
BICP
COMAL
EDB
for
Laboranter
Elsebeth
STRAADT
1984
TLF. 02-45 15 16

ELB-KURSUS

Betjening af udstyr.

##

```
0001 //  1: Tænd for udstyret.
0002 //  2: Indsæt disketten i Floppy-disken (uden vold).
0003 //  3: Indlæs CP/M-operativsystemet i primærlageret.
0004 //  4: Udskriv diskettens katalog på skærmen.  DIR
0005 //  5: Indlæs COMAL-80 i primærlageret.
0006 //  6: Udskriv diskettens katalog på skærmen.  CAT
0007 //  7: Slet skærmbilledet med een ordre.  CLEAR
0008 //  8: Aktiver den automatiske linienummerering (AUTO 40,10).
0009 //  9: Indtast nedenstående program.
0010 // 10: Afbryd den automatiske linienummerering.  ESCAPE
0011 // 11: Kør programmet.
0012 // 12: Indfør ordren PRINT mellem linie 100 og 110.
0013 // 13: Kør programmet. Hvilken effekt havde ændringen?
0014 // 14: Overfør programmet til baggrundslageret (disketten).SAVE
0015 // 15: Slet programmet fra primærlageret.
0016 // 16: Kontroller at programmet er fjernet fra primærlageret.(=arbejdslageret)
0017 // 17: Udskriv diskettens katalog på skærmen.
0018 // 18: Overfør programmet fra disketten til primærlageret.LOAD ("NAVN")
0019 // 19: Udskriv programmet på skærmen.  LIST   KUN LIST?
0020 // 20: Slet den før indførte PRINT ordre.  DEL 75,76
0021 // 21: Ret tallet 0.96 til 1.04 i linie 60, og fjern tegnet ";"  EDIT Linje
0022 //     i linie 120.
0023 // 22: Kør programmet. Hvilken effekt havde ændringen?  120 og 130 skelles på 2 linje
0024 // 23: Overfør programmet til baggrundslageret under samme navn som før;
0025 //     bemærk datamatens reaktion og tryk <RETURN>!!
0026 // 24: Indfør som linie 35 ordren: SELECT OUTPUT "LP:" ,og kør programmet
0027 // 25: Skriv LIST LP: , og bemærk virkningen!!  udskrives
0028 // 26: Udtag disketten (uden vold) og sluk for udstyret.
0029 //
0030 //   G O D   F O R N Ø J E L S E . Leon
0031 //----------------------------------------------------------------------
0032 //
0040 READ P, V, T
0050 IF T=0 THEN GOTO 0120
0060 DATA 1.2, 25.4, 293.0, 0.96, 28.0
0070 N:=P*V/(0.082*T)
0080 DATA 289.0, 0, 0, 0
0090 PRINT "Der er ";N;" gmol af luftarten."
0100 PRINT "Trykket = ";P,"  Rumfanget = ";V,"  Temperaturen = ";T
0110 GOTO 0040
0120 PRINT "D e t  var hurtigt hva"!!";
0130 PRINT "....kan du gøre det hurtigere??"
0140 END
```



LABORANTSKOLEN
Rådmandsgade

20/2-84

DELTAGERLISTE

FOR

KURSUS I EDB FOR LABORANTER 7. - 16. marts 1984

Andersen, Inge-Dorte	Cheminova	o7-834100
Andersen, Mette Pauli	Statens Seruminst.	o1-952817
Christensen, Anders B.	De Danske Sukkerfa.	o1-954100/36
Fischer, Vibeke	Retsmed.inst.	o1-373222/334
Friis, Vibeke	Novo	o2-982333
Grindsted, Vivian	Haldor Topsøe	o2-878100
Hansen, Karen Margrethe	Danm.Farm.Højskole	o1-37o85o/286
Elisabeth N. Hjarding	F.L. Smidth & Co.	o1-301166
Honig, Annelise	Zoofysiol.lab.	o1-377000
Jørgensen, Per	-	o2-643637
Kiersz, Aleksander	Jens Villadsens Fa.	o1-502433/23
Kreisby, Solveig	Stat.Seruminst.	o1-952817/2566
Larsen, Poul Erik	Inst.f.med.mikrobiologi	o1-391033
Nielsen, Anette H.	Nunc	o2-359065
Nørringgård, Eva	G.G.U.	o1-118866
Petersen, Lars Boje	Haldor Topsøe	o2-878100/238
Raskmark, Jørgen	D.T.H.	o2-884144/3646
Schøneberg, Astrid	D.T.H.	o2-881444/2123
Straadt, Elsebeth	Dumex	o1-544500
Sørensen, Marianne B.	Landbohøjskolen	o1-351788/2667
Venneberg, Leif	Cheminova	o7-834100
zum Vohrde, Kirsten	-	o1-180003

