm' henviser til en endimensionabel tabel. For referenceparametres vedkommende sker der ingen tildeling under kaldet, den formelle parameter bliver bare brugt som en slags "øgenavn" for den aktuelle parameter, 'karakter()' vil således lide for alt det, som 'skrivpost' udsætter 'm()' for. Alle ændringer i tabellen 'm()' referer altså tilbage til tabellens karakter.

Procedurehovedet kunne også have været:

```
PROC skrivpost(r,REF n$,REF m())
```

Den eneste forskel i forhold til det forrige procedure hoved er, at 'n$' nu også er en referenceparameter. 'n$' henviser kun til 'navn$', og der sker ingen tildeling. Det øger naturligvis hastigheden og sparer lagerplads.

Eksempel nr. 3:

En procedure er indledt med:

```
PROC udskriv(REF tabel (,))
```

Tegnene '(,)' efter parameternavnet 'tabel' angiver, at 'tabel' kun kan henvise til todimensionabel tabel. Tegnene '(,,)' viser, at der henvises til en tredimensionabel tabel osv.

Lukkede procedurer:

En procedure kan også være lukket. Dette angiver man ved at afslutte procedurehovedet med nøgleordet CLOSED. I dette tilfælde vil alle variable, der optræder i proceduren, blive behandlet som lokale variable på samme måde som værdiparametrene før. Arbejdslageret til de lokale variable bliver frigjort, når procedurekaldet er afsluttet.

Ønsker man at anvende variable, procedurer, funktioner eller labels, som findes i programmet udenfor den lukkede procedure, skal disse gøres tilgængelige for proceduren ved hjælp af en IMPORT sætning.

Se også under IMPORT.

PROTECT

er en kommando. Syntaksen er:

```
PROTECT
```

Kommandoen bruges til at LIST-beskytte et program, efter kommandoen:

```
PROTECT
```

er det meget svært at læse en liste af programmet i lageret.

SAVE og LOAD er dog stadig tilladt.

PUT

er en sætning, der er modstykket til GET$. Syntaksen er:

PUT FILE <numerisk udtryk>:<tekst udtryk>

hvor <numerisk udtryk> er et strømnummer.

Sætningen/kommandoen bruges til at sende enkelte bytes til en fil.

Hvis man f.eks. ønsker at lave en maskinkodepakke kan denne gemmes på disketten med PUT.

Hvis man ser på sætningen:

```
PUT FILE 2:"Dette er en test"
```

og sætningen:

```
PRINT FILE 2:"Dette er en test"
```

virker de på næsten samme måde. PRINT sætningen sender blot en 13-byte og en 10-byte efter teksten, det gør PUT IKKE.

PUT kan også bruges i filer med direkte tilgang (RANDOM fil), også her er fremgangsmåden den samme som i PRINT FILE.

Se også PRINT FILE.

QUIT

er en kommando, der har syntaksen:

```
QUIT
```

Kommandoen bringer systemet ud af COMAL-80/Z80.

Før selve udhoppet fra COMAL-80/Z80 systemet, bliver alle filer lukket (CLOSE) og alle pakker smidt ud (DISCARD).

RAND

er et nøgleord, der kan skrives i programmet i stedet for RANDOMIZE, systemet vil automatisk ændre det til RANDOMIZE.

Se RANDOMIZE.

RANDOM

er et nøgleord, der bruges i OPEN sætningen til at åbne filer med direkte tilgang.

Se nærmere under OPEN.

RANDOMIZE

er en sætning/kommando, der har syntaksen:

RANDOMIZE

eller:

RANDOMIZE ,<numerisk udtryk>

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 0 til 65535. Sætningen bruges i forbindelse med tilfældige tal, frembragt af RND funktionen. Hvis man benytter den første form:

RANDOMIZE

bruges den øjeblikkelige værdi af TIMER (MOD 65535) som startgrundlag.

Hvis f.eks. der skrives:

RAND 5

bruges 5 som startgrundlag for rækken af tilfældige tal, frembringer systemet den samme række af tilfældige tal, hver gang RND udføres.

Dette kan være nyttigt til visse eksperimentielle formål.

READ

er en sætning/kommando med syntaksen:

READ<variabelliste>

hvor <variabel> består af en række variable eller tabeller.

Se også DATA.

READ FILE

er en sætning/kommando, som har syntaksen:

READ FILE <numerissk udtryk>:<variabelliste>

READ FILE bruges til at læse sekventielle og direkte filer, som er gemt på diskette. For eksempel vil sætningen:

READ FILE 2:navn$,adresse$,by$

tildele værdier til de variabler på listen ved at læse sekventiel i den fil, som har strømnummeret 2 (Se OPEN).

Sætningen:

READ FILE $,i:navn$,elevnr,afdnr

tildeler værdier til de variable på listen, ved at læse post nr. i i den direkte fil, som er åbnet med strømnummeret 4.

Se også OPEN.

REF

er et nøgleord, der bruges til at angive reference-parametre blandt de formelle parametre.

Se under PROC.

RENAME

er en sætning/kommando med syntaksen:

RENAME <tekstl>, <tekst2>

hvor <tekstl> og <tekst2> er filnavne. Sætningen/kommandoen omdøber filen med navnet <tekstl> til <tekst2>.

RENUM

er en kommando, der bruges til at give programlinierne i det nuværende program nye linienumre. Hvis kommandoen afgives uden yderligere specifikation ændres rækken af linienumre i et program til sekvensen:

```
0010, 0020, 0030, ....
```

Kommandoen:

```
RENUM 1000
```

ændrer linienumrene til:

```
1000, 1010, 1020, ....
```

Kommandoen:

```
RENUM 1000,2
```

ændrer linienumrene til:

```
1000, 1002, 1004, ...
```

Kommandoen:

```
RENUM ,2
```

ændrer linienumrene til:

```
0010, 0012, 0014, ...
```

Kommandoen:

```
RENUM 560;5000,2
```

ændrer lininumrene i afsnittet fra og med linie 0560 til følgende:

```
5000, 5002, 5004, ...
```

REPEAT, UNTIL

er to sætninger, som indgår ifølgende løkkestruktur:

```
REPEAT
  <programafsnit>
```

UNTIL <numerisk udtryk>

<programafsnit> bliver gentaget indtil udtrykket efter UNTIL antager værdien TRUE (forskellig fra 0). Når dette sker, fortsættes programudførelsen med sætningen efter UNTIL sætningen.

Programteksten mellem REPEAT og UNTIL bliver indrykket under udskrift af programmet.

REPORT

er en sætning, der har syntaksen:

REPORT

eller:

REPORT <numerisk udtryk>

Hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 1 til 75. Sætningen bruges i forbindelse med brugerdefineret fejlhåndtering.

Se også appendix om fejlhåndtering i COMAL-80.

RESTORE

er en sætning/kommando. Syntaksen er:

RESTORE

eller:

RESTORE <label>

Hvis den første form benyttes sættes data-pilen til at pege på den første data-sætning i programmet, dvs. det første element i data-køen.

Hvis den anden form benyttes sættes data-pilen til at pege på den første data-sætningen, der er efter <label> i programmet.

Bemærk: En data-kø i en lukket procedure, er lokal for denne. En RESTORE sætning inde i en lukket procedure virker altså kun på evt. køer, der er opstillet indenfor rammerne af den pågældende

procedure, med mindre, man har brugt IMPORT, til at importere en <label> udenfor proceduren, og bruger RESTORE på denne <label>.

Se også LABEL.

RETURN

er en sætning. Dens syntaks er:

```
RETURN
```

eller:

```
RETURN <udtryk>
```

hvor <udtryk> er et tekstudtryk eller et numerisk udtryk.

Den første form bruges, når man ønsker at forlade en procedure, inden ENDPROC sætningen er nået.

Den anden form benyttes når man skal returnere værdien fra en brugerdefineret funktion (se FUNC).

Eksempel. Proceduren:

```
PROC prøve
  LOOP
    INPUT x
    IF x=0 THEN RETURN
    udskriv(x)
  ENDLOOP
ENDPROC prøve
```

vil efter kald vente på forskellige værdier indtastet til x og komme tilbage fra proceduren når værdien 0 er blevet indtastet.

Funktionen:

```
FUNC fak(n)
  IF n>1 THEN RETURN n*fak(n-1)
  RETURN 1
ENDFUNC fak
```

kan kaldes ved at skrive:

```
SCAN
PRINT fak(8)
```

systemet vil herefter give udskriften: '40320'

Se også PROC og FUNC.

RND

er en standardfunktion. Dens syntaks er:

```
RND
```

eller:

```
RND (<udtryk1>,<udtryk2>)
```

Den første form giver som værdi et tilfældigt flydende tal i det åbne interval fra 0 til 1.

Den anden form giver et helt tal i det lukkede interval fra <udtryk1> til <udtryk2>.

Bemærk: Hvis <udtryk1> og <udtryk2> ikke er hele tal, giver RND funktionen nogle resultater, som kan henligge brugeren i dyb coma(1)!

Se også RANDOMIZE.

ROUND

er en indbygget funktion. Dens syntaks er:

ROUND (<numerisk udtryk>).

Funktionen giver en korrekt afrunding af <numerisk udtryk>.

RUN

er en kommando. Syntaksen for RUN er:

RUN

Kommandoen bevirker, at alle variable nulstilles, der udføres et prepass (se SCAN) af programmet, og systemet starter programmet i arbejdslageret med den første sætning.

Se også SCAN.

SAVE

er en kommando. Dens syntaks er:

SAVE "<filnavn>"

Kommandoen gemmer således det program, der er i arbejdslageret, under navnet <filnavn> på disketten. Dette program kan hentes igen ved brug af LOAD, CHAIN eller RUN kommandoen.

Se også LOAD, RUN og CHAIN.

Bemærk: Eventuelle pakker, der er LINK'et ind i hukommelsen, bliver også gemt på disketten. Brugeren behøver altså ikke at bruge LINK kommandoen til at hente eventuelle maskinkodepakker med.

SCAN

er en kommando. Syntaksen er:

```
SCAN
```

Kommandoen bevirker, at der udføres et prepass af det program, som ligger i arbejdslageret. Herved undersøges opbygningen af alle strukturer, og eventuelle fejl rapporteres. Det samme sker, når kommandoen RUN udføres, men i modsætning til denne, startes programmet ikke af SCAN.

Efter at SCAN er udført, kan enhver procedure og funktion i programmet kaldes direkte, dvs. et procedurekald og et funktionskald kan udføres som en kommando.

SEEK

er en sætning/kommando. Dens syntaks er:

```
SEEK <udtryk1>,<udtryk2>
```

hvor <udtryk1> er et strømnummer og <udtryk2> postnummeret indenfor filen.
Sætningen/kommandoen får således pilen fra filen med strømnummer <udtryk1> til at pege på post nr. <udtryk2>. Det er nu muligt at skrive eller læse sekventielt fra denne post.

En anden måde at lave søgning på, er at indbygge søgningen i READ eller WRITE sætningen:

```
WRITE FILE 1,45:x
```

erstatter således:

```
SEEK 1,45
WRITE FILE 1:x
```

Se også READ og WRITE.

SELECT

er en sætning/kommando. Dens syntaks er:

```
SELECT OUTPUT "<filnavn>"
```

Sætningen/kommandoen bevirker at de efterfølgende udskrifter fra systemet dirigeres til filen med navnet <filnavn>.

Udover et lovligt filnavn, kan man også angive følgende filnavne:

"lp:"

betyder Line Printer.

"ds:"

betyder Data Screen.

"cs:"

betyder Cassette.

Bemærk: Tekster der bruges som vink i en INPUT sætning, bliver altid udskrevet på skærmen. EDIT kommandoen udskriver også altid sine linier på skærmen. Det samme gælder for FIND- og CHANGE-kommandoerne. Ved TRACE skal man angive et specielt filnavn.

SETDEG

er en sætning/kommando, der har syntaksen:

```
SETDEG
```

Sætningen/kommandoen bevirker, at systemet fra nu af regner i grader. Dette påvirker følgende funktioner: ACS, ASN, ATN, COS, SIN og TAN.

SETEXEC

er en kommando. Dens syntaks er:

```
SETEXEC +
```

eller:

```
SETEXEC -
```

Den første form bevirker, at nøgleordet EXEC bliver skrevet ud ved udlistning af programmer.

Den anden form bevirker, at nøgleordet IKKE udskrives. I forbindelse med systemstart udskrives EXEC nøgleordet ikke.

Se også EXEC.

SETGRAD

sætningen/kommandoen har følgende syntaks:

```
SETGRAD
```

Sætningen/kommandoen gør, at fra nu af regner i nygrader. Dette påvirker følgende funktioner: ACS, ASN, ATN, COS, SIN og TAN.

SETLABEL

er en kommando. Dens syntaks er:

```
SETLABEL +
```

eller:

```
SETLABEL -
```

Den første form bevirker, at nøgleordet LABEL bliver skrevet ud ved udlistning af programmet.

Den anden form bevirker, at nøgleordet IKKE udskrives. I forbindelse med systemstart udskrives LABEL nøgleordet ikke.

Se også LABEL.

SETLET

er en kommando. Dens syntaks er:

```
SETLET +
```

eller:

SETLET -

Den første form bevirker, at nøgleordet LET bliver skrevet ud ved udlistning af programmet.

Den anden form bevirker, at nøgleordet IKKE udskrives. I forbindelse med systemstart udskrives LET nøgleordet ikke.

Se også LET.

SETRAD

sætningen/kommandoen har følgende syntaks:

SETRAD

Sætningen/kommandoen gør, at fra nu af regner i radianer. Dette påvirker følgende funktioner: ACS, ASN, ATN, COS, SIN og TAN.

SGN

er en standardfunktion. Den syntaks er:

SGN (<numerisk udtryk>)

Funktionen giver fortegnet af værdien af det numeriske udtryk; hvis udtrykket er mindre end 0 er resultatet -1, hvis udtrykket er lig med 0 er resultatet 0, og hvis udtrykket er større end 0 er resultater 1.

SIN

er en indbygget funktion. Dens syntaks er:

SIN (<numerisk udtryk>)

Funktionen giver sinus af det numeriske udtryk. Ved hjælp af sætningerne/kommandoerne SETDEG, SETGRAD og SETRAD kan brugeren bestemme, hvorvidt værdien af <numerisk udtryk> angiver hhv. grader, nygrader eller radianer.

Se også SETDEG, SETGRAD og SETRAD.

SIZE

er en kommando med syntaksen:

```
SIZE
```

Kommandoen får systemet til at give en beskrivelse over lageret, hvor mange bytes programmet fylder, hvor mange bytes data optager, og hvor mange frie bytes der er. Antallet af frie bytes kan også hentes med FREE funktionen.

Se også FREE.

SPC$

er en indbygget funktion. Dens syntaks er:

```
SPC$(<numerisk udtryk>)
```

hvor <numerisk udtryk> skal være et helt positivt tal. Funktionen giver en tekst, bestående af det antal mellemrum, som værdien af udtrykket angiver.

SQR

er en standardfunktion. Syntaksen for SQR er:

```
SQR(<numerisk udtryk>).
```

STEP

er et nøgleord, som bruges i forbindelse med FOR-løkker.

Se nærmere under FOR.

STOP

er en sætning, der har syntaksen:

```
STOP
```

eller:

```
STOP <tekst udtryk>
```

Den første form bevirker, at programudførelsen standses og systemet skriver:

```
STOP i linie xxxx
```

Den anden form bevirker, at den ovenstående tekst erstattes med det tekst udtryk, der angives.

Eksempel. Sætningen:

```
STOP "Det var det!"
```

Vil stoppe programudførelsen og skrive: 'Det var det!'.

Programudførelsen kan optages igen med CON (se denne) kommandoen.

Se også CON.

STR$

er en indbygget funktion. Dens syntaks er:

```
STR$(<numerisk udtryk>)
```

Funktionen konvertere værdien af <numerisk udtryk> til den tekst, der angiver ciffernavnet for den pågældende værdi.

SYMBOLS

er en sætning/kommando med syntaksen:

```
SYMBOLS
```

Sætningen/kommandoen udskriver en liste over navnene på alle kendte procedurer, funktioner, labels, variable og tabeller.

Det er altså muligt at få oplyst hvilke procedurer, funktioner og labels der er i programmet ved at skrive:

SCAN

SYMBOLS

TAB

er et nøgleord, der bruges i forbindelse med PRINT sætningen.

Se nærmere under PRINT.

TAN

er en funktion. Syntaksen er:

TAN(<numerisk udtryk>)

hvor <numerisk udtryk> er et tal, der ikke må være i mængden:

PI/2+2*p*PI

hvor 'p' gennemløber alle hele tal.

Funktionen giver tangens af værdien af <numerisk udtryk>, idet denne værdi regnes i grader, nygrader eller radianer.

Se også SETDEG, SETGRAD og SETRAD.

THEN

er et nøgleord, der bruges til at afslutte en IF eller en ELIF sætning med.Brugeren behøver ikke at indtaste dette nøgleord, det bliver altid indsat af systemet, hvis det mangler.

Se også IF.

TIME$

er en tekstfunktion. Dens syntaks er:

```
TIME$
```

Funktionen giver uret (TIMER) som en tekst i standard ISO form:

```
hh:mm:ss
```

'hh' står for timetal, 'mm' for minuttal og 'ss' er sekunder.

Funktionerne kan kun virke, hvis der er Ur-modul i maskinen.

TIMER

kan bruges på to måder. Som sætning/kommando med syntaksen:

```
TIMER <numerisk udtryk>
```

hvor <numerisk udtryk> skal være et positivt tal i det lukkede interval fra 0 til 34359738. Sætningen/kommandoen bevirker, at det indbyggede ur sættes til <numerisk udtryk>, der regnes i sekunder.

TIMER kan også bruges som funktion. Syntaksen er:

```
TIMER
```

Funktionen giver det antal sekunder. COMAL-80/Z80 systemet har været i gang, hvis ikke TIMER er blevet sat til noget andet.

Eksempel. Sætningerne:

```
starttid:=TIMER
LOOP 50000 TIMES NULL
sluttid:= TIMER
PRINT "Løkken tog":sluttid-startttid;"sekunder."
```

beregner tiden for en løkke med 50000 gennemløb.

Funktionerne virker kun hvis der er Ur-modul i maskinen.

TIMES

er et nøgleord, der bruges til at angive et antal gange en løkke skal gennemløbes. Dette nøgleord kan erstattes af et kolon ved indtastning. Systemet vil så lave det om til nøgleordet TIMES.

Se nærmere under LOOP.

TO

er et nøgleord, der bruges i forbindelse med FOR-løkker.

Se nærmere under FOR.

TRACE

sætningen/kommandoen kan have en af følgende fire syntakser:

```
TRACE
TRACE OFF
TRACE "lp:"
```

eller:

TRACE "ds:"

Den første form bevirker, at systemet fra nu af vil udføre en sporing af de linier, der bliver udført. Den anden form bevirker, at sporingen afsluttes.

Sporingen kører på følgende måde:

Den linie, der skal udføres udlistes på skærmen. Systemet venter nu indtil brugeren har trykket på en vilkårlig tast. Efter en tast er blevet trykket vil systemet udføre den udlistede linie og udskrive samtlige udregnede udtryk i denne linie, herunder:

Data læst fra en DATA-kø eller fra tastaruret, med READ eller INPUT.

Data læst fra en fil, med READ FILE eller INPUT FILE.

Parametre til en procedure/funktion/array/deltekst angivelse.

Alle udregnede udtryk, som LET A=88+n udskriver resultatet af 88+n.

Før hver værdi udskrives et '<' og efter værdien udskrives et '>'.

Den tredie og fjerde form af TRACE sætningen/kommandoen bevirker at udlistningen af linier foregår til hhv. printer og skærm.

Bemærk: Hvis tredie eller fjerde form af TRACE benyttes, vil systemet IKKE vente på at brugeren trykker en tast mellem hver linie.

Eksempel. Sætningerne:

```
0010 PRINT "Dette er en test"
0020 TRACE "ds:"
0030 a:=sfd(25,15)
```

```
0040 TRACE OFF
```

I dette tilfælde er det kun linie 0030, der bliver sporet. Man kan her se hvilken værdi variablen 'a' får og om 'sfd' er en tabel eller en funktion.

Hvis man ønsker at slette TRACE sætningerne i et porogram, kan man gøre dette med:

```
DEL <linienummer>,<linienummer>,...
```

men man kan også skrive:

```
DEL TRACE
```

Hvorefter maskinen automatisk vil slette alle linier, der starter med nøgleordet TRACE.

TRAP

er en sætning med syntaksen:

```
TRAP
```

Sætningen bruges til at indlede en del af et program, hvor brugeren har angivet sin egen fejlhåndteringsrutine.