12/384 EG

```
0010 //  PROGRAM OPRET
0020 //
0030 //  Copyright Leon Bennicke  19-OKT-1983
0040 //
0050 //  Program til oprettelse af medarbejderposter i filen KARTOTEK.OPL
0060 //  Filen er af typen RANDOM og har en postlængde på 80 byte fordelt
0070 //  på følgende måde:
0080 //
0090 //  ------------------------------------------------------------------
0100 //  : STATUS# : MNR# : NAVN$ : LKAT$ : SPCT# : FRADRAG : REG# : KONTONR$ :
0110 //  ------------------------------------------------------------------
0120 //
0130 //  Hvor
0140 //
0150 //    STATUS#   ( 2 byte) = Poststatus (0=oprettet, 1=slettet).
0160 //    MNR#      ( 2 byte) = Medarbejdernr (0..999).
0170 //    NAVN$     (42 byte) = Navn (max 40 tegn).
0180 //    LKAT$     (14 byte) = Lønkategori (TIMELØNNET,UGELØNNET,MÅNEDSLØNNET)
0190 //    SPCT#     ( 2 byte) = Skatteprocenten (0..100).
0200 //    FRADRAG   ( 4 byte) = Fradrag før skat (0..10000).
0210 //    REG#      ( 2 byte) = Registreringsnr. for pengeinstitut.
0220 //    KONTONR$ (12 byte) = Kontonr. i pengeinstitut (max 10 tegn).
0230 //
0240 //  Bemærk at tekstvariable (f.eks. NAVN$) fylder 2 byte mere end det
0250 //  antal tegn den maximalt kan indeholde !
0260 //
0270 // : Erklæringer og startværdier.
0280 //
0290 DIM NAVN$ OF 40, LKAT$ OF 12, KONTONR$ OF 10, TL$ OF 12, UL$ OF 12
0300 DIM ML$ OF 12, SVAR$ OF 1
0310 //
0320 OPRET#:=0; CONTINUE#:=TRUE
0330 READ TL$, UL$, ML$
0340 DATA "TIMELØNNET", "UGELØNNET", "MÅNEDSLØNNET"
0350 //
0360 // : Åbning af filen KARTOTEK.OPL
0370 //
0380 OPEN FILE 0, "KARTOTEK.OPL", RANDOM , 80
0390 READ FILE 0, 1: MAX#, ANTAL#
0400 ANT#:=ANTAL#
0410 //
0420 // : Indlæsning af medarbejder-oplysninger.
0430 //
0440 REPEAT
0450   IF ANTAL#=MAX# THEN
0460     PRINT "Medarbejder-kartotek fyldt op. Sorry ! Call manager !"
0470     CONTINUE:=FALSE
0480   ELSE
0490     REPEAT
0500       INPUT "Medarbejdernr. (0..999):": MNR#
0510     UNTIL MNR#)=0 AND MNR#(=999
0520     PRINT "                          1---------10---------20---------30--------40"
0530     INPUT "Navn (max 40 tegn):": NAVN$
0540     REPEAT
0550       INPUT "Kategori (TIMELØNNET,UGELØNNET,MÅNEDSLØNNET):": LKAT$
0560     UNTIL TL$ IN LKAT$ OR UL$ IN LKAT$ OR ML$ IN LKAT$
0570     REPEAT
0580       INPUT "Skatteprocent (0..100):": SPCT#
0590     UNTIL SPCT#)=0 AND SPCT#(=100
0600     REPEAT
0610       INPUT "Fradrag (0..10000):": FRADRAG
0620     UNTIL FRADRAG)=0 AND FRADRAG(=10000
0630     REPEAT
0640       INPUT "Pengeinstitut (registreringsnr. 0000..9999):": REG#
0650     UNTIL REG#)=0 AND REG#(=9999
```

```
0660      PRINT "                              1---5---10"
0670      INPUT "Kontonr (max 10 tegn):": KONTONR$
0680      //
0690      // : Kontrolforanstaltninger.
0700      //
0710      IDX:=1
0720      REPEAT // Find 1. ledige post
0730        IDX:+1
0740        READ FILE 0, IDX: STATUS#
0750      UNTIL STATUS#=1
0760      I:=1
0770      REPEAT // Eksisterer medarbejdernr i forvejen ?
0780        I:+1
0790        READ FILE 0, I: STATUS#, MB#
0800      UNTIL MB#=MNR# AND STATUS#=0 OR I=MAX#+1
0810      IF MB#=MNR# AND STATUS#=0 THEN
0820        PRINT "Medarbejdernr ";MNR#;" eksisterer i forvejen ! Prøv et andet
            nr."
0830      ELSE
0840        WRITE FILE 0, IDX: OPRET#, MNR#, NAVN$, LKAT$, SPCT#, FRADRAG, REG#,
            KONTONR$
0850        ANTAL#:+1
0860        WRITE FILE 0, 1: MAX#, ANTAL#
0870      ENDIF
0880    ENDIF
0890    IF CONTINUE# THEN
0900      INPUT "Ønskes flere oprettelser (<RETURN>/NEJ):": SVAR$
0910    ENDIF
0920  UNTIL NOT CONTINUE# OR SVAR$<>""
0930  CLEAR
0950  PRINT ANTAL#-ANT#;" Oprettelse(r) foretaget."
0960  CLOSE FILE 0
0970  CHAIN "TRANSAK"
0980  END
```

```
0010 //  PROGRAM ÆNDRE
0020 //
0030 //  Copyright Leon Bennicke  19-OKT-1983
0040 //
0050 //  Program til ændring af medarbejderposter i filen KARTOTEK.OPL .
0060 //  Filen er af typen RANDOM og har en postlængde på 80 byte fordelt
0070 //  på følgende måde:
0080 //
0090 //  ---------------------------------------------------------------------
0100 //  : STATUS# : MNR# : NAVN$ : LKAT$ : SPCT# : FRADRAG : REG# : KONTONR$ :
0110 //  ---------------------------------------------------------------------
0120 //
0130 //  Hvor
0140 //
0150 //    STATUS#   ( 2 byte) = Poststatus (0=oprettet, 1=slettet).
0160 //    MNR#      ( 2 byte) = Medarbejdernr (0..999).
0170 //    NAVN$     (42 byte) = Navn (max 40 tegn).
0180 //    LKAT$     (14 byte) = Lønkategori (TIMELØNNET,UGELØNNET,MANEDSLØNNET)
0190 //    SPCT#     ( 2 byte) = Skatteprocenten (0..100).
0200 //    FRADRAG   ( 4 byte) = Fradrag før skat (0..10000).
0210 //    REG#      ( 2 byte) = Registreringsnr. for pengeinstitut.
0220 //    KONTONR$  (12 byte) = Kontonr. i pengeinstitut (max 10 tegn).
0230 //
0240 //  Bemærk at tekstvariable (f.eks. NAVN$) fylder 2 byte mere end det
0250 //  antal tegn den maximalt kan indeholde !
0260 //
0270 // : Erklæringer og startværdier.
0280 //
0290 DIM NAVN$ OF 40, LKAT$ OF 12, KONTONR$ OF 10, TL$ OF 12, UL$ OF 12
0300 DIM ML$ OF 12, SVAR$ OF 1
0310 //
0320 //
0330 // : Udskrivnings-procedure
0340 //
0350 PROC UDSKRIV
0360   PRINT "MEDARBEJDERNR: ";MNR#
0370   PRINT "NAVN           : ";NAVN$
0380   PRINT "LØNKATEGORI    : ";LKAT$
0390   PRINT "SKATTEPROCENT: ";SPCT#
0400   PRINT "FRADRAG        : ";FRADRAG
0410   PRINT "PENGEINSTITUT: ";REG#
0420   PRINT "KONTONUMMER    : ";KONTONR$
0430   PRINT
0440 ENDPROC UDSKRIV
0450 //
0460 READ TL$, UL$, ML$
0470 DATA "TIMELØNNET", "UGELØNNET", "MANEDSLØNNET"
0480 //
0490 // : Åbning af filen KARTOTEK.OPL
0500 //
0510 OPEN FILE 0, "KARTOTEK.OPL", RANDOM , 80
0520 READ FILE 0, 1: MAX#, ANTAL#
0530 ANT#:=0
```

```
0540 //
0550 // : Indlæsning af medarbejder-oplysninger.
0560 //
0570 REPEAT
0580   CLEAR
0590   INPUT "Indtast medarbejdernr for ændring: ": MNR#
0600   IDX#:=1
0610   REPEAT // Eksisterer medarbejdernr ?
0620     IDX#:+1
0630     READ FILE 0, IDX#: STATUS#, MB#, NAVN$, LKAT$, SPCT#, FRADRAG, REG#,
         KONTONR$
0640   UNTIL MB#=MNR# OR IDX#=MAX#+1
0650   IF MB#=MNR# AND STATUS#=0 THEN
0660     REPEAT
0670       CLEAR
0680       PRINT "NUVÆRENDE KARTOTEKSINDHOLD:"
0690       PRINT
0700       EXEC UDSKRIV
0710       PRINT
0720       PRINT "Hvad ønskes ændret:"
0730       PRINT "   0: INGEN ÆNDRING."
0740       PRINT "   1: Medarbejdernr."
0750       PRINT "   2: Navn."
0760       PRINT "   3: Lønkategori."
0770       PRINT "   4: Skatteprocent."
0780       PRINT "   5: Fradrag."
0790       PRINT "   6: Pengeinstitut (registreringsnr.)."
0800       PRINT "   7: Kontonummer (lønkonto)."
0810       PRINT "  99: IKKE FLERE ÆNDRINGER."
0820       PRINT
0830       REPEAT
0840         INPUT "Indtast nummer for ændring (0..7,99): ": FELTNR#
0850       UNTIL FELTNR#>=0 AND FELTNR#<8 OR FELTNR#=99
0860       CASE FELTNR# OF
0870       WHEN 1
0880         HMNR#:=MNR#
0890         REPEAT
0900           INPUT "Medarbejdernr. (0..999):": MNR#
0910         UNTIL MNR#>=0 AND MNR#<=999
0920         I#:=1
0930         REPEAT
0940           I#:+1
0950           READ FILE 0, I#: STAT#, MEDL#
0960         UNTIL MNR#=MEDL# AND STAT#=0 OR I#=MAX#+1
0970         IF MNR#=MEDL# AND STAT#=0 AND I#<>IDX# THEN
0980           PRINT "Medarbejdernr ";MNR#;" Findes i forvejen ! Prøv et andet
               Nr."
0990           MNR#:=HMNR#
1000         ENDIF
```

```
1010        WHEN 2
1020          PRINT "                    1---------10---------20---------30-------4
1030          INPUT "Navn (max 40 tegn):": NAVN$
1040        WHEN 3
1050          REPEAT
1060            INPUT "Kategori (TIMELØNNET,UGELØNNET,MÅNEDSLØNNET):": LKAT$
1070          UNTIL TL$ IN LKAT$ OR UL$ IN LKAT$ OR ML$ IN LKAT$
1080        WHEN 4
1090          REPEAT
1100            INPUT "Skatteprocent (0..100):": SPCT#
1110          UNTIL SPCT#>=0 AND SPCT#<=100
1120        WHEN 5
1130          REPEAT
1140            INPUT "Fradrag (0..10000):": FRADRAG
1150          UNTIL FRADRAG>=0 AND FRADRAG<=10000
1160        WHEN 6
1170          REPEAT
1180            INPUT "Pengeinstitut (registreringsnr. 0000..9999):": REG#
1190          UNTIL REG#>=0 AND REG#<=9999
1200        WHEN 7
1210          PRINT "                          1---5---10"
1220          INPUT "Kontonr (max 10 tegn):": KONTONR$
1230        WHEN 99
1240          ANT#:+1
1250          WRITE FILE 0, IDX#: STATUS#, MNR#, NAVN$, LKAT$, SPCT#, FRADRAG,
              REG#, KONTONR$
1260        OTHERWISE
1270          PRINT "INGEN ÆNDRINGER FORETAGET"
1280        ENDCASE
1290      UNTIL FELTNR#=0 OR FELTNR#=99
1300      //
1310    ELSE
1320      PRINT "Medarbejdernr ";MNR#;" er ikke at finde i Kartoteket !"
1330    ENDIF
1340    INPUT "Ønskes flere Poster ændret (<RETURN>/NEJ):": SVAR$
1350  UNTIL SVAR$<>""
1360  CLEAR
1380  PRINT ANT#;" Poste(r) ændret."
1390  CLOSE FILE 0
1400  CHAIN "TRANSAK"
1410  END
```

```
0010 //  PROGRAM SLET
0020 //
0030 //  Copyright Leon Bennicke 19-OKT-1983
0040 //
0050 //
0060 //  Program til sletning af medarbejderposter i filen KARTOTEK.OPL
0070 //  Filen er af typen RANDOM og har en postlængde på 80 byte fordelt
0080 //  på følgende måde:
0090 //
0100 //  ---------------------------------------------------------------------
0110 //  : STATUS# : MNR# : NAVN$ : LKAT$ : SPCT# : FRADRAG : REG# : KONTONR$ :
0120 //  ---------------------------------------------------------------------
0130 //
0140 //  Hvor
0150 //
0160 //    STATUS#   ( 2 byte) = Poststatus (0=oprettet, 1=slettet).
0170 //    MNR#      ( 2 byte) = Medarbejdernr (0..999).
0180 //    NAVN$     (42 byte) = Navn (max 40 tegn).
0190 //    LKAT$     (14 byte) = Lønkategori (TIMELØNNET,UGELØNNET,MANEDSLØNNET)
0200 //    SPCT#     ( 2 byte) = Skatteprocenten (0..100).
0210 //    FRADRAG   ( 4 byte) = Fradrag før skat (0..10000).
0220 //    REG#      ( 2 byte) = Registreringsnr. for pengeinstitut.
0230 //    KONTONR$ (12 byte) = Kontonr. i pengeinstitut (max 10 tegn).
0240 //
0250 //  Bemærk at tekstvariable (f.eks. NAVN$) fylder 2 byte mere end det
0260 //  antal tegn den maximalt kan indeholde !
0270 //
0280 // : Erklæringer og startværdier.
0290 //
0300 DIM NAVN$ OF 40, LKAT$ OF 12, KONTONR$ OF 10, UDEV$ OF 7
0310 DIM NS$ OF 40, SVAR$ OF 1
0320 SLETTET#:=1
0330 //
0340 // : Udskrivnings-procedure
0350 //
0360 PROC UDSKRIV
0370   PRINT "MEDARBEJDERNR: ";MNR#
0380   PRINT "NAVN         : ";NAVN$
0390   PRINT "LØNKATEGORI  : ";LKAT$
0400   PRINT "SKATTEPROCENT: ";SPCT#
0410   PRINT "FRADRAG      : ";FRADRAG
0420   PRINT "PENGEINSTITUT: ";REG#
0430   PRINT "KONTONUMMER  : ";KONTONR$
0440   PRINT
0450 ENDPROC UDSKRIV
0460 //
0470 // : Åbning af filen KARTOTEK.OPL
0480 //
0490 OPEN FILE 0, "KARTOTEK.OPL", RANDOM , 80
0500 READ FILE 0, 1: MAX#, ANTAL#
0510 ANT#:=ANTAL#
```

```
0520 //
0530 // : Udskrivning af medarbejder-oplysninger.
0540 //
0550 IF ANTAL#(=0 THEN
0560   PRINT "kartoteket er tomt. Sorry ! Programmet stopper. "
0570 ELSE
0580   REPEAT
0590     INPUT "Vælg Uddata-enhed (SKÆRM,PRINTER):": UDEV$
0600   UNTIL UDEV$="SKÆRM" OR UDEV$="PRINTER"
0610   IF UDEV$="SKÆRM" THEN
0620     UDEV$:="DS:"
0630   ELSE
0640     UDEV$:="LP:"
0650   ENDIF
0660   REPEAT
0670     SELECT OUTPUT UDEV$
0680     //
0690     REPEAT
0700       INPUT "Indtast Medlemsnummer til sletning: ": MEDL#
0710     UNTIL MEDL#)=0
0720     //
0730     I#:=1
0740     REPEAT
0750       I#:+1
0760       READ FILE 0, I#: STATUS#, MNR#, NAVN$, LKAT$, SPCT#, FRADRAG, REG#,
           KONTONR$
0770     UNTIL MNR#=MEDL# AND STATUS#=0 OR I#)=MAX#+1
0780     IF STATUS#=0 AND MNR#=MEDL# THEN
0790       PRINT "+-------------------------+"
0800       PRINT ":    MEDARBEJDERNR: ";MEDL#;TAB(27);": S L E T T E T"
0810       PRINT "+-------------------------+"
0820       EXEC UDSKRIV
0830       PRINT
0840       WRITE FILE 0, I#: SLETTET#, MEDL#
0850       ANTAL#:-1
0860     ELSE
0870       SELECT OUTPUT "DS:"
0880       PRINT "Medlemsnr ";MEDL#;" er ikke at finde i Kartoteket !"
0890     ENDIF
0900     //
0910     SELECT OUTPUT "DS:"
0920     INPUT "Ønskes flere Sletninger foretaget (<RETURN)/NEJ):": SVAR$
0930   UNTIL SVAR$<>""
0940   //
0950 ENDIF
0960 IF ANT#)ANTAL# THEN
0970   PRINT ANT#-ANTAL#;" Sletning(er) foretaget !"
0980   WRITE FILE 0, 1: MAX#, ANTAL#
0990 ENDIF
1010 CLOSE FILE 0
1020 CHAIN "TRANSAK"
1030 END
```

//LAV PROGRAM UDSKRIV 13/3 84

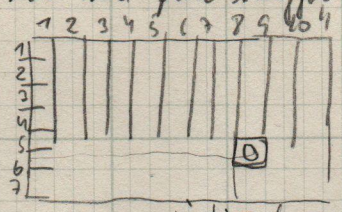
	sign/dato	side
	erstatter	projekt

4 typer variable

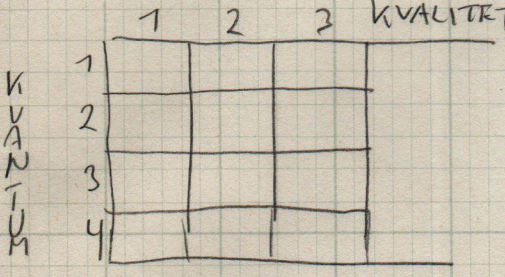
1) ET TAL TAL := 0 TAL := TAL + 1 eller SUM := 0 # heltal