Se nærmere under appendix om fejlhåndtering i COMAL-80/Z80

TRAP ESC

er en sætning, der har syntaksen:

```
TRAP ESC -
```

eller:

```
TRAP ESC +
```

Sætningen bruges til at sætte den normale funktion af <ESC> tasten ud af funktion. Normalt vil et tryk på <ESC> tasten medføre afbrydelse af programudførelsen. Hvis den første form af sætningen/kommandoen er blevet benyttet bevirker et tryk på <ESC> tasten kun at funktionen ESC (se denne) bliver sat til TRUE, mens programudførelsen fortsætter uantastet.

Man kan genskabe den normal virkning af <ESC> tasten ved at bruge den anden form af sætningen.

Se desuden ESC.

TRUE

er en indbygget funktion. Dens syntaks er:

```
TRUE
```

Funktionen giver altid værdien 1, hvilket svarer til "logisk sand".

Se FALSE for flere detaljer.

TRUNC

er en numerisk standardfunktion, der har syntaksen:

```
TRUNC(<numerisk udtryk>)
```

Funktionen giver heltalsværdien til det numeriske udtryk.

Forskellen fra TRUNC til INT funktionen er, at negative tal også bliver rundetned mod 0 . TRUNC er defineret som:

```
TRUNC (x) = SGN(x)*INT(ABS(x))
```

Se også INT.

UNIT

er en sætning/kommando, med syntaksen:

```
UNIT <enhed>
```

hvor <enhed> er et tekstudtryk. Sætningen/kommandoen vælger, hvilken ydre enhed der er den primære, dvs. den enhed, der bliver valgt ved gemning og hentning i filer, når den ikke er angivet direkte. Ved opstart af systemet sættes denne UNIT til 'DKO:'.

Eksempel. Sætningen/kommandoen:

```
UNIT "DK1:"
```

sætter den primære enhed til disketten i drev 1.

UNIT$

er en tekstfkunktion. Dens syntaks er:

```
UNIT$
```

Funktionen giver navnet på den primære ydre enhed.

Ved systemets opstart vil sætningen/kommandoen:

```
PRINT UNIT$
```

give udskriften: 'DKO:'

hvilket viser, at disketten i drev 0 er primær enhed.

UNTIL

er en sætning, der har følgende syntaks:

```
UNTIL <numerisk udtryk>
```

hvor <numerisk udtryk> opfattes som en sandhedsværdi. Sætningen bruges til at afslutte en REPEAT-struktur.

Se nærmere under REPEAT.

UPPER$

er en indbygget tekstfunktion. Syntaksen er:

```
UPPER$(<tekstudtryk>)
```

Funktionen giver <tekst udtryk>, men alle små bogstaver er blevet lavet om til store bogstaver.

Se også LOWER$.

USE

er en sætning/kommando. Dens syntaks er:

```
USE "<pakkenavn>"
```

Sætningen/kommandoen bruges til at aktivere en maskinkodepakke med navnet <pakkenavn>. Pakken under dette navn skal være hentet ind i lageret med LINK-kommandoen (se denne).

Bemærk: De procedurer, man kan bruge efter at have afgivet USE kommandoen slettes af f.eks. SCAN, dvs. hvis pakken ønskes brugt i et program, SKAL der være en USE sætning i programmet. Dette betyder også, at der kan være en USE sætning i en lukket procedure. I så fald er pakken lokal for den pågældene procedure.

USER

er en indbygget funktion. Den har syntaksen:

```
USER
```

Funktionen giver en adresse, der angiver hvor de (med NEW) reserverede ligger.

Se også MAXMEM, NEW og appendix om maskinkode i COMAL-80/Z80.

USING

er et nøgleord, der bruges til at angive format udskrivning af tal eller tekster.

Se yderligere under PRINT.

VAL

er en standardfunktion. Syntaksen er:

```
VAL(<tekst udtryk>)
```

hvor <tekst udtryk> indeholder et ciffernavn. Funktionen giver

værdien af <tekst udtryk>. Teksten kan angive decimale, binære og hexadecimale tal, bliver disse tegn ignoreret.

WHEN

er en sætning. Dens syntaks er følgende:

```
WHEN <udtryk>,<udtryk>,....
```

Sætningen bruges til at indlede en sætningsblok i en CASE-struktur.

Se nærmere forklaring under CASE.

WHILE,ENDWHILE

sætningerne indgår i følgende WHILE-struktur:

```
WHILE <numerisk udtryk> DO
  <programafsnit>
ENDWHILE
```

<programafsnit> udføres, sålænge værdien af udtrykket efter WHILE er TRUE. Når udtrykket bliver FALSE (lig 0), fortsættes udførelsen med sætningen efter ENDWHILE.

Programafsnittet mellem WHILE og ENDWHILE bliver udtrykket under en listning af programmet.

Hvis <programafsnit> kun består af een sætning, kan WHILE løkken skrives på een linie:

```
WHILE x<a(i) DO i:+1
```

ENDWHILE må ikke indsættes i dette tilfælde (jfr. FOR og LOOP).

WRITE FILE

er en sætning. Syntaksen for WRITE er:

```
WRITE FILE <numerisk udtryk>:<udtryk>,<udtryk>,...
```

hvor <numerisk udtryk> angiver et strømnummer.
Sætningen bruges til at skrive i sekventielle eller direkte filer.

Eksempel. Sætningen:

```
WRITE FILE 2:navn$,adresse$,kode
```

vil skrive værdien af de variable på listen sekventielt i filen, der er åbnet med strømnummeret 2. Sætningen:

```
WRITE FILE 4,9:navn$,adresse$,afdnr
```

skriver værdien af de variable på listen i post nr. 9, i den direkte fil, som er åbnet med strømnummeret 4.

Hvis man forsøger at skrive til en post, der ikke eksisterer i filen, bliver filen udvidet automatisk.

En anden måde at bruge filer med direkte tilgang på er:

```
SEEK 4,9
WRITE FILE 4:navn$, adresse$,afdnr
```

Tabeller kan også udskrives i filen. Følgende sætning vil således skrive alle elementer i den todimensionale tabel 'tabel' ud i filen, der er åbnet med strømnummeret 0:

```
WRITE FILE 0:tabel (,)
```

Tabellen skrives ud i følgende rækkefølge:

```
tabel (1,1), tabel (1,2), tabel (1,3), tabel (2,1), ....
tabel (3,3)
```

Disse komponenter kan hentes ind med simple READ FILE sætningen.

ZONE

kan bruges på to måder. Som sætning/kommando med syntaksen:

```
ZONE <numerisk udtryk>
```

Hvor det numeriske udtryk skal være et helt positivt tal, og må ikke overskride bredden af skærmen.

Sætningen/kommandoen bruges til at sætte skrivezonernes bredde.

ZONE kan også bruges som funktion. I så fald er syntaksen:

```
ZONE
```

Funktionen giver den aktuelle værdi af skrivezonernes bredde.

Se om skrivezoner under PRINT.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

CP/M-versionen af COMAL-80/Z 80

TILLÆGSMANUAL TIL CP/M VERSION

COMAL-80/Z80 kører på en række forskellige CP/M systemer. I disse versioner føres al kommunikation med disketten via BDOS. Kommunikation med skærm og tastatur køres dog af COMAL fortolkeren selv. For at opnå fuld kompatibilitet mellem de forskellige CP/M systemer arbejder COMAL fortolkeren med de samme kontrolkoder på alle CP/M systemer. Disse koder er listet senere i tillægsmanualen. ASCII filer skrives på disketten med kontrolkoderne CR (13) og LF (10) efter hver linie. En ASCII fil kan således også læses af et andet program, f.eks.. tekstbehandlingssystemet WordStar.

Der er imidlertidig et mindre problem forbundet med dette. METANIC COMAL-80 skriver ASCII filer uden LF (10) kontrol koden. For at kunne læse et program, der er listet ud af METANIC COMAL-80, kan nedstående program, som ligger på COMAL-disketten køres, idet man før kørsel ændrer Programnavnet 'gammel' i linje 10 til det aktuelle programnavn:

```
0010 navn$:="gammel"// navn på METANIC COMAL-80 ascii fil
0020 DIM a$ OF 1000
0030 RENAME navn$+".cml",navn$+".bak"
0040 OPEN FILE 1.navn$+".bak",READ
0050 OPEN FILE 2.navn$+".cml",WRITE
0060 LOOP
0070   a$:=GET$(1,1)
0080   EXIT WHEN NOT a$ IN "0123456789"
0090   REPEAT
0100     a$:+GET$(1,1)
0110   UNTIL CHR$(13) IN a$
0120   PUT FILE 2: a$+CHR$(10)
0130 ENDLOOP
0140 CLOSE
0150 DELETE navn$+".bak
```

Fra METANIC-COMAL-80 version 1.6c og fremefter kan man dog klare problemet lettere, idet man i disse versioner kan ud-LIST'e programmerne på disketten - med den manglende CHR$(10) i slutningen af hver linje - ved at skrive /C efter programnavnet.

Eks. Kommandoen: LIST DK1:VEJLEDNI/C

vil i nævnte METANIC-versioner bevirke, at det i programlageret værende program udskrives under navnet VEJLEDNI.CML og påen sådan form, at det kan indlæses med COMAL-80/Z80 - eller med PASCAL eller med tekstbehandlingssystem.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

KONTROL KODER VED PRINT

Ved at sende en kontrol kode til PRINT rutinen, kan man opnå, at få en række specielle funktioner udført.

""2"" CUROFF

Markøren slukkes.

""3"" CURON

Markøren tændes.

""5"" BACK

Markøren bevæges en position mod venstre.

""6"" FOR

Markøren bevæges en position mod højre.

""7"" BELL

Systemet giver en kort lyd fra sig, hvis dette er muligt.

""8"" BS

Tegnet til venstre for markøren slettes.

""9"" TAB

Markøren bevæges frem til begyndelsen af næste ZONE.

""10"" DOWN

Markøren bevæges en position nedad.

""11"" UP

Markøren bevæges en position opad.

""12"" PAGE

Skærmen slettes.

""13"" CR

Markøren flyttes til den første position i den pågældene linie.

""16"" INS LINE

Indsætter en tom linie, der hvor markøren er placeret.

""17"" DEL LINE

Sletter en linie, der hvor markøren er placeret.

""18"" HOME

Flytter markøren til position 1,1.

""19"" EOL

Sletter resten af linien, derfra hvor markøren er placeret.

""20"" SPECIAL 0

Speciel kontrol kode nr. 0. Funktionen er afhængig af systemet, og kan f.eks. bruges til at tænde og slukke en statuslinie.

""21"" SPECIAL 1

Speciel kontrol kode nr. 1. (Se SPECIAL 0)

""22"" SPECIAL 2

Speciel kontrol kode nr. 2. (Se SPECIAL 0)

""23"" SPECIAL 3

Speciel kontrol kode nr. 3. (Se SPECIAL 0)

""24"" SPECIAL 4

Speciel kontrol kode nr. 4. (Se SPECIAL 0)

""25"" WIDTH

Skift mellem 40 og 80 tegns skærmbredde, hvis dette er muligt på systemet.

""26"" FLASH

Skift mellem normal og blinkende udskrivning, hvis dette er muligt på systemet.

""27"" ESC

Bevirker at det næste tegn betragtes som et bogstav. Dvs. eventuelle kontrolkoder bliver IKKE adlydt, hvis dette er muligt på systemet.

""28"" HIGHLIGHT

Skift mellem normal og fremhævet udskrivning, hvis dette er muligt på systemet.

""29"" INVERSE

Skift mellem normal og omvendt udskrivning, hvis dette er muligt på systemet.

""30"" UNDERLINE

Skift mellem normal og understreget udskrivning, hvis dette er muligt på systemet.

""31"" INS

Indsætter et mellemrun i teksten på markørens position.

""127"" DEL

Sletter tegnet på markørens position.

CP/M FILNAVNE

Et filnavn er et navn på en fil, der er gemt på disketten. Det skal være et lovligt comal-navn på maksimalt 8 tegn. Efter dette navn kan der angives et punktum efterfulgt af en fil-type, så man kan skelne mellem de forskellige filer. Hvis man ikke selv angiver en fil-type vil COMAL-80/z80 bruge følgende filtyper:

'DTA' ved DaTA Screen'
'RND' ved RaNDom filer

```
        'LST'  ved  programmer, der er gemt med LIST
        'SAV'  ved  programmer, der er gemt med SAVe
        'pck'  ved  ved maskinkodepakker
```

Udover normale filnavne er der følgende specielle filnavne:

```
        'DS:'  for  'Data Screen'
        'LP:'  for  'CP/M Line Printer'
        'PU:'  for  'CP/M PUnch device'
        'RD:'  for  'CP/M ReaDer device'
        'KB:'  for  'KeyBoard'
```

ADRESSER I COMAL-80

I dette afsnit er der listet adresser en række maskinkoderutiner, som brugeren kan benytte i pakker oa. Efter disse rutiner er der en liste over samtlige systemvariable i COMAL-80. Adresserne, der angives her, er placeret i selve COMAL-80 fortolkeren, men ligger i en tabel, der ser således ud:

```
    257:  JP      PRINT
          JP      KEY
          JP      ERROR
          ...
```

CALL 257 bevirker altså, at den første rutine i tabellen kaldes, ligeledes bevirker CALL 260, at den anden rutine kaldes osv.

257

PRINT-rutine. Denne rutine bevirker, at A registeret bliver udskrevet på skærmen. Et kald til denne rutine svarer præcis til:

```
        PUT FILE 1:CHR$(A)
```

hvis strømnummeret 1 er åbent til skærmen (DS:).

260

KEY-rutine. Denne rutine bevirker, at cursoren tændes. COMAL-80 venter på, at der trykkes en tast, og placerer tastens værdi i A

registeret. Et kald til denne rutine svarer præcis til:

```
A=ORD(GET$(5,1))
```

hvis strømnummeret 5 er åbnet fra tastaturet.

263

ERROR-rutine. Denne rutine bevirker, der gives en fejlmeddelelse. Fejlmeddelelsens nummer er angivet af A registeret. Et kald til denne rutine svarer præcis til:

```
REPORT A-1
```

266

ERROR19-rutine. Denne rutine giver fejlmeddelelsen:

```
Tal udenfor tilladt område.
```

Svarer til:

```
REPORT 19
```

269

ERROR46-rutinen. Denne rutine giver fejlmeddelelsen:

```
Fejl i tekst.
```

Svarer til:

```
REPORT 46
```

272

LPPRINT-rutine. Denne rutine svarer til rutinen på 257, blot sendes bogstavet til CP/M LST-device i stedet for til skærmen.

275

PUNCH-rutine. Denne rutine svarer til rutinen på adresse 257, blot sendes bogstavet til CP/M PUN-device i stedet for til skærmen.

278

READER-rutine. Denne rutine svarer til rutinen på adresse 260, blot hentes bogstavet fra CP/M RDR-device i stedet for til tastaturet.

281

CURSOR-rutine. Denne rutine placerer cursoren på den position, BC registeret angiver. B registeret angiver linien, og C registeret angiver positionen i den pågældene linie. Svarer til:

```
CURSOR B+1,C+1
```

Andre adresser

Udover disse rutiner, angives følgende adresser:

16

Henter værdien af det tal, der ligger øverst på kalkulatorstakken, tallet placeres i BC registeret. Der gives en fejlmeddelelse, hvis tallet er mindre end 0, eller hvis tallet er større end 65535.

24

Henter pointere til det øverste element på kalkulatorstakken. Efter kaldet vil DE pege ummiddelbart efter det øverste element (STKEND), HL vil pege på værdien af øverste element og BC vil være 5, hvis det øverste elemtent er et tal, ellers vil BC angive længden af den tekst, der ligger øverst.

Se yderligere information om stakkens opbygning i manualen.

40

Package call rutinen. Et kald til denne rutine vil bevirke, at systemet hopper videre. Adressen på den faktiske rutine angives af en relativ adresse, placeret på de efterfølgende to bytes.

Se yderligere information om denne rutine i manualen.

48

Print i destination. Næsten som rutinen på 63455, bortset fra at denne rutine, vælger det strømnummer, der ligger i systemvariablen DESTIN, og kalder den rutine, der er angivet i DSTTAB+2*DESTIN. Normalt vil værdien 8 ligge i DESTIN, hvilket angiver, at der skal skrives i den destination, som SELECT OUTPUT har valgt.

SYSTEMVARIABLE

adresse	antal	navn	kort forklaring
256	1	LASTKY	Sidst trykkede tast
257	87	JMPTAB	Jump tabel
344	1	LNCLED	'leader' til næste tabel (altid 0)
345	25	LINCON	Fortæller om linie fortsætter
370	1	LOWLIN	Angiver antal linier på skærmen
371	1	FYSLEN	Fysisk skærmbredde
372	2	INBPNT	Pointer til INBUF
374	2	CODPNT	Pointer til CODBUF
376	2	SYNPNT	Pointer til syntax tabellen
378	2	LINENR	Linienummer i indtastet linie
380	1	ERRNR	Error nummer (bruges kun internt)
381	1	RESTYP	Resultat type under syntax check
382	36	MEMSTK	6 memory lokationer til FP-tal (6-bytes)
418	6	LOGBAS	FP for LOG(logbase)
424	6	TRIGBA	FP for 2*PI, 360 eller 400
430	10	NUMBUF	Buffer til at gemme tal, der skal udskrives
440	1	TOTAL	Antal digits i NUMBUF
441	1	BEFORE	Antal digits før kommaet i NUMBUF
442	1	SIGN	Fortegn for tallet i NUMBUF
443	1	PARM	Parameter til nogle af kalk, funktionerne
444	2	HLSTO	Registerparret HL
446	2	DESTO	Registerparret DE
448	2	BCSTO	Registerparret BC
450	2	AFSTO	Registerparret AF
452	1	ZONSTO	Værdien for ZONE
453	1	ESCSTO	Værdien for ESC
454	1	EODSTO	Værdien for EOD
455	2	SEED	Randomsize SEED
457	2	LSTSTA	Peger på sidste statement under syntax check
459	2	ENDPGM	Fysisk slutning af fortolkeren
461	2	LISTSP	SP værdi under LIST

463	1	LISTYP	Y tæller under LIST
464	1	LISTXP	X tæller under LIST
465	1	LINLST	Flag, der angiver om linienumre skal listes
466	1	ANTSPC	Antal indrykninger
467	1	FSETLA	SETLABEL flag
468	1	FSETLE	SETLET flag
469	1	FSETEX	SETEXEC flag
470	2	AUTLIN	Linienummer under AUTO
472	2	AUTINC	Increment værdi under AUTO
474	1	AUTFLG	AUTO flag
475	1	EDTFLG	EDIT flag
476	1	FINFLG	FIND flag
477	1	CHNFLG	CHANGE flag
478	1	ENTFLG	ENTER flag
479	2	THSADR	Adresse hvor den sidste telst blev fundet
481	1	RDFLAG	READ/READ FILE flag
482	1	TRPESC	TRAP ESC flag
483	2	BASADR	Base adresse under udregning af array-item
485	2	MLTFAK	Mult.faktor under udregning af array-item
487	2	LSLOOP	Sidste LOOP/FOR/REPEAT/WHILE under SCAN
489	2	LSFREE	Sidste FREE værdie. Bruges ved SCAN af PROC's
491	2	LSDATA	Sidste DATA sætning
493	2	ERRCUR	Position hvor error-linien er taget fra
495	1	INPLEN	Længde af input i INPUT sætning
496	1	RETVAL	Return værdi flag
497	2	VALADR	Adresse på return værdi
499	2	VALLEN	Længde af return værdi
501	1	VARTYP	Type af variabel
502	2	DATAPT	DATA pointer
504	2	FRSTDT	Første DATA (RESTORE)
506	1	ANTFIL	Fil-tæller (bruges under disk-DIR)
507	1	FILTYP	Default fil type nummer
508	2	TRACIN	TRACE flag
510	1	TRCDEL	TRACE delay
511	1	VDUWRT	DS flag
512	2	MSKLEN	Længde af maskin-stakken
514	4	CLOCK	TIMER*125 værdi
518	160	OLDLIN	Gammel error-linie
678	160	FINTXT	Tekst der skal findes (FIND, CHANGE)
838	10	EOFSTO	EOF flag for 10 filer
848	400	FCBADR	FCB for de forskellige filer
1248	1280	DMAADR	DMA for de forskellige filer
2528	256	CODBUF	Kode buffer
2784	1	INBLED	Input buffer 'leader' (altid=255)
2785	256	INBUF	Input buffer
3041	42	DSTTAB	Adresser på strømnumres rutiner
3083	1	SCNFLG	SCAN flag
3084	1	CONFLG	CON flag