2) ET TALSÆT - som # men med flere skuffer

eks JERN(6,9) := 10



indiceret variabel = talsæt



- FOR KVALITET# := 1 to 3 STEP 1 DO
- FOR KVANTUM# := 1 to 4 STEP 1 DO
- PRINT KVALITET#; KVANTUM#; VIN (KVALITET#, KVANTUM#)
- NEXT KVANTUM
- NEXT KVALITET

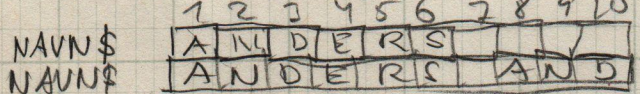
KVALITET KVANTUM PRIS

11 13 7 1
12 26 osv!
13
14
21
22
23
24
31
32
osv
:
34

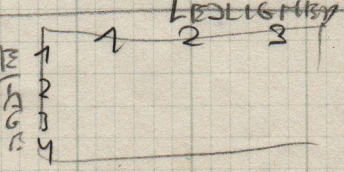
10 STRÅLING (17, 22, 10, 3) := 811 en skuffe

3) EN FØLGE AF TEGN kan indeholde en tekst - tekstvariabel efterfølges af $

STED$:= "Laboratorieskolen"
DIM NAVN$ OF 10
NAVN$:= "ANDERS"
NAVN$:= NAVN$ + " " + "AND" mellemrum
PRINT NAVN$ udskriver ANDERS AND



4) Et sæt af tegnfølger : (tekster)



DIM EJER$ (3,4) OF 10 (10 tegn i hver skuffe
PRINT "LEJLIGHED ETAGE EJER"
FOR LEJLIGHED# := 1 TO 3 STEP 1 DO
FOR ETAGE# := 1 TO 4 STEP 1 DO
PRINT LEJLIGHED#, ETAGE#, EJER$(LEJLIGHED#, ETAGE#)
NEXT ETAGE NEXT LEJLIGHED#

A

7/3 84

Basic er en del mængde af Comal-80

10 tals syst. 17 1 tier 7 enere $= 7 \times 10^0 + 1 \times 10^1$ se Appendix

1 0 1 3 ~~[illegible]~~

1 tour ~~[illegible]~~ 1 tie 3 ene

2 tals syst $17 = 16 + 0 + 0 + 0 + 1$

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$

$\underline{1} \times 2^4 + \underline{0} \times 2^3 + \underline{0} \times 2^2 + \underline{0} \times 2^1 + \underline{1} \times 2^0$

dvs 17 ~ 10001

opgave app B side B 1

4-1 a) $\boxed{A = A - 17}$ :: henter det der ligger i A - trækker 17 fra og lægger det til bage i A

a)

6	LDX, 1	Læg indholdet i skuffe 1 = A i X
1	MVI, 17	Læg 17 i I
3	MIS, 4	Læg 17 i 4
7	LDY, 4	Læg 17 i Y
9	SUB	trækk 17 (Y) fra A (X)
5	MRS, 1	læg resultet i skuff 1 = A

b) $B = -B + 3 \cdot C = 3 \cdot C - B$

1	MVI, 3	Læg 3 i I
3	MIS, 4	Læg 3 i 4
6	LDX, 4	Lægg 3 i X
7	LDY, 3	Læg indhold af skuff 3 = C i Y
10	MUL	gang 3 med Y
5	MRS, 5	læg result 3×C i 5
6	LDX, 5	skuffe 5 i X
7	LDY, 2	skuffe B = 2 i Y
9	SUB	
5	MRS 2	til bage i 2 = B

Binærtal

10^4	10^3	10^2	10^1	10^0
2	3	4	1	3

$2\times10^4 + 3\times10^3 + 4\times10^2 + 1\times10^1 + 3\times10^0 = 23413$

2^4	2^3	2^2	2^1	2^0
2	3	4	1	3
1	0	0	1	0

$1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

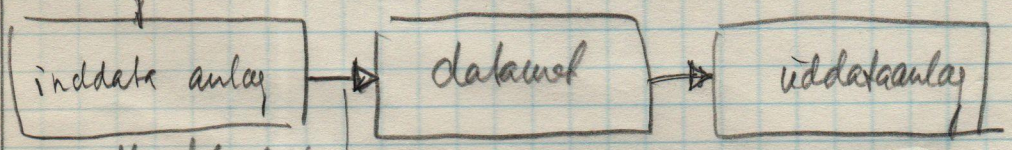
$16 + 0 + 0 + 2 + 0 = 18.$

$73 = 64 + 9$

~~128~~	64	32	16	8	4	2	1
~~0~~	1	0	0	1	0	0	1

$\rightarrow 64 + 8 + 1 = 73$

kapitel 5



omsætter tal og bogst til elektriske signaler

skal kunne læse ens dra og

værktøj: datamedier
fets hulkort

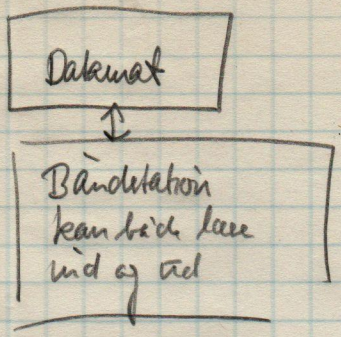
inddata enhed:
fets hulkortlæser
fotoceller

Hulkort + Hullestrimler

HULKORT
- plads til 80 tegn incl. mellemrum
- Billigt
- Fylder meget
- brand farl

kapitel 5 EDB 7/3 84 3

Magnetbånd



Datamat

↕

Båndstation
kan både læse
ind og ud

---

1 CPM indlæsning af diskette tryk på 1
2 Parallel
3 Monitor

Select : □

A> DIR [RETURN] får skrevet register - DIRectory vel
↑ GIV MIG en ordre

Startup program } ROM 1 2 3 Read only memory

skærmprogram
CP/M - styreprogram } RAM Control Program for Microprocessor
fehl comm transient program
område Random access? memory
Bruger programmer

LAGEROPDELING

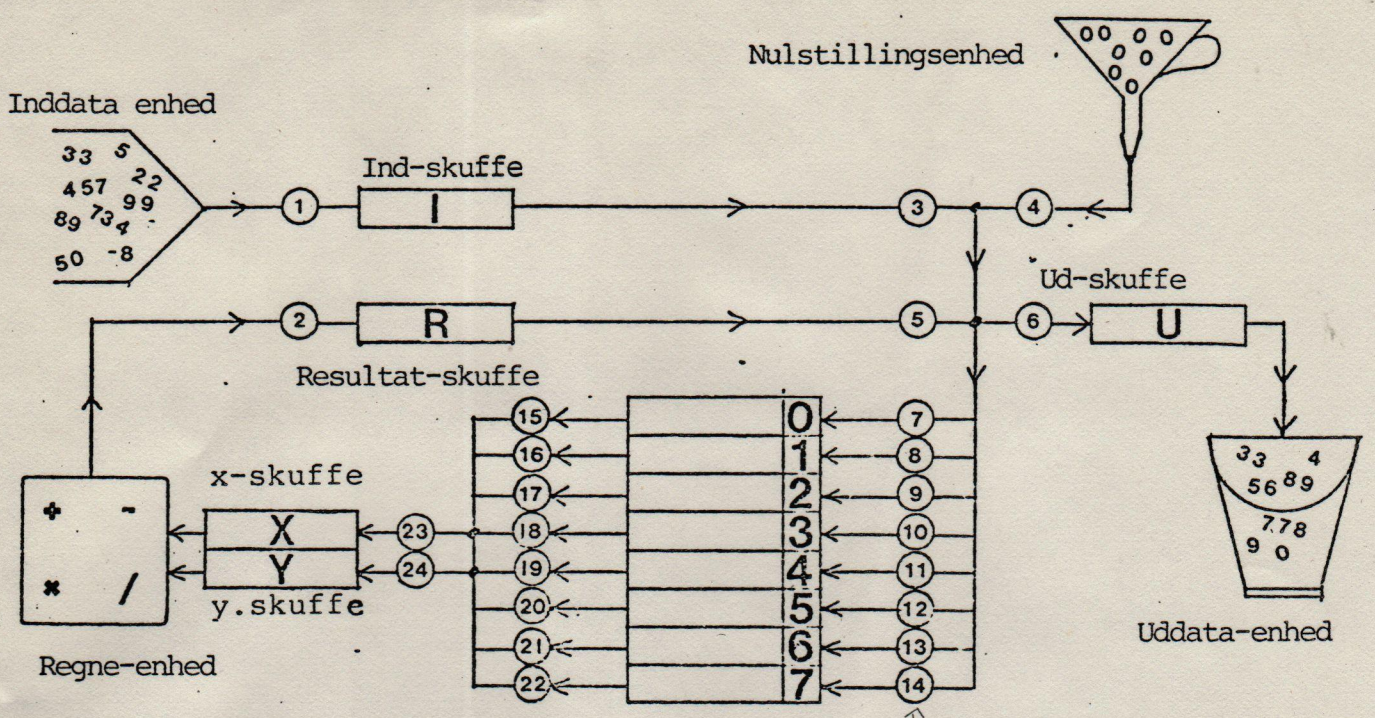
A> COMAL-80 [RETURN] et program på disketten hedder COMAL 80
intet mellemum
kun -

ønskes fejl tekst - hvis nej får kun fejl kode
svar J

* □ blinker invitation til at give ordre til comal-80 oversætteren
indende tryk en returner

* 10 PRINT "UK" □
---- [⇄] indtast K first [INS] "□UK"
indtast K "K[U]K" □
Return => "KUK" □

ORD - et antal bytes
en byte = 8 bit for 10010110
en bit = et ciffer i det binære syst 0 et 1



kan kun regne på 2 tal af gangen

porte som styreenhed åbner og lukker

Fig. 4.2a Datamatmodel.

Ordre	Decimalt	Binært	Betydning	Åbner port nr.
RESET	0	0000	læg 0 i skufferne 0-7	4, 7-14
MVI,T	1	0001	læg et tal T i skuffe I.	1
MVU	2	0010	læg indhold af I i skuffe U	3,6
MIS,N	3	0011	læg indhold af I i skuffe nr. N	3,N+7
MVR	4	0100	Læg indhold af R i skuffe U	5,6
MRS,N	5	0101	læg indhold af R i skuffe nr. N	5, N+7
LDX,N	6	0110	læg indhold af skuffe N i skuffe X	N +15, 23
LDY,N	7	0111	læg indhold af skuffe N i skuffe Y	N +15, 24
ADD	8	1000	læg skuffe x og y sammen i R	2
SUB	9	1001	Træk skuffe Y fra X i R	2
MUL	10	1010	Gang skuffe X med Y, i R	2
DIV	11	1011	Divider skuffe X med Y i R.	2

Fig. 4.2b ordresæt for datamatmodel.

10? A: = B×20 + 3 sætning i Cornal 80

eks 2. A = B×20 + 3 skuffe 0 kaldes A
" 1 " B
indhold af B placeret i X

eks læg 19 i skuffe 0 ∴ kopi af 19 i I skuffe

MVI, 19	1
MIS, 0	3

en variabel ~ en skuffe fra I

ADD	B×20 + 3	
5	MRS, 0	Result i skuffe 0 = A

	LDX, 1	X = B
1	MVI, 20	I = 20
3	MIS, 2	sk2 = I = 20
7	LDY, 2	Y = 20
10	MUL	skuffe R indeholder nu B×20
5	MRS, 3	skuffe 3 = B×20
6	LDX, 3	X = B×20
1	MVI, 3	I = 3
3	MIS, 4	skuffe 4 = 3
7	LDY, 4	Y = 3

Laborantskolen i Hovedstadsområdet

☐ Krimsvej 29
2300 København S
Telf.: 01 - 55 19 22

☒ Rådmandsgade 71
2200 København N
Telf.: 01 - 85 12 66

Dumex
att.: Laboratorietekniker
Elsebeth Straadt
Prags Boulevard 37
2300 S

20. februar 1984
KH/amc

Kursus i EDB for laboranter 7. - 16. marts 1984

Vi bekræfter hermed, at du er optaget på ovennævnte kursus.