3085	1	XPOSI	CURPOS værdi (0-79)
3086	1	YPOSI	CURLIN værdi (0-23)
3087	1	DESTIN	Destination (0-11)
3088	2	RUNLIN	Linienummer på linien, der udføres nu
3090	2	NXTLIN	Adresse på næste linie
3092	2	LSTKEN	Sidste STKEND værdi
3094	2	ENPRSP	SP værdi ved ENDPROC
3096	2	ERRSTO	ERR værdi
3097	1	INTRAP	Fortæller om fortolkeren er i en TRAP-HANDLER
3098	1	ERRST1	Error nummer (bruges ved FOR-ENDFOR)
3099	1	ERRST2	Error nummer (bruges ved PROC-ENDPROC)
3100	2	ERRLIN	Linie nummer, hvor fejlen opstod
3102	2	LSIMPO	Adresse hvorfra der skal IMPORT'es
3104	2	CNERSP	ERRSP værdi ved CON
3106	2	CONTSP	SP værdi ved CON
3108	18	CONARE	Område hvor RUNLIN-LSIMPO gemmes ved ESC/STOP
3126	2	PROG	Adresse på program i hukommelsen
3128	2	EDTADR	EDIT start adresse
3130	2	EDTEND	EDIT slut adresse
3132	2	VARS	Adresse på 3-bytes værdierne i VARS
3134	2	NAMES	Adresse på navnetabellen
3136	2	STACK	Adresse på kalk. stakken
3138	2	STKEND	Adresse efter kalk. stakken
3140	2	VARSF	Værdi for VARS under syntax check
3142	2	NAMESF	Værdi for NAMES under syntaks check
3144	2	STACKF	Værdi for STACK under syntaks check
3146	2	STKENF	Værdi for STKEND under syntaks check
3148	2	ERRSP	SP værdi ved ERROR
3150	2	FREE	Adresse på variablernes værdier
3152	2	PCKSTK	Adresse på pakkerne
3154	2	SPBOT	Adresse på maskinstacken
3156	2	CLRADR	USER værdi
3158	2	MAXMEM	MAXMEM værdi

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

Appendix A - G

APPENDIX A : Udtryk i COMAL-80/Z80.

I COMAL-80/Z80 skelnes der mellem numeriske udtryk og tekstudtryk.

Et numerisk udtryk kan indeholde numeriske konstanter, variable og funktioner. Der kan bruges paranteser og de følgende regnetegn (operatorer) i henhold til matematikkens sædvanlige regler.

+	Monadisk +	+<udtryk>
-	Monadisk -	-<udtryk>
^	Potensopløftning	<udtryk>^<udtryk>
*	Multiplikation	<udtryk>*<udtryk>
/	Division	<udtryk>/<udtryk>
DIV	Heltalsdivision	<udtryk> DIV <udtryk>
MOD	Rest ved division	<udtryk> MOD <udtryk>
+	Addition	<udtryk>+<udtryk>
-	Subtraktion	<udtryk>-<udtryk>
BITAND	'Logisk og'	<udtryk> BITAND <udtryk>
BITOR	'Logisk eller'	<udtryk> BITOR <udtryk>
BITXOR	'Eksklusivt eller'	<udtryk> BITXOR <udtryk>

Bemærk: <udtryk> kan betyde såvel en konstant, en variabel som en formel, kort sagt alt, hvad der kan give en talværdi direkte eller ved udregning.

I COMAL-80/Z80 kan numeriske konstanter angives i to talsystemer, udover det normale, decimale talsystem:

Hexamalt: Angives ved at placere et '$' før tallet.

Binært: Angives ved at placere et '%' før tallet.

Eksempler:

$FF04 giver konstanten 65284.

%10111101 giver konstanten 189.

Et numerisk udtryk kan bruges som boolsk udtryk, f.eks. i IF, WHILE og mange flere sætninger, hvor det numeriske udtryk angiver en sandhedsværdi. Hvis det numeriske udtryk udregnes til 0, fortolkes udtrykket som falsk (FALSE), hvorimod en værdi forskellig fra 0 fortolkes som sand (TRUE).

Følgende boolske operatorer returnerer enten 0 eller 1:

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

```
  <       Mindre end.
 <=       Mindre end eller lig med.
  =       Lig med.
 >=       Større end eller lig med.
  >       Større end.
 <>       Forskellig fra.
NOT       logisk negation.
AND       logisk disjunktion.
 OR       logisk konjunktion.
```

Eksempler:

NOT <udtryk>

har værdien TRUE, hvis <udtryk> er FALSE, ellers FALSE.

<udtryk> AND <udtryk>

har værdien TRUE, hvis begge <udtryK> er TRUE, ellers FALSE.

<udtryk> OR <udtryk>

har værdien FALSE, hvis begge <udtryk> er FALSE, ellers TRUE.

Et tekstudtryk kan være en tekstkonstant, en tekstvariabel, en tekstfunktion, eller flere sådanne sammensat med tegnet '+'. Man kan også multiplicere en tekst med et tal, hvilket giver teksten sammensat med sig selv, det antal gange tallet angiver. I tekst udtryk kan man også benytte paranteser, da multiplikation stadig er højere prioriteret end addition.

Eksempler:

```
"Anette"+"Asmussen"

"Brian og"+" Henrik,"*5+" Henrik"

20*("Jeg er "+a$)
```

Mellem tekstudtryk kan man anvende relationerne:

```
  <       Kommer før.

 <=       Kommer før eller er lig med.
```

=	Er lig med.
>=	Følger efter eller er lig med.
>	Følger efter.
<>	Er forskellige fra.

Endvidere er følgende relation til rådighed:

IN	Der betyder 'deltekst af'

<tekstl> IN <tekst2>

Giver værdien FALSE, hvis <tekstl> ikke findes som deltekst i <tekst2>. Hvis den findes, vil værdien være lig positionen af <tekstl> i <tekst2>. Hvis <tekstl> er den tomme tekst, vil resultatet give længden af <tekst2>+1.

Eksempel:

"-" IN "COMAL-80/Z80" giver værdien 6.

Prioriteten af de nævnte operatorer er:

```
^
* / DIV MOD
Monadisk +/-
+ -
BITAND BITOR BITXOR
< <= = >= > <>
NOT
AND
OR
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

APPENDIX B : Fejlhåndtering i COMAL-80/z80

Hvis der under udførelsen af et program opstår en fejl, standser COMAL normalt programudførelsen og giver en fejlmelding. I nogle tilfælde kan dette være en meget uhensigtsmæssig reaktion, og derfor findes der i COMAL en særlig styrestruktur, ved hjælp af hvilken brugeren selv kan bestemme, hvilken virkning visse fejl skal have på programmets videre afvikling. Man omtaler dette som programstyret fejlhåndtering, og den tilsvarende styrestruktur er opbygget således:

```
TRAP
  <programafsnit 1>
HANDLER
  <programafsnit 2>
ENDTRAP
```

Det første programafsnit (TRAP-delen) består af de sætninger, som skal overvåges. Det andet afsnit (HANDLER-delen) består af de sætninger, der skal udføres, hvis der opstår en fejl i en af sætningerne i det første afsnit.

Hvis sætningerne i det første afsnit udføres uden fejl, fortsættes programudførelen med sætningerne efter ENDTRAP, dvs. HANDLER-delen springes over.

Hvis der opstår en fejl under udførelsen af en sætning i det første afsnit, fortsættes porgramudførelsen med HANDLER-delen. Fejlens nummer bliver husket så længe det andet afsnit udføres, og kan læses i funktionen ERR. Hvis der opstår fejl i udførelsen af HANDLER-delen, er der 2 muligheder: Hvis TRAP-strukturen overvåges af en ydre TRAP struktur, overtager dennes HANDLER-del udførelsen; hvis en sådan ikke findes, overtager systemet håndteringen af fejlen på sædvanlig vis.

Hvis en procedure kaldes fra TRAP-delen, og der opstår en fejl under udførelsen af proceduren, behandles der, som om fejlen var opstået på det sted, hvor procedurekaldet står. Programudførelsen fortsættes altså med HANDLER-delen.

Eksempler på fejlhåndtering:

```
0010 LOOP
0020   INPUT "Angiv n: ":n
0030   fak:=1
0040   TRAP
0050     FOR i:=2 TO n DO fak:=fak*i
0060     PRINT n,"! = ",fak
0070   HANDLER
0080     PRINT "n er for stor"
0090   ENDTRAP
0100 ENDLOOP
```

```
0010 FUNC f(n) CLOSED
0020   FUNC fak (n)
0030     IF n=0 THEN RETURN 1
0040     RETURN fak (n-1)*n
0050   ENDFUNC fak
0060
0070     TRAP
0080       værdi:=fak(n)
0090     HANDLER
0100       værdi:=1e+38
0110     ENDTRAP
0120     RETURN værdi
0130 ENDFUNC f
```

Mens systemet udfører HANDLER-delen har du følgende sætninger og funktioner til rådighed:

REPORT

er en sætning. Den kan have syntaksen:

```
REPORT
```

eller:

```
REPORT <numerisk udtryk>
```

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 1 til 75. Den første form bevirker at systemet udskriver den fejlmeddelelse, TRAP-strukturen fangede. Den anden form bevirker at systemet giver fejlen med det nummer, som værdien af <numerisk udtryk> angiver.

Eksempel. Sætningen:

```
REPORT 6
```

Vil give fejlmeddelelsen: 'linieslut forventet'.

ERR

er en funktion, der kan bruges i HANDLER-delen af en TRAP-struktur.

Syntaksen for ERR funktionen er:

```
ERR
```

Funktionen giver nummeret på den fejl, der opstod i TRAP-delen, og som var skyld i at HANDLER-delen blev kaldt.

ERRTEXT$

er en tekstfunktion. Dens syntaks er:

ERRTEXT$(<numerisk udtryk>)

hvor <numerisk udtryk> skal være et helt tal i det lukkede interval fra 1 til 75. Funktionen giver teksten for den fejlmeddelelse, der har det nummer, som værdien af <numerisk udtryk> angiver.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

APPENDIX C : Maskinkode i COMAL-80/z80

NB! Dette appendix forlanger en vis kendskab til maskinkode. Brugeren skal kende registrene i Z80 processoren og kende til kodning af Z80 processoren.

Der er 3 forskellige måder, at implementere maskinkode i COMAL-80/Z80 på.

Man kan angive koden direkte in-line ved brug af CODE sætningen.

Man kan anvende POKE sætninger til at lagre maskinkode og derpå starte udførelsen med CALL sætningen.

Den tredie og sidste metode består i at anvende maskinkodepakker.

Dette er beskrevet i appendix om pakker i COMAL-80/Z80.

CODE sætningen

har syntaksen:

```
CODE <udtryk>,<udtryk>,<udtryk>,...
```

Når en CODE sætning mødes af COMAL-80/Z80, bliver alle udtryk udregnet et efter et, og lagt ned i hukommelsen som enkelte bytes. Efter hele linien er regnet igennem, bliver der udført et CALL til den første byte på rækken.

Ved udførelsen af denne maskinkode vil, registrene AF, BC, DE og HL indholde den værdi, de er sat til af brugeren, og ved returnering fra maskinkoden kan man fra COMAL-80/Z80 igen hente registrenes værdier, ved at bruge f.eks. BC som funktion. Eksempel:

```
BC 12 // Sætter BC registret til 12

CODE $1,0,0 // Sætter BC registret til 0

PRINT BC // Skriver 0 ud på skærmen
```

CALL sætningen

har syntaksen:

CALL <adresse>

Sætningen kan bruges til at kalde en maskinkoderutine, der er lagret på adressen <adresse> i hukommelsen, ved brug af POKE og DPOKE.

Her gælder det samme som ovenfor, at registrene AF, BC, DE og HL kan sættes før kald af maskinkoden og læses efter udførelse af koden.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 icicph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

APPENDIX D : Pakker i COMAL-80/Z80.

NB! Dette appendix forlanger en vis kendskab til maskinekode. Brugeren skal kende registrene i Z80 processoren og kende til kodning af Z80 processoren.

Hvis man ønsker at implementere maskinkode i COMAL-80/Z80 programmer, kan dette gøres på 3 forskellige måder. To af disse metoder er forklaret i appendix C. Den sidste metode er:

MASKINKODEPAKKER

En pakke kan indeholde procedurer og funktoner i maskinkode.

Procedurer/funktioner i en pakke kaldes på præcis samme måde som normale COMAL-80/Z80 procedurer/funktioner.

En pakke bliver hentet ind i hukommelsen (fra disketten) med LINK.

En pakke bliver slettet af hukommelsen med DISCARD.

En pakke tages i brug med USE.

Bemærk: Følgende kommandoer bevirker at alle pakker bliver slettet af hukommelsen:

NEW, QUIT, LOAD, ENTER.

Det skulle nu være klart, hvad en pakke er. I det følgende gennemgås, hvorledes en pakke skal være opbygget.

Bemærk: Pakker skrevet til COMAL-80/Z80 skal være fuldt relokerbare:

1: Versions nummer

I den første byte i pakken skal versionsnummeret anføres.

COMAL-80/Z80 rev 1.70 forlanger versionsnummer 1.

2: Pakkens længde

I disse to bytes skal der anføres, hvor mange bytes resten af pakken fylder.

3: Navnetabellen

Navnetabellen er en angivelse af navnene på forskellige procedurer og funktioner i pakken.

Først skal der angives to bytes, som angiver, hvor mange bytes selve navnetabellen fylder.

Efter denne længdeangivelse, skal selve navnetabellen være. Hvert navn skal afsluttes med en byte, der har værdien 255.

Eksempel:

```
        DW      10
        DB      "PLOT",255
        DB      "DRAW",255
```

De 3 dele, der indtil nu er omtalt bliver kun brugt af LINK kommandoen og bliver derfor ikke i lageret efter at LINK er udført.

4: Relativ adresse på næste pakke

En relativ adresse er to bytes, der skal adderes til den adresse, hvor de er placeret. Dette giver så den absolutte adresse.

Den relative adresse der angives her, skal pege umiddelbart efter den sidste byte i pakken. Det gøres i de fleste Z80 assemblere på følgende måde:

```
        DW      PCKEND-$
```

```
        (Selve pakken)

PCKEND: END
```

Eksempel:

Hvis resten af pakken fylder 43 bytes skal den relative pegepind indeholde tallet 45, da relative adresser også skal medregne de to bytes som adressen ligger i.

5: Pakkens navn

Pakkens navn angives ved først at angive en byte, som fortæller, hvor mange tegn selve navnet fylder. Herefter angives navnet i ascii-code, uden afsluttende tegn.

Eksempler:

```
        DB      6,"turtle"
        DB      8,"graphics"
```

6: Data område

På dette punkt i pakken kan der angives et data-område. Først skal der med en byte angives, hvor mange bytes, data-området fylder. Herefter skal selve data-området angives. Når en pakke kaldes vil IY registret pege på pakkens data-område. Hvis du altså ønsker at hente den 6. byte i data-området skal instruktionen:

```
        LD      A,(IY+5)
```

udføres. Instruktionerne:

```
        PUSH    IY
        POP     HL
```

får HL til at pege på data-området.

Bemærk: COMAL-80/Z80 bruger ikke IY registret til andet end pegepind til pakke-data. Du behøver derfor ikke gemme IY registret, hvis du skal kalde en rutine i selve COMAL-80/Z80 kernen.

7: Relative adresser på initialiseringsrutinerne

En pakke skal indeholde 3 initialiseringsrutiner:

En der kaldes, når der udføres en USE på pakken.

En der kaldes, når der udføres en DISCARD på pakken.

En der kaldes, hver gang systemet befinder sig i skærmeditoren.

Adresserne på disse 3 rutiner er relative, og fylder hver to bytes. De angives på samme måde, som den relative adresse på den næste pakke (punkt 4).

Hvis man har en pakke ved navn 'turtle' og skriver:

```
USE "turtle"
```

Kaldes den første af de tre rutiner. Man kunne forestille sig, at denne rutine lavede billedet til en grafik-skærm og tegnede en turtle på midten af skærmen.

Hvis man senere skriver NEW eller DISCARD, skal pakken smides ud af hukommelsen. Umiddelbart før dette bliver gjort, bliver rutine nr. 2 kaldt. Man kunne forestille sig, at denne rutine lavede skærmen til en tekst-skærm igen.

Den sidste rutine kaldes temmelig ofte. Hver gang brugeren skal indtaste en kommando (i skærmeditoren), bliver rutinen kaldt. Denne rutine skal sikre, at brugeren kan se, hvad han skriver. Rutinen kunne eventuelt checke, om hele skærmen er grafik-skærm, og hvis dette var tilfældet, da lave det til en tekstskærm.

Procedurernes/funktionernes numre

Dette afsnit skal IKKE udfyldes, det klarer LINK kommandoen.

Der skal kun angives to bytes, der skal angiver antallet af procedurer og funktioner i pakken, multipliceret med 2.

Efter disse to bytes skal der være et antal frie bytes, svarende til 2 bytes pr. procedure/funktion.

Hele dette afsnit vil se således ud:

```
DW      4
DS      4
```

hvis der er 2 procedurer/funktioner i pakken.

9: Procedure/funktion definition

Dette afsnit skal angive en række bytes for alle procedurer og funktioner i pakkken. Disse bytes skal fortælle type, antal parametre, disse parametres typer samt en relativ adresse på selve proceduren.

9.1: Type byte

Denne type-byte skal være 7, hvis der er tale om en procedure. Hvis det er en funktion, skal type-byten være 0. Typen af funktionen angives ved at sættes "#" eller "$" efter funktionens navn i navnetabellen (punkt 3).

9.2: Antal parametre

Her skal der angives en byte, der svarer til antallet af parametre (punkt 9.2).

Disse type-bytes skal være:

0 hvis det er en numerisk parameter og

2 hvis det er en tekst parameter.

9.4: Relativ adresse på rutinen

Den relative adresse angives som forklaret i punkt 4. Den angiver hvor selve proceduren starter.

Hele afsnit 9 kunne altså se således ud:

```
        DB      7           ;Procedure
        DB      2           ;2 parametre
        DB      0           ;1. parametertype=numerisk
        DB      2           ;2. parametertype=tekst
        DW      proc-$      ; adresse rutine
proc:   (Selve proceduren)
        RET
```

10: Selve pakken

Her skal selve pakken nu ligge.

Alle procedurer/funktioner i vilkårlig orden.

Husk stadig: Pakken skal være fuldt relokerbar, dvs. der må IKKE være nogen JP eller CALL instruktioner mellem forskellige rutiner i pakken.

JP instruktionen skal laves relativ ved at bruge JR instruktionen i stedet for. Men en pakke hvor man ikke må kalde fra en rutine til en anden, er meget svær at skrive.

Til dette formål er der lavet en lille rutine i COMAL-80 der kaldes 'pcall' (Package CALL). Med denne rutine kan du kalde relativt rundt omkring i pakken. Eksempel:

```
        CALL pcall
        DW   rutine-$
```

adressen på pcall rutinen, og mange andre hjælperutiner, samt alle systemvariable, er opgivet i tillægsmanualen.

Yderligere oplysninger

Et sidste punkt i oplysningen omkring pakker er, hvordan man læser parametrene, og hvordan man returnerer en værdi til COMAL-80 fra en funktion i pakke. COMAL-80 arbejder meget simpelt på dette punkt. Den arbejder med en kalkulator-stak, hvor både tal og tekste bliver gemt. Når COMAL-80 kalder selve pakkens procedure vil alle parametre være udregnet, og være lagret på denne stak. Således vil følgende procedurekald:

```
plottekst(x,y,tekst$)
```

give anledning til at stacken vil se således ud:

```
tekst$

y

x
```

Hvor det første element på stakken altså er teksten fra 'tekst$', det næste element er værdien for variablen 'y', og nederst ligger værdien for 'x'.

Nu er det kun et spørgsmål om, hvordan man læser disse parametre fra denne stak. I tillægsmanualen er der opgivet en adresse på en systemvariabel der hedder STKEND og angiver adressen umiddelbart efter det øverste element på stakken. Tal fylder altid 6 bytes på stakken, og tekste fylder 3 bytes udover længden af teksten.

En tekst ligger således på stakken:

```
STKEND

        !---------!

        !    2    !

        !  High   !

        !  Low    !

   ... Selve teksten ...

        !---------!


  ....  Program mm. ....
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

0000

Hvis man altså vil læse en tekst fra stakken, skal man først gøre:

```
STKPNT: LD      HL, (STKEND)
        DEC     HL
        DC      HL
        LD      B, (HL)
        DEC     HL
        LD      C, (HL)
        AND     A
        SBC     HL, BC
```

Efter disse operationer vil HL registret pege på det første tegn i teksten,og BC registret vil indeholde længden af teksten.

Bemærk: Hvis man har en procedure med parametre skal man huske, at slette alle parametre fra stakken inden man hopper tilbage til COMAL fortolkeren. Hvis der blot var en tekst på stakken, gøres dette ved at lægge adressen på det første tegn i teksten, ned i STKEND. Hvis der er flere parametre skal man finde adressen på det nederste element af parameter-elementerne og placere denne adresse i STKEND.