Du bedes møde onsdag den 7. marts 1984, kl. 8,30.

Undervisningen foregår i laborantskolens lokaler i Rådmandsgade 71, 2200 København N, lokale 265, 2. sal. (Ved Iersø Parkalle/Tagensvej).

Skriverekvisitter og regnemaskine bedes medbragt.

Kompendie udleveres første dag.

Skolen råder over kantine, hvor man kan købe eller medbringe mad og drikkevarer.

./. Deltagerliste vedlægges.

Pris: kr. 1.700,-.

Med venlig hilsen
LABORANTSKOLEN
Kirsten Holm
Kirsten Holm
akademiingeniør

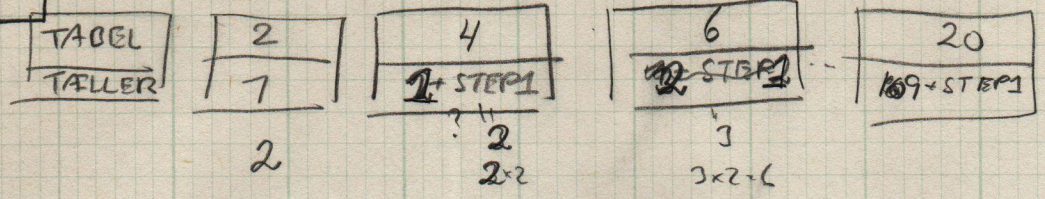
Bilag

Comal-80	sign/dato 8/3 84	side 1
	erstatter	projekt EDB

7.2 2-tabellen

Tabel tæller×2

TABEL	2	4	6	20
TÆLLER	1	1 + STEP1	~~2 STEP1~~	19 + STEP1
	2	? 2	3	
		2×2	3×2=6	



7.6 // Bemærkning
Til forklaring overfor en ede og andre - bliver ikke oversat men bliver bevaret

// ~ REM i basic

: = sættes hig mere
(noget puttes i en skuffe

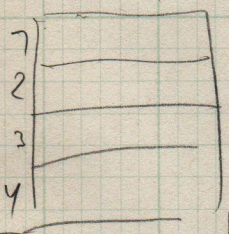
! bliver også til //

Reserverede Ord - App G

; adskillelse af printelementer tal variable tekst

alle ikke variabel navne skal i " "
∴ tekst sættes i "

; og, bruges ofte ensbetydende - her har de samme betydning
(ikke helt !!)



AK

1	
2	
3	
4	

variabl navn AK

10 AK(2) := 4.3
skuffe 2 i AK

7.7 varnar SUM heltal ↓ #

7.12 $ tekst var

7.15 R^3 R↑3 udskrives R^3

	sign/dato 8/3 84	side 2
	erstatter	projekt EDD

7.20

1: $(3+4*6)/9$ DIV $2*(4\uparrow 2+1)$

$(3+24)$

$27/9$ DIV $2*(4^2+1)$

$\boxed{27/9 \text{ DIV } 2*17}$

3 DIV $2\cdot 17$

$1\cdot 17 = \underline{\underline{17}}$

2: 17 MOD $(2\uparrow 3-19)*(16*(8/4-1))$

17 MOD $(2^3-19)*(16*1)$

~~17 MOD 8 − 19~~

$\underbrace{17 \text{ MOD } -11}$ $*$ 16

$\cancel{17}$ $5*16 = \underline{\underline{80}}$

17 MOD $(2^3-19)\cdot(16\cdot 1)$

17 MOD $(8-19)\cdot 16$

$\underbrace{17 \text{ MOD } -11}\cdot 16$

$6 \cdot 16 = \underline{\underline{96}}$

$2^0=1$
$2^1=2$
$2^2=4$
$2^3=8$
$16-5$
$0\,3$

~~$(17-11x)\ x=2$~~

~~$17-22=5$~~

$(17-11x)\ x=1$

$(17-11)=6$

(3) $\underbrace{29 \text{ MOD } 15}$ DIV 2 MOD 3

$\underbrace{4 \text{ DIV } 2}$ MOD 3

$\underbrace{7 \text{ MOD } 3}$

$\underline{\underline{1}}$

A

	sign/dato 8/3 84	side 3
	erstatter	projekt EDB

CPR nr. DD MM ÅÅ - XXX/X

MOD 11 på disse verificerer om det sidste ciffer er rigtig

IF "SEN" IN EDER$ (Len(EDER$)-3,3)

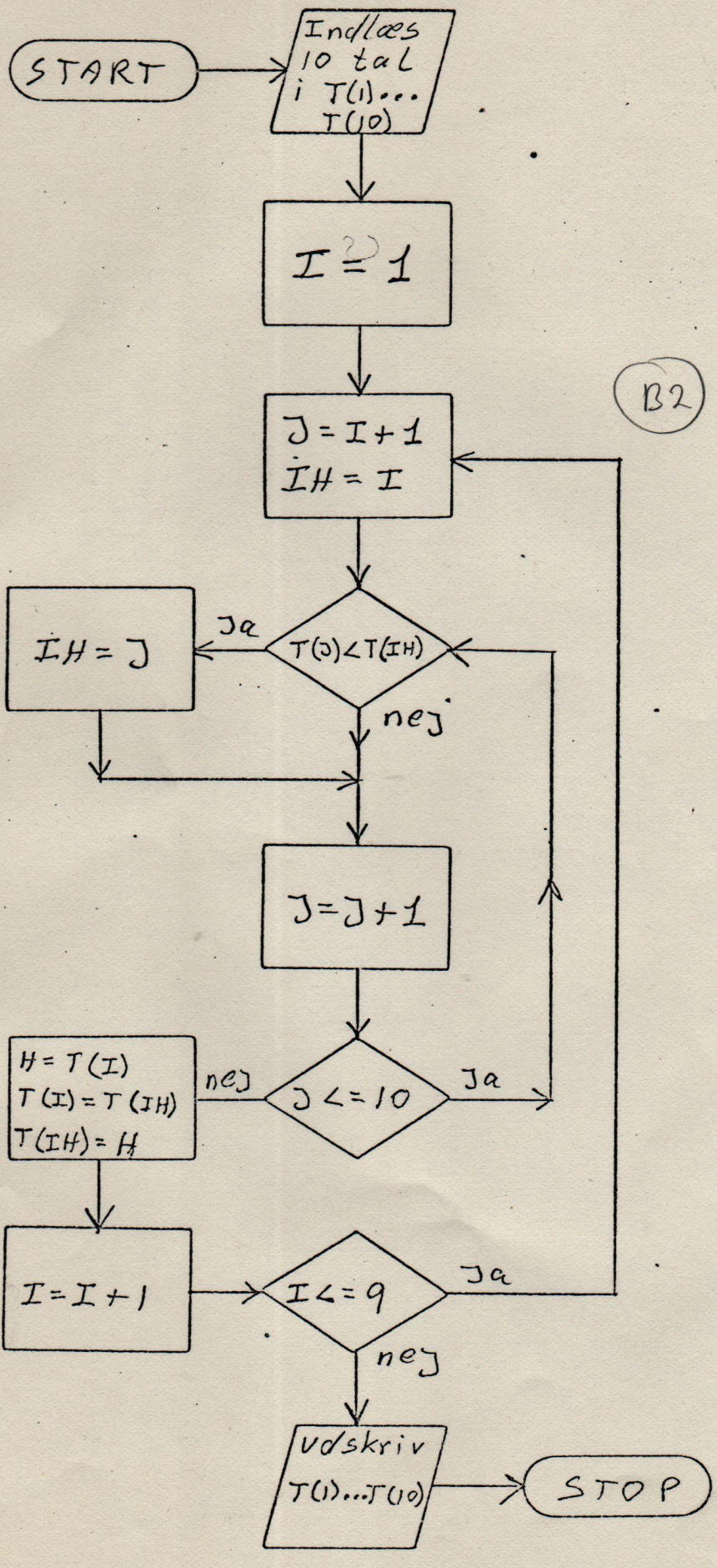
mål langden af navn tæl 3 fra og se på resten!

A 3

```
0010 // PROGRAM TÆNK PÅ ET TAL
0020 //
0030 DIM SVAR$ OF 1
0040 CONTINUE:=TRUE; ANTALGÆT:=1
0050 CLEAR
0060 PRINT "Tænk på et heltal mellem 1 og 100, så skal jeg forsøge at gætte det
     "
0070 INPUT "Når du har tænkt på et tal så tryk på <return> knappen": SVAR$
0080 RANDOM
0090 GÆT#:=RND(1,100); LL#:=1; HL#:=100
0100 REPEAT
0110   FEJL:=FALSE
0120   PRINT "Jeg gætter på tallet  ";GÆT#
0130   PRINT "Hvis tallet er:"
0140   PRINT "    1: rigtigt trykker du <return>"
0150   PRINT "    2: For lille, trykker du på  L-tasten"
0160   PRINT "    3: For stort, trykker du på  S-tasten"
0170   INPUT "Indtast L, S  eller  <return>: ": SVAR$
0180   CASE SVAR$ OF
0190   WHEN "S", "s"
0200     HL#:=GÆT#
0210   WHEN "L", "l"
0220     LL#:=GÆT#+GÆT# DIV 99
0230   WHEN ""
0240     EXEC RESULTAT("Jeg")
0250     CONTINUE:=FALSE
0260   OTHERWISE
0270     PRINT "Du har trykket på en forkert tast (";SVAR$;")."
0280     FEJL:=TRUE
0290   ENDCASE
0300   IF (NOT FEJL AND CONTINUE) THEN
0310     ANTALGÆT:+1; GÆT#:=(HL#-LL#) DIV 2+LL#
0320   ENDIF
0330 UNTIL NOT CONTINUE
0340 //
```

```
0350 PRINT
0360 PRINT
0370 PRINT "Nu er det din tur"
0380 PRINT " Jeg tænker på et heltal mellem 1 og 100, og du skal gætte det!"
0390 PRINT " Nu har jeg tænkt på et tal"
0400 I:=0; CONTINUE:=TRUE; NUM#:=RND(1,100)
0410 REPEAT
0420   I:+1
0430   PRINT "Hvad er dit ";I;". gæt: ";
0440   INPUT GÆT#
0450   IF GÆT#<NUM# THEN
0460     PRINT "Tallet er for lille !"
0470     PRINT
0480   ELIF GÆT#=NUM# THEN
0490     ANTALGÆT:=I
0500     EXEC RESULTAT("Du")
0510     CONTINUE:=FALSE
0520   ELSE
0530     PRINT "Tallet er for stort !"
0540     PRINT
0550   ENDIF
0560 UNTIL NOT CONTINUE
0570 END
0580 //
0590 PROC RESULTAT(S$)
0600   IF ANTALGÆT<2 THEN
0610     PRINT S$;" er EXPERT."
0620   ELIF ANTALGÆT<4 THEN
0630     PRINT S$;" er MÆGTIG GOD."
0640   ELIF ANTALGÆT<8 THEN
0650     PRINT S$;" ER GOD."
0660   ELIF ANTALGÆT<16 THEN
0670     PRINT S$;" er UNDER MIDDEL."
0680   ELSE
0690     PRINT S$;" er IKKE GOD."
0700   ENDIF
0710 ENDPROC RESULTAT
```

BLOKDIAGRAM



9/3 84
app B
opgave ~~[illegible]~~

SE BLOCKDIAGRAM

ukendt antal tal

```
                1  2  3  4  5  6  7  8  9  10
0010 DATA 7, 11, 2, 1, 3, 18, 2, 17, 9, 6
0015 DIM T(10)
0020 FOR I:=1 TO 10 DO
0030  READ T(I)
0040 NEXT I
0050 FOR I:=1 TO 9 DO
0060  IH:=I
0070  FOR J:=I+1 TO 10 DO
0080   IF T(J)<T(IH) THEN IH:=J
0090  NEXT J
0100  H:=T(I); T(I):=T(IH); T(IH):=H
0110  PRINT T(I);
0120 NEXT I
0130 PRINT T(10)
0140 END
```

DIM T(1000)
~~NUM=0~~
REPEAT (-99 bl.stop)
INPUT "Indtast nast tal: " ; A
IF A<>-99 THEN
NUM:= NUM+1
T(NUM):=A
ENDIF
UNTIL A = -99

FOR I=1 til NUM DO
FOR I=1 til NUM-1 DO
FOR J=I+1 TO NUM

hvis; ikke ender skrives ud under hinanden

Udskriften bliver:

```
1
2
2
3
6
7
9
11
17 18
```

	sign/dato 9/3 84	side 1
	erstatter	projekt EDB

6.10 Trinvis programforbedring, dansk

Eks.

Læs en række tal og summer dem og udskriv summen

(1)
1) Læs en række tal og lag dem sammen (hvorsidste tal er -99)
2) Udskriv summen

0 Gentag

sum = S

(2)
1. Læs et tal T
2. sæt S := T+S
3. indtil T = -99
4. skriv sum
5. STOP

REPEAT UNTIL

(3)
1. sæt S := 0
2. GENTAG
3. LÆS ET TAL T
4. SÆT S := S+T
5. indtil T = -99
~~6. skriv sum~~
6. lag 99 til de S := S+99 (eller S = S-T hvor T = -99)
7. skriv summen - udskriv S
8. stop

eller brug af IF THEN

1. sæt S := 0
2. LÆS ET TAL T
3. IF T = -99 så gå til 6.
4. ELLERS sæt S = S+T
5. gå til punkt 2
6. skriv S
7. STOP

S := S+T Cornel-80

LET S = S+T fik i basic

	sign/dato 9/3 84	side 2
	erstatter	projekt EDB

app B opgave 6.1

T
tabrakke

~~10 Gentag~~
~~20 Læs et tal T~~
~~30 IF~~

~~10 READ~~

~~1) Læs tabrakken ind~~

Dansk - pseudo program

10 Læs et tal T
20 sæt MIN := T
30 sæt MAX := T
40 Gentag følgende
50 Læs et tal T
60 hvis T < MIN så MIN = T
70 hvis T > MAX så MAX = T
80 indtil ikke flere tal
90 udskrivi min, max
10 stop

COMAL-80

```
10 // Dette er et bringen venligt program der finder min og max af
20 // en tabrakke
30 //
40 INPUT " Indtast første tal i tabrakken" : T
50 MIN := T
60 MAX := T
70 REPEAT WHILE T <> -99
80 INPUT " Indtast næste tal i tabrakke" : T
90 IF T < MIN THEN MIN := T
100 IF T > MAX THEN MAX := T
110 UNTIL T := -99 ENDWHILE
120 PRINT " MINIMUM " = ; MIN
130 PRINT
140 PRINT " MAXIMUM " = ; MAX
150 END
```

(~~hvis ikke flere tal~~ - ~~[illegible]~~ (-99 betyder stop) ~~[illegible]~~)

<> ~ ≠

hvis dette vi min idette tilfælde blive fortsat hvis der ikke er tal < -99 i tabrakken

altså

```
70 INPUT " Indtast
80 IF T := "" THEN GOTO
90 IF T < MIN
100 IF T > MAX
110 GOTO 80
120 PRINT
```

	sign/dato 9/3 PY	side 3
	erstatter	projekt EDB

opgave 6.2 i app B

```
10 //  Dette er et eks --
20 //
30 //  en talrække
40 //
50 INPUT "Indtast første tal" : T
60 MIN:=T
70 MAX:=T
75 IF T<>-99 THEN
80 WHILE T<>-99
90 IF T<MIN THEN MIN:=T
100 IF T>MAX THEN MAX:=T
110 Input "Indtast næste tal (-99 betyder stop)"
120 ENDWHILE
130 PRINT "minimum = "; MIN
140 PRINT
150 PRINT "maximum = "; MAX
155 ELSE IS PRINT "TOM talrække"
160 END     159 ENDIF
```

	sign/dato 9/3	side 4
	erstatter	projekt EDB

6-2

Læse tal og lægge dem sammen og dividere med antallet af indlæste tal

REPEAT

S = sum af tal

n = antal tal

1/ INPUT " Indtast positive tal" : T

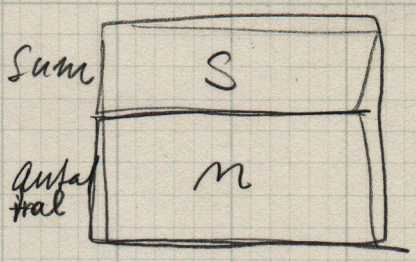
2/ n = n + 1 STEP 1?

3/ sæt S := S + T

4/ UNTIL [illegible] .

5/ S := S / n

6/ slut



indsæt 1. tal

indsæt 2. tal

[illegible] ét tal indtastet

[illegible] 2 tal indtastet

Læg tal sammen

Læg tal sammen osv.

div SUM med antal indtastede tal ~ $\bar{x}$

udskriv $\bar{x}$

slut

---

sæt n og s lig nul

1 7 50 | -99

n = 3 | n = 4

1) REPEAT

2) INPUT " indtast positive tal" : T

3) n = n + 1

4) S := S + T

5) UNTIL ? stop funktion -99 fx. el. i talrækken usandsynligt forekommende tal

6) ~~INPUT~~ ~~S := S/n~~ $S = \frac{S+99}{n-1}$

7) PRINT S

8) SLUT

1

1+1

$S := \frac{S+99}{n-1}$

4 ~~INPUT "S, n" = 0,0~~ ? S=0 n=0

5

10 PROGRAM GENNEMSNIT

20 REPEAT

30 INPUT " indtast positive tal (-99 betyder stop)" : T

40 n := n + 1

50 S := S + T

60 UNTIL T = -99

70 S := (S+99)/(n-1) / 75 PRINT 80 PRINT S / 90 SLUT /

KARTOTEKER	sign/dato 12/3 84	side 1
7.26	erstatter LB	projekt EDB

SEKVENTIEL TILGANG de indrette data tar længere tid at hente
alle data på magnetbånd er sekventielle

SEKVENTIEL FIL
har sekventiel tilgang

DIREKTE TILGANG alle data hentes frem lige hurtigt
plader kan være direkte eller sekventielle

DIREKTE FIL
har direkte tilgang

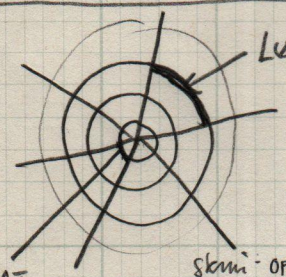
ASCII :

BINÆR : fylder mindre end ascii

CATALOG!

LØN, OPL	
spor nr	2
sektor nr	8
længde	4
status	W

filtype



skrin - OPEN FIL 1
LØN, OPL
data skrin
over i tabel
lagret i skrine under 1 ∴ kan nøjes med at referere til 1

Filnr.	SPOR	SEKTOR	LÆNGDE	STATUS
0				
1	2	8	4	W
2				
3				
4				
5				
6				
7				
8				
9				

10 kartoteker kan være åbne samtidig. (i COMPAS 80)

CLOSE FILE 1
⇒ alle relationer mellem fil 1 og data
∴ kan åbne mulh andre fil

A 3

	sign/dato 12/3 84	side 2
	erstatter KH	projekt EDB

7.34 Skrivesætninger.

7.35

REKORDTAVLE:

Rekorden i sakkelob på tid = 835.4 meter

HURRA!!

SELECT OUTPUT "LP:" skal stå i programmet – udskrive på linjeskriver

SELECT OUTPUT "MINFIL" diskette –

"DS:" dataskærmen –

LIST LP : selve program udskrives på printeren

7.38 COMAL-80

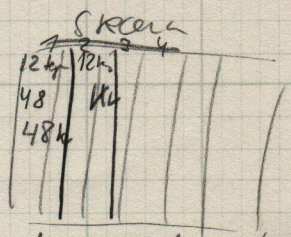
, angiver at 48; "Kr" 48; "Kr" ; lavere ei blank ;

; angiver at Kr skal stå lige efter med en plads i mellem

Her er det nærmeste ligemeget – ∴ ~~det er ligemeget~~

Med tab = 0 fås 48; "Kr" 48 kr

48; "Kr" 48 kr
et mellemrum



Ken indeles med tab
fås 12 tegn i hver zone

Tab = 12 ⇒ 12 tegn
i hver gruppe

7.37 Betingede sætninger

A 3

6-3

```
10  // PGROGRAM GENNEMSNIT        [PRINT " PR..   G.SNIT"
20     INPUT  S:=0
30     INPUT  n:=0
40     REPEAT
50       INPUT " Indtast facit tal (-99 betyder stop)" : T
60       n:=n+1
70       S:=S+T
80     UNTIL  T:=-99
90     S:= (S+99)/(n-1)
100    PRINT
110    PRINT S                    PRINT "gennemsnit=" ; S
120    END
```

```
       p
       R
INPUT  S
       B
INPUT  L
INPUT  F
S := L - F
```

IF $S \leq 103.200$

$\frac{103.200 \times 48.3}{100\ S_A} = S_1$

IF $103.200 < S < 172.200$

$\frac{103.200 \cdot 48.3}{100\ S_B} + \frac{68.800 \cdot 62.7}{100\ S_C} = S_1$

IF $172.200 < S$

$\frac{(S - 172.200)\,73.5}{100\ S_D} = S_1$

	sign/dato 12/3 84	side 4
opgave mik 12/3 84	erstatter	projekt EDB

```
10  // PROGRAM RESTSKAT
20  INPUT "Indtast bruttolön 1983 =" L
30  INPUT " — " Fradrag 1983 =" F
40  INPUT " —"— Betalt skat i 1983 =" B
50  INPUT " —"— Restskat fra 81 =" R
60  INPUT " —"— Personfradrag skattemæssig værdi =" = P
70  S := L - F
80  KONTROL?
90  IF S ≤ 103200 THEN
100         S1 := (S·48,3)/100
110 ELIF  S ≤ 172000 THEN
120         S1 := (103200·48,3)/100 + ((S - 103200)62,7)/100
130 ELSE  172000 < S
140         S1 := (103200·48,3)/100 + (68000 62,7)/100 + ((S - 172000)73,5)/100
150 ENDIF
(160 PRINT S1)
170 T := B - (S1 - P) - R
180 PRINT
190 PRINT T
200 PRINT "---"
```

12/3 84 [illegible]

Opgave:

Lav et program der beregner hvor meget man skal have tilbage i skat for 1983 når skuffe:

inddata kontrol ~~[illegible]~~

L = Lønnen (brutto) i 1983
F = Fradrag (totalt) i 1983
B = Betalt skat i 1983
R = Restskat fra 1981
P = Personfradraget (Skattemæssig værdi)
S = Skattepligtig indkomst (= L − F)

Af S skal betales 48.3 % i skat af de første 103.200 kr, 62.7% af de næste 68.800 kr og 73.5% af den del af S der overstiger 172.000 kr.

Den heraf beregnede skat kalder vi for S1.