Hvis det er et tal, der ligger øverst på kalkulatorstakken vil den se således ud:

STKEND

```
        !---------!
        !    0    !
        !   tal   !
        !   tal   !
        !   tal   !
        !   tal   !
```

```
        !Eksponent!
        !---------!
```

Man taler her om en 5-bytes flydende tal form:

```
Byte nr.   1     2     3     4     5
        !-----!-----!-----!-----!-----!
        ! EXP ! TAL ! TAL ! TAL ! TAL !
        !-----!-----!-----!-----!-----!
```

Hvor det øverste bit i byte nr. 2 angiver fortegnet for tallet.

Bemærk: Hvis eksponenten er 0 er tallet et lille heltal, dvs. et helt tal, der ligger i det lukkede interval fra -65535 til 65535.

I dette tilfælde angiver den byte fortegnet for tallet.

255 betyder negativ og
0 betyder positiv

Den tredie og fjerde byte angiver hhv. lav og høj del af tallet.

Hvis man altså vil hente et lille heltal fra stakken kan man:

```
INTBC:  LD      HL, (STKEND)
        DEC     HL
        DEC     HL
        DEC     HL
        LD      B, (HL)
        DEC     HL
        LD      C, (HL)
        DEC     HL
        LD      A, (HL)
```

```
        DEC     HL
```

efter disse instryktioner, vil HL pege på tallets eksponent, A er den byte, der angiver fortegnet, og BC indeholder tallet.

Bemærk: Hvis fortegnet er minus, vil BC=65536-ABS(tal).

Eksempel. Hvis tallet er -1 vil

A=255 og

BC=65535.

Se i tillægsmanualen adresser på rutine mm. i COMAL-80.

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 icicph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

APPENDIX E: OVERSIGT OVER GRAFIKPROCEDURER

Her anføres i alfabetisk orden de procedurer og funktioner, som i grafikpakken stilles til rådighed for brugeren.

Bemærk at nogle procedurenavne skal bruges i forkortet form: (GRAPHICSSCREEN = gs, o. lign.)

Grafikpakken skal LINK'es til programmet, og der skal først i programmet være en USE "GRAPHICS" sætning, før grafikprocedurerne/funktionerne kan anvendes.

Hvis man SAVE'er et program, som har tilkoblet grafikpakke, gemmes grafikpakken med programmet, således at når man LOAD'er det senere er grafikpakken tilkoblet.

```
**********************************
** PROC ARC(x,y,r,s,v)           ** Tegner buestykke
**   Parametre                   **
**     centrum-x                 **
**     centrum-y                 **
**     radius                    **
**     startvinkel (0 - 360)     **
**     vinkel      (0 - 360)     **
**********************************

**********************************
** PROC BOX(x,y,xl,yl,v)         ** Tegner rektangel
**   Parametre                   **
**     x-koordinat               **
**     y-koordinat               **
**     x-længde                  **
**     y-længde                  **
**     vinkel      (0 - 360)     **
**********************************

**********************************
** PROC CIRCLE(x,y,r)            ** Tegner cirkel
**   Parametre                   **
**     centrum-x                 **
**     centrum-y                 **
**     radius                    **
**********************************
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

```
**********************************
** PROC CLEARSCREEN  (cs)       ** Sletter grafikskærm
**********************************

**********************************
** DISCARD funktionen           ** Frakobler grafikpakke
**********************************

**********************************
** PROC DRAW(x,y)               ** Tegner linje
**   Parametre                  **
**     relativ x-koordinat      **
**     relativ y-koordinat      **
**********************************

**********************************
** PROC DRAWTO(x,y)             ** Tegner linje
**   Parametre                  **
**     x-koordinat              **
**     y-koordinat              **
**********************************

**********************************
** PROC ELLIPSE(x,y,rx,ry)      ** Tegner ellipse
**   Parametre                  **
**     centrum-x                **
**     centrum-y                **
**     radius-x                 **
**     radius-y                 **
**********************************

**********************************
** PROC FILL(x,y)               ** Udfylder lukket område
**   Parametre                  ** omkring (x,y)
**     x-koordinat              **
**     y-koordinat              **
**********************************

**********************************
** FUNC GETCOLOR    (gc(x,y))   ** Returnerer, hvad der er
**   Parametre                  ** skrevet i (x,y)
**     x-koordinat              **
**     y-koordinat              **
**********************************

**********************************
** PROC GRAPHICSSCREEN  (gs(x))** Vælger grafikskærm
**   Parametre                  **
**     skærm-nummer (1 eller 2)**
**********************************
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

```
**********************************
** PROC MOVE(x,y)                ** Flytter pen uden at
**   Parametre                   ** skrive
**     relativ x-koordinat       **
**     relativ y-koordinat       **
**********************************

**********************************
** PROC MOVETO(x,y)              ** Flytter pen uden at
**   Parametre                   ** skrive
**     x-koordinat               **
**     y-koordinat               **
**********************************

**********************************
** PROC PENCOLOR   (pc(x))      ** Vælger pentype:
**   Parametre                   ** 0 --> skriver
**     farve       (0 - 3)       ** 1 --> sletter
**********************************  2 --> revers skrift

**********************************
** PROC PIE(x,y,r,sv,v)          ** Tegner lagkagestykke
**   Parametre                   **
**     centrum-x                 **
**     centrum-y                 **
**     radius                    **
**     startvinkel (0 - 360)     **
**     vinkel      (0 - 360)     **
**********************************

**********************************
** PROC PLOT(x,y)                ** Sætter et punkt
**   Parametre                   **
**     x-koordinat               **
**     y-koordinat               **
**********************************

**********************************
** PROC PLOTTEXT("XXX",x,y)      ** Skriver en tekst fra
**   Parametre                   ** (x,y).
**     tekst der skal skrives    **
**     x-koordinat               **
**     y-koordinat               **
**********************************

**********************************
** PROC POLYGON(x,y,s,a,v)       ** Tegner regulær polygon med
**   Parametre                   ** centrum i (x,y).
**     x-koordinat               **
**     y-koordinat               **
**     sidelængde                **
**     antal sider (3 - n)       **
**     vinkel      (0 - 360)     **
**********************************
```

```
**********************************
** PROC TEXTSTYLE                ** Vælger teksttype og
**   Parametre                   ** skriftretning
**     x-multiplikation          **
**     y-multiplikation          **
**     retningsindikator(0 - 3)  **
**********************************

**********************************
** FUNC XCOR                     ** Returnerer aktuel X-
**********************************  koordinat.

**********************************
** FUNC YCOR                     ** Returnerer aktuel Y-
**********************************  koordinat.
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

APPENDIX F : OVERSIGT OVER FUNKTIONSTASTERNES ANVENDELSE

OVERLÆG TIL FUNKTIONSTASTERNE --->

I stedet for at skrive nogle af de almindeligste COMAL-80/Z80-kommandoer, hver gang man skal bruge dem, er der mulighed for at få dem indtastet ved at bruge funktionstasterne øverst på tastaturet.

Med COMAL-80/Z80 leveres et overlæg (se billedet side 180), som kan lægges ned omkring funktionstasterne øverst på tastaturet.

De kommandoer, som er angivet på overlægget, kan aktiveres v. hj. a. funktionstasterne.

Hvis man vil aktivere de kommandoer, som er angivet over funktionstasterne, skal man samtidigt trykke på SHIFT-knappen (Upper-case), medens kommandoerne under tasterne (Lower-case) blot aktiveres ved at trykke på funktionstasten.

Kommandoer kan kun afgives, når Comal er i kommando-modus, og det viser systemet ved, at cursoren blinker på skærmen.

Hvis man er i tvivl om kommandoernes betydning, kan man slå op i manualens alfabetiske kommandooversigt.

Man kan også på en lidt mere besværlig måde aktivere disse funktioner, nemlig ved at trykke på CONTROL-tasten til venstre på tastaturet og SAMTIDIGT trykke på en anden tast. Anvendelser af kontroltast som funktionstast skrives her ved hjælp af tegnet ^. (^A betyder altså, at man skal aktivere CONTROL-tast og A-tast samtidigt).

OVERSIGT OVER CONTROL-TASTERNES BETYDNING:

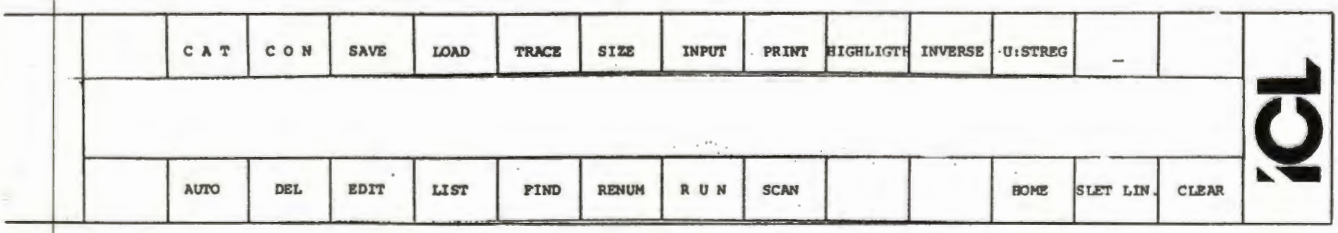
LOWER-CASE		UPPER-CASE	
^"	AUTO	^S	CAT
^A	DEL	^T	CON
^B	EDIT	^U	SAVE"
^C	LIST	^V	LOAD"
^D	FIND"	^W	TRACE
^E	REN	^X	SIZE
^F	RUN	^Y	INPUT

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

^G	SCAN	^Z	PRINT
^N		^Ø	DOBB.LYS
^O		^Å	INVERS SKRIFT
^P	CURSOR HOME	^^^	UNDERSTREG
^Q	DEL LINIE		
^R	PAGE		

BEM.: AUTO-funktionstasten og naturligvis også DOBB. LYS-, INVERS SKRIFT-, og UNDERSTREGNINGS-tast virker kun på 4 MHz-maskiner.



International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 62 75 88

APPENDIX G: MINITEKST

Følgende programliste, er et lille teksstbehandlingssystem, der viser, hvordan man bruger deltekste og meget andet i COMAL-80. Brugeren kan naturligvis også udvide programmmet, hvis dette er ønskeligt.

Under kørsel af programmet er det muligt at trykke 'H' tasten hvis man ønsker hjælp. Man vil i dette tilfælde få en liste over mulige jobs.