Det beløb vi skal have tilbage i skat kalder vi for T. T kan nu beregnes af formlen:

$$\boxed{T = B - (S1 - P) - R}$$

	sign/dato	side
LB OPGAVE mak 12/3 84 PROGRAMOPRET M.V.	13/3 84	2
	erstatter	projekt EDB

opgave
udskrive efter nr eller efter navn

STATUS skal vare 0
~ GYLDIGT KORT

NR — max / min

NAVN — søgetekst på NAVN — fiktes søgetekst 2200 => alle med 2200 udskrives
if søgetekst IN NAVN THEN 2200 forekommer som en del af NAVN eks Poul Bart, 2200
* betyder at alle kort udskrives

NR OR NAVN

OPEN FILE 4, "KARTOTEK.OPL", RANDOM, 80

READ FILE 4,

Find ~~[illegible]~~ medarbejder nr og udskriv alle data på ~~[illegible]~~ nr.

~~INPUT "Indtast medarbejder nr.": MNR#~~ 0 999

~~FOR MNR# - TO - DO~~

INPUT "Ønskes udskrift efter navn eller medarbejder nr.": ~~NAVN~~ NAVN ~~MNR#~~ NR
IF ~~NAVN~~ SVAR$ = NR
INPUT "Indtast søgetekst"
~~REPEAT~~ MNR# $2 TO MAX DO
UNTIL

PROC UDSKRIV	sign/dato 13/3 84	side 2
	erstatter	projekt EDB

SVAR$

NAVN
NR

NI

INPUT " Ønskes søgning efter Navn el Nr " : SVAR$

IF SVAR$ = NR. ~~Indtast~~ THEN

INPUT " Indtast medarbejdernr " = ~~MNR#~~ MED

~~FOR~~ 785

~~IF SVAR$ = NAVN THEN~~

~~INPUT " Indtast fulde navn " = NAVN$~~

FOR I=2 til MAX DO ok=False

READ FILE 4, I : STATUS#, MNR#, NAVN$,

LKAT#, SPCT#, FRADRAG, REG#, KONTONR$

IF STATUS = 0 AND MNR# = MED THEN

EXEC ~~PROC~~ UDSKRIV

ok = TRUE

NEXT I ENDIF

IF not ok

NAVN NR ELSE

INPUT " Indtast fulde navn " : FELIX$

FOR I=2 til MAX DO

READ FILE 4, I : hele sensen

IF STATUS=0 AND FELIX$ IN NAVN$ THEN

EXEC PROC UDSKRIV

NEXT I

ENDIF

Max Peder Jensenius

Max P. Jensene

LOAD

INPUT " Indtast max, min (cm) ": MAX, MIN

	sign/dato 14/3	side
	erstatter	projekt RDB

Input " " : max, min

loop

Repeat

EXEC DACONV (1.8, 0)

EXEC ADCONV (SP, 0)

HF = 8 + 5.7/5 · SP

UNTIL HF >= MAX

REPEAT

EXEC DACONV (0, 0)

EXEC ADCONV (SP, 0)

HF = 2 + 5.7/5 · SP

UNTIL HF <= MIN

ENDLOOP

	sign/dato	side
	erstatter	projekt

efter indledende? se program ændre

DiM ??

```
OPEN FILE0, "KARTOTEK.OPL", RANDOM, 80
READ FILE 0, 1: MAX#, ANTAL#
// PROGRAM UDLÆSNING AF MEDARBEJDER OPLYSNINGER
// : Udskriuning procedure
PROC UDSKRIV
        PRINT
          :
        osv
ENDPROC UDSRIV
```

COPYRIGHT
FJUMREMIKKEL & CO.

? ERKLÆRINGER STARTVÆRDIER - se PROGRAMÆNDRE

Indtast NR for medarbejdern indtast for søgning eft med arbejdn efternav

```
INPUT "~~Ønskes~~ søgning efter Nr. eller ~~efter~~ NAVN": SVAR$
IF SVAR$ = NR. THEN
INPUT "Indtast medarbejdernr" = MED
OK = FALSE
FOR I = 2 TO MAX DO
READ FILE 0, I : STATUS#, MNR#, NAVN$, LKAT#, SPCT#, ;
      FRADRAG, REG#, KONTONR$
   IF STATUS = 0 AND MNR# = MED THEN
   EXEC UDSRIV
   OK = TRUE
   ENDIF
NEXT I
IF NOT OK THEN
PRINT "~~Identificeret~~ Ingen medarbejdere med det nr."
ENDIF
ELSE
INPUT "Indtast efternav": FELIX$
OK = FALSE
FOR I = 2 TO MAX DO
READ hele emnøren
IF STATUS = 0 AND FELIX IN NAVN$ THEN
EXEC UDSKRIV
OK = TRUE
ENDIF
NEXT I
IF NOT OK THEN
PRINT "Ingen medarbejdere med det navn"
ENDIF
ENDIF
```

Rettet se udskrift

A 3

Ordre i CP/M (Control Program/Microprocessor).

Filnavne under CP/M er beskrevet i kompendiet på s. 9.7, 9.22 og 9.23.

DIR devise fehn B: primærdel.sekundærdel (s. 9.22, 9.23)
Udskriver hele eller dele af filkataloget.

ED dev: primærdel.sekundærdel (s. 9.9, 9.12, 9.15)
Er en editor til brug ved tekstbehandling. Skal en ny fil oprettes skrives blot ED og filnavnet. Skal der skrives (indsættes) i filen angives ordren I. Er man færdig med skrivningen, og vil ud af editoren trykkes først på knappen CONTROL og Z samtidig og herefter tastes et E.
Ønsker man at kunne skrive med små bogstaver, angives som første ordre: -U
Ønsker man at erstatte en tekst med en anden skrives f.eks. S And ↑Z Vin hvor S betyder substitute og ↑Z betyder control Z.

ERA dev:primærdel.sekundærdel (s. 9.13)
Sletter de(n) angivne fil(er) fra disketten.

PIP dev: primærdel.sekundærdel = filnavn 1, filnavn 2,.....
UDDATA-FIL = INDDATA-FIL(ER) (s. 9.18),
Kopierer filer mellem forskellige medier. Er der flere end en inddatafil, sammensættes filerne (konkateneres) i den rækkefølge de angives.

REN filnavn 1 = filnavn 2 (ikke omtalt)
Omdøber filnavn 2 til filnavn 1. Der er altså ikke tale om en kopiering!

STAT dev: filename {$R/O eller $R/W} (9.20, 9.21)
Aflæser filstatus, og ændrer beskyttelses-status for filer.
Det i { } angivne $R/O betyder: sæt denne til Read/Only.

TYPE dev: filnavn (s. 9.12)
Udskriver filindholdet af en ASCII fil på skærmen.

15/1 84

Øvelse i filbehandling.

1. del af øvelsen kan udføres på alle stationerne.
2. del kan kun udføres på station 3 fra venstre (dobbelt floppy-disk).

1. del: CP/M-operativsystemet indlæses, og følgende ting foretages: (diskette 12, 13, 14, 15, 18, 19, 20 eller 21).

1: Udskriv navnene på filer der som primærdel af navnet kun har et enkelt tegn.

2: Udskriv navnene på filer der som primærdel har 3 tegn og hvor 3. tegn er et P, og hvor 1. tegn i sekundærdelen er et C.

3: Udskriv navnene på filer der som sekundærdel har tegnene COM.

4: Udskriv indholdet af alle filer der som sekundærdel har tegnene TXT

5: Udskriv navnene på filer der som sekundærdel har tegnene SUB.

6: Lav en fil der er sammensat af filerne A.TXT, B.TXT, C.TXT og giv dem navnet D.TXT

7: Udskriv indholdet af D.TXT på skærmen.

8: Undersøg størrelsen af samtlige filer på disketten, og hvor meget plads der er tilbage på disketten.

9: Beskyt filen D.TXT mod sletning.

10: Opret en fil der indeholder dit navn og din adresse, v.h.a. editoren ED, under navnet ADRESSE.MIN

11: Udskriv hele filkataloget.

12: Slet filen D.TXT

13: Omdøb filen ADRESSE.MIN til ANDERS.AND

14: Slet alle filer med AND som sekundærdel.

2. del: Gå til station 3 og indsæt din diskette fra første del i venstre spalte (drive A), og diskette nr. 16 i højre spalte (drive B).

15: Kopier alle filer fra disketten til venstre over på disketten til højre.

16: Udskriv filkataloget for disketten til højre uden at bytte om på disketterne.

17: Udtag disketten til højre og indsæt i stedet diskette nr. 17. Gentag punkt 15 og 16.

sign/dato	side
9/3 84	5
erstatter	projekt
	EDB

EDB SJOVEORD

9 bogstav + et mellemrum

```
10  DIM LIN$ OF 10
20  RANDOM
30  SELECT OUTPUT "LP:"
40  FOR I := 1 TO 400 DO
50  FOR J := 1 TO 9 DO
60      LIN$ (J) := CHR$ (RND (65, 93))
70  NEXT J
80  PRINT LIN,$" ";
90  NEXT I
100 END
```

aktiverer Random generator
udskriver linjeser
skal var

blanke sættes efter

alle store bogstav en repr. af tal fra 63 til 93 i ASCII

CHAR CHR$ omsætte tallet til bogstavet der svarer til

BLOKDIAGRAM BLOKDIAGRAM

	7	11	9	2	3	7	2			
T	1	2	3	4	5	6	7	8	9	10
tallokken:	7	11	2	1	3	18	2	17	9	6
	I = 1	J								

I peger på 1. tal i tallokken
J = I + 1
J peger på næste tal i tallokken

I = IH hvis T(J) < T(IH)

Kartoteker

OPRETTE
ÆNDRE
SLETTE
UDSKRIVE
SORTERE

Plads (der skal være plads)
kontrollere eksisterend nr.
Inddata kontrol (grænser, fel skal 80-100

eksisterer nr.? kontrollere dette
Inddata kontrol

eksisterer nr.?
udskriv det der slette inden det slettes

eksisterer nr.?
udvælge det kort man vi have udskrevet
J: fra - til nr.

skal kunne selektere kartoteket

KARTOTEK

LAV UDSKRIVNINGSPROGRAM til et kartotek
der kan oprette, ændre og slette

	sign/dato	side
	erstatter	projekt

JUKS

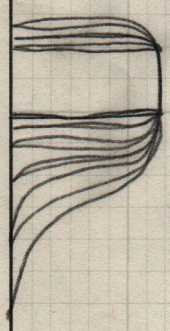
5// PROGRAM SJOVE ORD
6//

```
10  DIM STAV$ (5) OF 3
20  FOR I#:=1 TO 5 STEP1 DO
30  INPUT " indtast en stavelse på 3 bogstaver : ": STAV$ (I#)
40  NEXT I#          TAK / BIT / FIS / HAK / MUK
50  DIM LIN ~~ORD~~$ OF 10 ?
60  FOR I#:=1 TO 5 STEP1 DO
70  FOR J#:=1 TO 5 STEP1 DO
80  FOR K#:=1 TO 5 STEP1 DO
90  LIN ~~ORD~~$ := STAV$ (I#) + STAV$ (J#) + STAV$ (K#)
95  SELECT OUTPUT "LP"
100 PRINT ~~ORD~~ LIN$," ";
110 NEXT K# ~~BIT~~
120 NEXT J# 130
130 NEXT I# 140
140 PRINT " Per Hojholt ville være misundelig"
150 END
```

enten stavelse alle tre
vare med ellen ingen af de

ORD bestyrkt!

behøver ikke : starven den selv



	sign/dato	side
	erstatter	projekt

```
PROGRAM SJOVE ORD
PRINT " Copyright
 DIM STAV$ (5) OF 3
 FOR I := 1 TO 5 STEP 1 DO
 INPUT " indtast én stavelse på 3 bogstaver " : STAV$ (I)
 NEXT I
 DIM LIN$ OF 10
    FOR I := 1 TO 5  STEP 1  DO
    FOR J := 1 TO 5   STEP 1  DO
    FOR K := 1 TO 5   STEP 1  DO
    LIN$ := STAV$ (I) + STAV$(J) + STAV$(K)
    SELECT OUTPUT "LP:"
    PRINT LIN$," " ;
  NEXT K
  NEXT J
  NEXT I
  PRINT "Per Højholt ville have vævet misundelig"
  END
```

tegn ?

tegn ?

A

```
0010 // PROGRAM TÆNK PÅ ET TAL
0020 //
0030 DIM SVAR$ OF 1
0040 CONTINUE:=TRUE; ANTALGÆT:=1
0050 CLEAR
0060 PRINT "Tænk på et heltal mellem 1 og 100, så skal jeg forsøge at gætte det
     "
0070 INPUT "Når du har tænkt på et tal så tryk på <return> knappen": SVAR$
0080 RANDOM
0090 GÆT#:=RND(1,100); LL#:=1; HL#:=100
0100 REPEAT
0110   FEJL:=FALSE
0120   PRINT "Jeg gætter på tallet  ";GÆT#
0130   PRINT "Hvis tallet er:"
0140   PRINT "    1: rigtigt trykker du <return>"
0150   PRINT "    2: For lille, trykker du på  L-tasten"
0160   PRINT "    3: For stort, trykker du på  S-tasten"
0170   INPUT "Indtast L, S  eller  <return>: ": SVAR$
0180   CASE SVAR$ OF
0190   WHEN "S", "s"
0200     HL#:=GÆT#
0210   WHEN "L", "l"
0220     LL#:=GÆT#+GÆT# DIV 99
0230   WHEN ""
0240     EXEC RESULTAT("Jeg")
0250     CONTINUE:=FALSE
0260   OTHERWISE
0270     PRINT "Du har trykket på en forkert tast (";SVAR$;")."
0280     FEJL:=TRUE
0290   ENDCASE
0300   IF (NOT FEJL AND CONTINUE) THEN
0310     ANTALGÆT:+1; GÆT#:=(HL#-LL#) DIV 2+LL#
0320   ENDIF
0330 UNTIL NOT CONTINUE
0340 //
```

```
0350 PRINT
0360 PRINT
0370 PRINT "Nu er det din tur"
0380 PRINT " Jeg tænker på et heltal mellem 1 og 100, og du skal gætte det!"
0390 PRINT " Nu har jeg tænkt på et tal"
0400 I:=0; CONTINUE:=TRUE; NUM#:=RND(1,100)
0410 REPEAT
0420    I:+1
0430    PRINT "Hvad er dit ";I;". gæt: ";
0440    INPUT GÆT#
0450    IF GÆT#<NUM# THEN
0460      PRINT "Tallet er for lille !"
0470      PRINT
0480    ELIF GÆT#=NUM# THEN
0490      ANTALGÆT:=I
0500      EXEC RESULTAT("Du")
0510      CONTINUE:=FALSE
0520    ELSE
0530      PRINT "Tallet er for stort !"
0540      PRINT
0550    ENDIF
0560 UNTIL NOT CONTINUE
0570 END
0580 //
0590 PROC RESULTAT(S$)
0600    IF ANTALGÆT<2 THEN
0610      PRINT S$;" er EXPERT."
0620    ELIF ANTALGÆT<4 THEN
0630      PRINT S$;" er MÆGTIG GOD."
0640    ELIF ANTALGÆT<8 THEN
0650      PRINT S$;" ER GOD."
0660    ELIF ANTALGÆT<16 THEN
0670      PRINT S$;" er UNDER MIDDEL."
0680    ELSE
0690      PRINT S$;" er IKKE GOD."
0700    ENDIF
0710 ENDPROC RESULTAT
```

1

2

3

4

5

6

7

8

9

10

11

12

BICP

TLF. 02 - 45 15 16

nr. 35 52 49

LABORANTSKOLEN
Rådmandsgade

22/2-84

KURSUS I EDB 7. - 16/3-1984

	Onsdag 7/3	Torsdag 8/3	Fredag 9/3
8,30-12,00	Introduktion LB + KH Datamatens-opbygning LB Opgaver LB + KH	Comal-80 LB	Løsnings-beskrivelses-metoder KH
12,00-12,30	F R O K O S T		
12,30-16,00	Yem KH + LB Øv. beskri-velse KH + LB Øvelser/yem LB + KH	Comal-80/ Øvelser LB + KH	Opgaver LB + KH Kartotek-systemer LB + KH

Yem = ydre enheder og medier

	Mandag 12/3	Tirsdag 13/3	Onsdag 14/3	Torsdag 15/3	Fredag 16/3
8,30-12,00	Comal-80 LB Comal-80 KH	Comal-80 KH Comal-80 LB	Øvelser/ praktisk anvendelse LB + KH	Filbe-handling / Øvelser LB + KH Filbehandling LB + KH	Øvelser / Spec. ø. LB + KH
12,00-12,30	F R O K O S T				
12,30-16,00	Opgaver LB + KH Øvelser/ Kartotek LB + KH	Opgaver LB + KH	Opgaver LB + KH Øvelser/ filbehandling LB + KH	Filbehand-ling LB	Afslutning LB + KH

Undervisere: Akademiingeniør Kirsten Holm
Akademiingeniør Leon Bennicke

Sted: Laborantskolen, Rådmandsgade 71, 2200 Kbh. N, Tlf. 01-851266.

Varighed: 60 timer

LABORANTSKOLEN
Rådmandsgade

20/2-84

DELTAGERLISTE

FOR

KURSUS I EDB FOR LABORANTER 7. - 16. marts 1984

Andersen, Inge-Dorte	Cheminova	o7-834100
Andersen, Mette Pauli	Statens Seruminst.	o1-952817
Christensen, Anders B.	De Danske Sukkerfa.	o1-954100/36
Fischer, Vibeke	Retsmed.inst.	o1-373222/334
Friis, Vibeke	Novo	o2-982333
Grindsted, Vivian	Haldor Topsøe	o2-878100
Hansen, Karen Margrethe	Danm.Farm.Højskole	o1-37o85o/286
Elisabeth N. Hjarding	F.L. Smidth & Co.	o1-301166
Honig, Annelise	Zoofysiol.lab.	o1-377000
Jørgensen, Per	-	o2-643637
Kiersz, Aleksander	Jens Villadsens Fa.	o1-502433/23
Kreisby, Solveig	Stat.Seruminst.	o1-952817/2566
Larsen, Poul Erik	Inst.f.med.mikrobiologi	o1-391033
Nielsen, Anette H.	Nunc	o2-359065
Nørringgård, Eva	G.G.U.	o1-118866
Petersen, Lars Boje	Haldor Topsøe	o2-878100/238
Raskmark, Jørgen	D.T.H.	o2-884144/3646
Schøneberg, Astrid	D.T.H.	o2-881444/2123
Straadt, Elsebeth	Dumex	o1-544500
Sørensen, Marianne B.	Landbohøjskolen	o1-351788/2667
Venneberg, Leif	Cheminova	o7-834100
zum Vohrde, Kirsten	-	o1-180003
Christiansen, Claus Norup	Novo	o2-982333

EDB-Kompendium

FOR LABORANTER



2.Udg. 1983

Leon Bennicke.
Kirsten Holm.

FORORD:

Dette kompendium er skrevet til brug ved undervisning i faget Datalære på Laborantskolerne. Det kursus der ligger til grund for kompendiet, gives som specialkursus for færdiguddannede laboranter uden forudgående kendskab til EDB. Som grundlag for kompendiets tilblivelse ligger forfatternes selverhvervede erfaring med laboranters brug af EDB i praksis. Som følge heraf vil en stor del af stoffet være rettet mod den praktiske side af EDB (hvordan skriver man data, hvordan læser man data o.s.v.). Erfaringsmæssigt volder det, for nye brugere af EDB, problemer at overskue hvilke hjælpemidler man kan forvente at finde i et brugervenligt EDB-system, hvorfor dette emne ligeledes vil blive behandlet i detaljer. Kompendiet behandler emnerne:

1: Datamatens principielle opbygning og virkemåde.
2: Ydre enheder og medier.
3: Løsningsbeskrivelses-metoder.
4: Programmering og afprøvning.
5: Problemanalyse og løsningsforslag.
6: Filbehandling.
7: Praktisk anvendelse af EDB i laboratoriemiljø.

Udover det teoretiske stof, indeholder kompendiet opgaver og løsninger, kodetabeller, brugervejledning for opstart af comet-datamaten, regning med binære tal og en ordliste med nogle EDB-fagord og deres betydning, altsammen vedlagt som appendix A-F.

Som programmeringssprog er benyttet COMAL-80, hvor COMAL står for COMmon Algorithmic Language. Dette sprog er valgt på grund af dets forholdsvis enkle opbygning og lighed med BASIC. Samtidig er det muligt at lave struktureret programmering, der i sin form minder om PASCAL, hvorfor evt. overgang til dette sprog vil kunne ske ubesværet. Endelig er COMAL et konverserende sprog d.v.s. at sprog der er velegnet til terminalbrug, hvor man normalt har en dialog med datamaten.

Kompendiet er 1. udgave, og vil som sådan sandsynligvis indeholde fejl, som snirklede formuleringer, bortfald af ord m.v., der ikke er opdaget ved korrekturlæsningen. Vi vil derfor med tak modtage forslag til rettelser og omformuleringer m.v. hvor sådanne af læseren skønnes nødvendige.

Endelig vil vi rette en tak til alle der direkte eller indirekte har haft indflydelse på det færdige resultat. Specielt siger vi tak til det altid utrættelige kontorpersonale på Rådmandsgade Laborantskole, uden hvis arbejde med maskinskrivning af vore ulæselige manuskripter, kompendiet ikke ville være blevet færdig til tiden.

Rådmandsgade Laborantskole
november 1982

Leon Bennicke
Kirsten Holm

EDS 14/3 84

ØVELSE I PROCESSTYRING.

Formål: at illustrere brugen af A/D- og D/A-Converteren i en simpel processtyring.

Opstilling: Datamat 4 (helt til højre i lokalet) er udstyret med en A/D- og D/A-Converter, hvor forbindelserne til disse er ført ud i en grå kasse på datamatens højre side. A/D-Converteren har 8 indgange (kanal 0-7) hvor kanal 0-3 har en opløsning på 10.0/255 = 0.04 Volt og kanal 4-7 en opløsning på 0.1/255 = 0.4 mV. Den maximalt tilladelige indgangsspænding på kanal 0-3 er 10.0 V og på kanal 4-7 100 mV.

D/A-Converteren har 2 udgange (kanal 0-1) der hver kan afgive en spænding mellem +10.0 og -10.0 V. D/A-Converteren har en udgangsmodstand på 100 ohm, hvorfor en maximal belastning på 10 x 100 ohm må tilrådes (a.h.t. nøjagtighed). Begge kanaler har en opløsning på 5 mV.

Opstillingen der skal styres er vist på nedenstående tegning, fig. 1.