```
0005 // MINITEXT
0010 TRAP ESC -
0020 initialiser
0030 minitekst
0040
0050 PROC initialiser
0060   PAGE
0070   PRINT "Job :",AT 1,20: "Pos="
0080   PRINT AT 3,1: "="*80
0090   maxside:=FREE DIV 1680-2
0100   DIM førside(0:9),førlinie(0:9)
0110   DIM side$(maxside) OF 1680
0120   FOR i:=1 TO maxside DO
0130     side$(i)(:1680):=EMPTY$
0140   ENDFOR i
0150   side:=1; tekst'gemt:=TRUE
0160   indryk:=8; linier:=70
0170   find$:=EMPTY$
0180   OPEN FILE 0,"kb:",READ
0190 ENDPROC initialiser
0200
0210 PROC minitekst
0220   LOOP
0230     PRINT AT 1,7: "Kommando"
0240     PRINT AT 4,1: side$(side),
0250     PRINT AT 1,24: side,""19""
0260     PRINT "Kommando : "19"",
0270     REPEAT
0280       a$:=LOWER$(GET$(0,1))
0290     UNTIL a$ IN "efhlsp"27""8""12""
0300     CASE a$ OF
0310     WHEN "e"
0320       minitekst'editor
0330     WHEN "f"
0340       definer'format
0350     WHEN "h"
0360       PRINT AT 2,1: "Edit Format Help Load Print
           Save "19"",
0370       WHILE GET$(0,1)<>""27"" DO NULL
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

```
0380      WHEN "l"
0390        load'tekst
0400      WHEN "p"
0410        print'tekst
0420      WHEN "s"
0430        save'tekst
0440      WHEN ""8""
0450        IF side>1 THEN side:-1
0460      WHEN ""12""
0470        IF side<maxside THEN side:+1
0480      OTHERWISE
0490        IF NOT tekst'gemt THEN
0500          PRINT AT 2,1: "Skal teksten gemmes først ?
              (J/N) "19"",
0510          REPEAT
0520            a$:=LOWER$(GET$(0,1))
0530          UNTIL a$ IN "jn"
0540          IF a$="j" THEN save'tekst
0550        ENDIF
0560        PAGE
0570        CLOSE
0580        END "Slut på minitekst"
0590      ENDCASE
0600    ENDLOOP
0610  ENDPROC minitekst
0620
0630  PROC minitekst'editor
0640    PRINT AT 1,7: "Edit     "
0650    PRINT ""19""
0660    PRINT
0670    tekst'gemt:=FALSE
0680    LOOP
0690      x:=CURPOS; y:=CURLIN
0700      PRINT AT 1,24: side,":",y-3,":",x,""19""
0710      CURSOR y,x
0720      a$:=GET$(0,1)
0730      pos:=80*CURLIN+CURPOS-320
0740      rest:=80-CURPOS
0750      CASE ORD(a$) OF
0760      WHEN 8
0770        IF CURLIN>4 OR CURPOS>1 THEN
0780          PRINT ""5"",
0790        ENDIF
0800      WHEN 12
0810        IF CURLIN<24 OR CURPOS<80 THEN
0820          PRINT ""6"",
0830        ENDIF
0840      WHEN 10
0850        IF CURLIN<24 THEN PRINT ""10"",
0860      WHEN 11
0870        IF CURLIN>4 THEN PRINT ""11"",
0880      WHEN 18
0890        PRINT AT 2,1: "Skal siden slettes ? (J/N) "19"",
0900        REPEAT
0910          a$:=LOWER$(GET$(0,1))
```

```
0920        UNTIL a$ IN "jn"
0930        IF a$="j" THEN
0940          side$(side)(:1680):=EMPTY$
0950          x:=1; y:=4
0960          PRINT AT 4,1: side$(side),
0970        ENDIF
0980        PRINT AT 2,1: ""19"",
0990        CURSOR y,x
1000      WHEN 13
1010        IF CURLIN<24 THEN
1020          PRINT
1030        ELIF CURLIN=24 THEN
1040          PRINT ""13"",
1050        ENDIF
1060      WHEN 31
1070        side$(side)(pos:pos+rest):=" "+side$(side)(pos:
            pos+rest-1)
1080        PRINT side$(side)(pos:pos+rest),""5""*(rest+1),
1090      WHEN 127
1100        side$(side)(pos:pos+rest):=side$(side)(pos+1:
            pos+rest)
1110        PRINT side$(side)(pos:pos+rest),""5""*(rest+1),
1120      WHEN 27
1130        EXIT
1140      WHEN 16
1150        CURSOR 4,1
1160      WHEN 17
1170        side$(side)(pos:pos+rest):=EMPTY$
1180        PRINT SPC$(rest+1),""5""*(rest+1),
1190      WHEN 14
1200        INPUT AT 2,1,20: "Find teksten "19"": find$
1210        PRINT AT 2,1: ""19""
1220        found:=find$ IN side$(side)(pos+1:)
1230        IF found THEN
1240          found:+pos
1250          y:=found DIV 80+4
1260          x:=found MOD 80
1270        ENDIF
1280        CURSOR y,x
1290      WHEN 15
1300        found:=find$ IN side$(side)(pos+1:)
1310        IF found THEN
1320          found:+pos
1330          y:=found DIV 80+4
1340          x:=found MOD 80
1350        ENDIF
1360        CURSOR y,x
1370      OTHERWISE
1380        IF a$>""31"" AND a$<""128"" THEN
1390        side$(side)(pos):=a$
1400        PRINT a$,
1410        IF CURLIN=25 THEN PRINT ""11"",
1420        ENDIF
1430      ENDCASE
1440    ENDLOOP
```

```
1450 ENDPROC minitekst'editor
1460
1470 PROC definer'format
1480   PROC hentkoder(REF kode(),t$)
1490     PRINT AT 2,1: "Antal kontrol koder før hver";t$;"?
         "19"",
1500     REPEAT
1510       a$:=GET$(0,1)
1520     UNTIL a$ IN "0123456789"
1530     kode(0):=VAL(a$)
1540     FOR i:=1 TO VAL(a$) DO
1550       PRINT AT 2,1: "Kode nr.";i;": "19"",
1560       kode(i):=ORD(GET$(0,1))
1570     ENDFOR i
1580   ENDPROC hentkoder
1590   PRINT AT 2,1: "Indrykning på printer ? (0-9) "19"",
1600   REPEAT
1610     a$:=GET$(0,1)
1620   UNTIL a$ IN "0123456789"
1630   indryk:=VAL(a$)
1640   PRINT AT 2,1: "Antal linier pr. side ? (10-99) "19"",
1650   REPEAT
1660     a$:=GET$(0,1)
1670   UNTIL a$ IN "123456789"
1680   linier:=10*VAL(a$)
1690   PRINT a$,
1700   REPEAT
1710     a$:=GET$(0,1)
1720   UNTIL a$ IN "0123456789"
1730   linier:+VAL(a$)
1740   hentkoder(førside(),"side")
1750   hentkoder(førlinie(),"linie")
1760   PRINT AT 2,1: ""19""
1770 ENDPROC definer'format
1780
1790 PROC load'tekst
1800   INPUT AT 2,1,8: "Load teksten med navnet "19"": navn$
1810   FOR i:=1 TO maxside DO
1820     side$(i)(:1680):=EMPTY$
1830   ENDFOR i
1840   nr:=0
1850   DS -
1860   OPEN FILE 1,navn$,READ
1870   WHILE NOT EOF(1) DO
1880     INPUT FILE 1: l$
1890     side$(nr DIV 20+1)(nr MOD 20*80+1:(nr MOD 20+1)*80):
         =l$
1900     nr:+1
1910   ENDWHILE
1920   DS +
1930   CLOSE FILE 1
1940   tekst'gemt:=TRUE
1950 ENDPROC load'tekst
1960
1970 PROC print'tekst
```

```
1980   OPEN FILE 1,"lp:",WRITE
1990   nr:=0
2000   FOR i:=1 TO sidste'linie DO
2010     IF nr=0 THEN
2020       FOR i:=1 TO førside(0) DO
2030         PUT FILE 1: CHR$(førside(i))
2040       ENDFOR i
2050     ENDIF
2060     FOR i:=1 TO førlinie(0) DO
2070       PUT FILE 1: CHR$(førlinie(i))
2080     ENDFOR i
2090     FOR i:=1 TO indryk DO
2100       PUT FILE 1: " "
2110     ENDFOR i
2120     PRINT FILE 1: lin$(i)
2130     nr:+1
2140     IF nr=linier AND linier<>0 THEN
2150       PUT FILE 1: ""12""
2160       nr:=0
2170     ENDIF
2180   ENDFOR i
2190   CLOSE FILE 1
2200 ENDPROC print'tekst
2210
2220 PROC save'tekst
2230   INPUT AT 2,1,8: "Save teksten under navnet "19"":
       navn$
2240   IF navn$=EMPTY$ THEN RETURN
2250   tekst'gemt:=TRUE
2260   OPEN FILE 1,navn$,WRITE
2270   FOR i:=1 TO sidste'linie DO
2280     PRINT FILE 1: lin$(i)
2290   ENDFOR i
2300   CLOSE FILE 1
2310 ENDPROC save'tekst
2320
2330 FUNC lin$(linie)
2340   s:=(linie+19) DIV 20
2350   p:=(linie-1) MOD 20*80+1
2360   l$:=side$(s)(p:p+79)
2370   IF l$=SPC$(80) THEN RETURN EMPTY$
2380   FOR i:=80 DOWNTO 1 DO
2390     IF l$(i)<>" " THEN RETURN l$(:i)
2400   ENDFOR i
2410 ENDFUNC lin$
2420
2430 FUNC sidste'linie
2440   FOR i:=maxside*20 DOWNTO 1 DO
2450     IF lin$(i)<>EMPTY$ THEN RETURN i
2460   ENDFOR i
2470   RETURN 0
2480 ENDFUNC sidste'linie
```

International Computers Limited a/s

Hovedkontor: Gladsaxevej 372, DK 2860 Søborg. Tel. 01 - 67 97 00
Telex 39147 iclcph dk. Telegram Computel, København. Reg.nr. 40572. Giro 5 45 30 11
Jylland og Fyn: Romancevej 9, DK 8700 Horsens. Tel. 05 - 64 75 88

ØVRIGE DEMO-PROGRAMMER PÅ COMAL-80/Z80 DISKETTEN.

På den leverede COMAL-80/Z80 diskette ligger ud over de i manualen nævnte programmer også to demonstrationsprogerammer udarbejdet af Freddy Dan Dalgas Kristiansen:

1. DEMO.SAV. Et grafikdemonstrationsprogram, som kan give nogle ideer om hvilke muligheder, der ligger i grafikken.

2. ATOM.LST. Et simuleringsprogram, som på forenklet form simulerer styring af et atomkraftværk.

a.c.-Tryk ApS, Kbhvn.