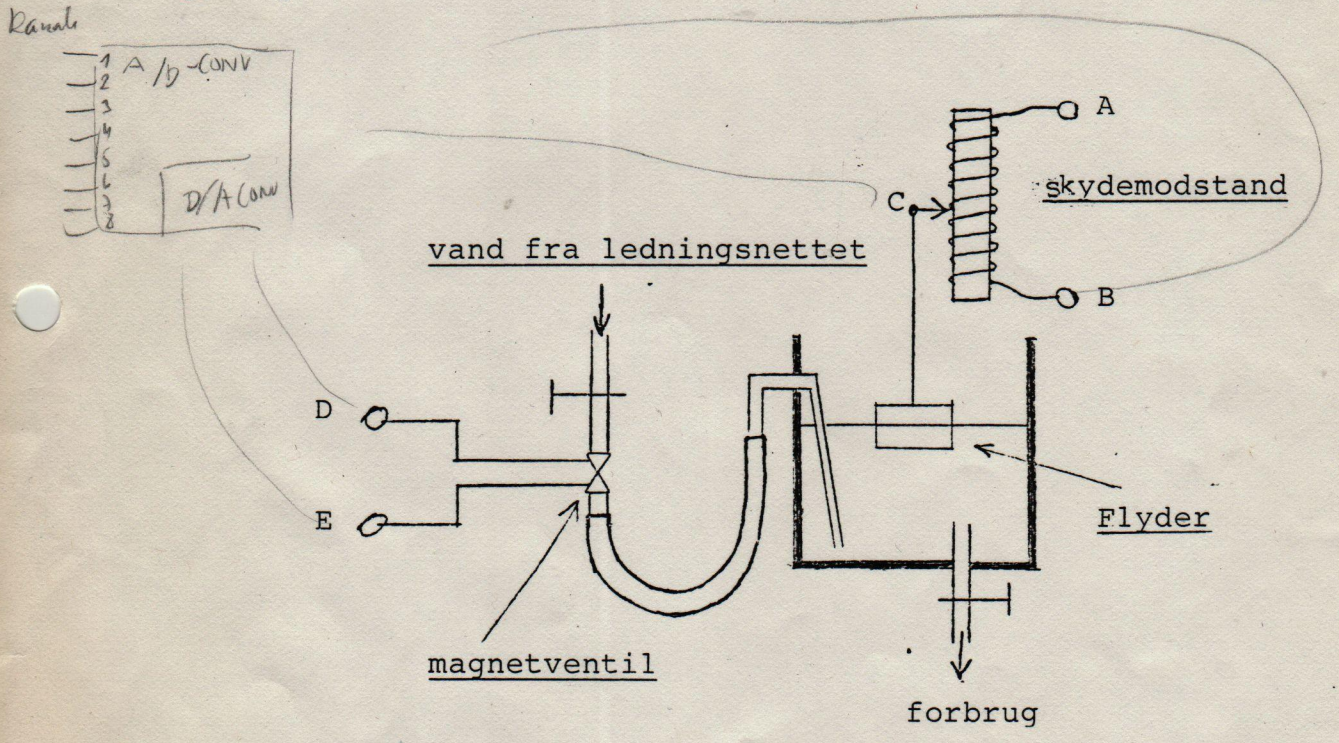
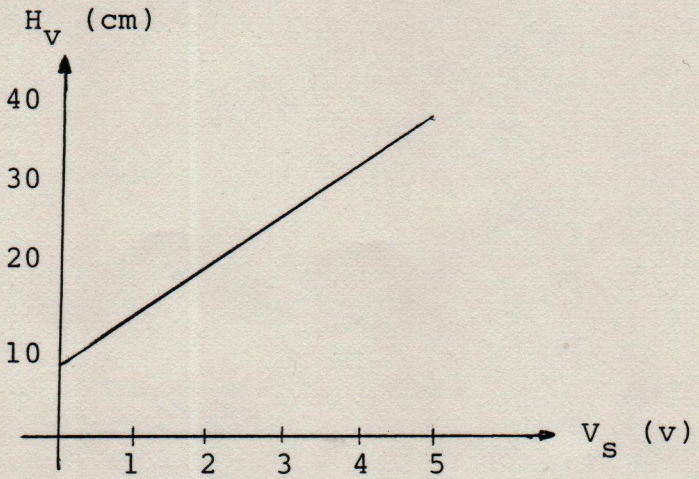


Fig. 1 Over punkterne A-B lægges en spænding på 5V. C-B forbindes til A/D Converteren, og D-E til D/A-Converteren via en forstærker som er vist på fig. 4.

Den består af en vandbeholder med afløb (til forbrug). Til påfyldning af vand i beholderen anvendes en magnetventil der er tilsluttet vandledningsnettet. Til registrering af vandstanden i beholderen anvendes en flyder i forbindelse med en skydemodstand, således at spændingen V_s mellem 0 (jord) og skyderen vil være et udtryk for væskestanden H_v idet $H_v = H_o + k \cdot V_s$ hvor H_o er væskestanden for $V_s = 0$, og k er en konstant der afhænger af skydemodstandens udformning her længden (enhed: cm/v).

Hv væskestand
Vs spænding når Vs = 0 må Ho !



Figur 2.

I ovenstående eksempel figur 2, er H_o = 10 cm og k = 20/3 = 6,67 cm/v.

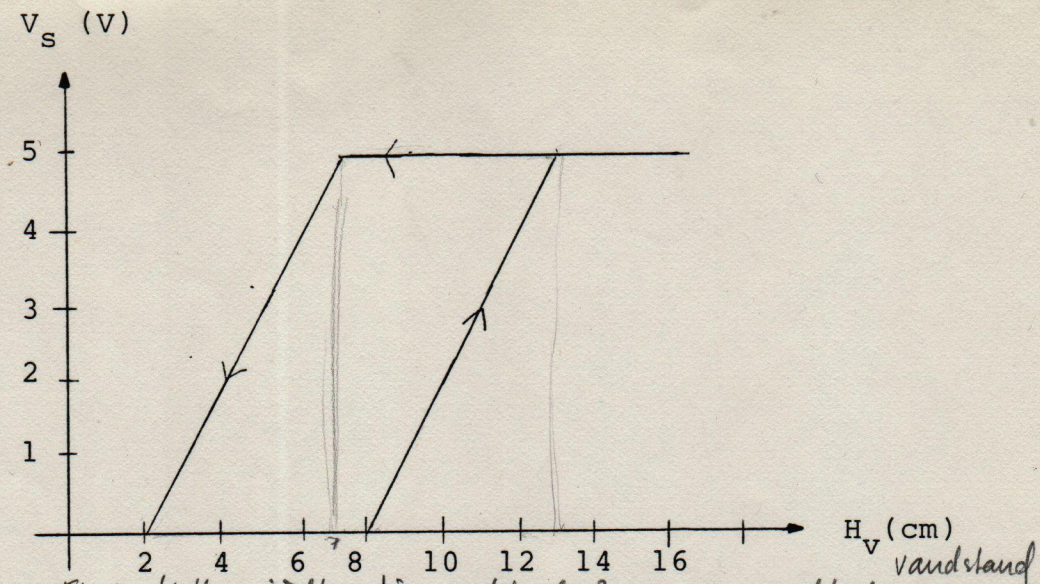
I forsøgsopstillingen er skydemodstandens længde 5.7 cm og spændingen herover skal være 5v. Find konstanten k for skydemodstanden:

$H_v = H_o + k \cdot V_s$

$k = \frac{H_v - H_o}{V_s} = \frac{5{,}7 - 0 \text{ cm}}{5} = \frac{5.7}{5} = 1{,}14 \text{ cm/V}$

P.g.a. Archimedes lov vil H_o for fyldning og tømning være forskellige. I det skitserede system er H_o for fyldning $H_{o,f}$ = 8 cm og H_o for tømning $H_{o,T}$ = 2 cm.

Dette fænomen er vist på omstående tegning.



Flyder flyder sig ikke for vandstand er 8 cm oppe max vandstand 13
Tilbage flyder flyder sig først max vand er nåt ned til 7 cm

Fig. 3 Spændingen V_s over punkterne BC som funktion af væskehøjden H_v.

Angiv formlerne for vandstanden som funktion af V_s for fyldning H_F og for tømning H_T, med indsatte konstanter (k og H_o):

$H_F = 8 + \frac{5.7}{5} VS$ Fyldning | $12 = 1.14 \cdot 8 + 1.4 \cdot VS \Rightarrow VS = \frac{4}{1.4} =$

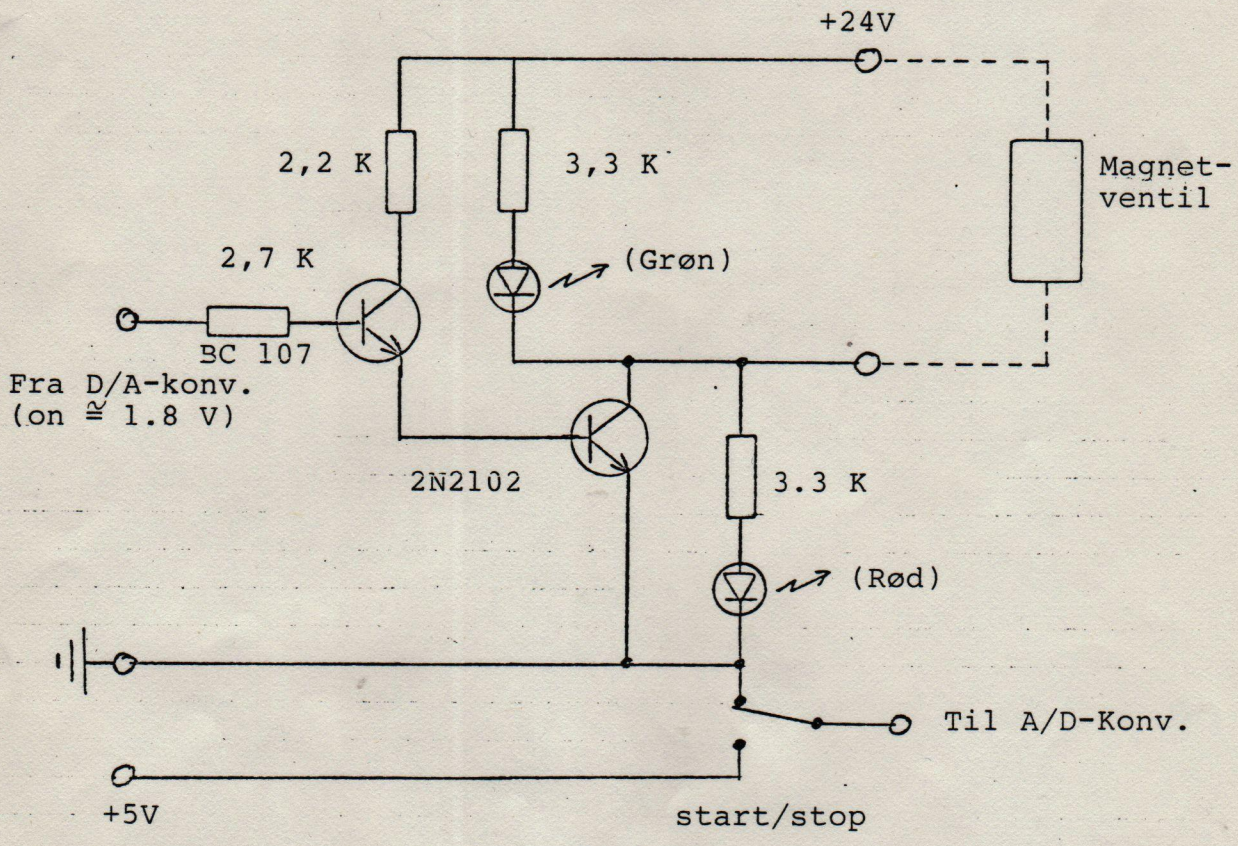
$H_T = 2 + \frac{5.7}{5} VS$ Tømning

Så vidt skydemodstanden. Magnetventilen er en ventil der aktiveres ved at sende en strøm I_M = 250 mA ved 24 V = igennem en spole på siden af magnetventilen.

Da det er åbning og lukning af denne ventil vi skal styre som funktion af spændingen over skydemodstanden, må vi have en forstærkning af signalet fra D/A-Converteren til magnetventilen, da Converteren jo maximalt kan afgive ca. 1 mA ved 10.0 V. Til dette brug indskydes en "black-box" med følgende indhold: (Se næste side).

$$\boxed{H_{vaskestand} = \underset{Vs=0}{H_o} + \underset{spænding}{k \cdot Vs}}$$

Niveau =


+24V
2,2 K
3,3 K
Magnet-
ventil
2,7 K
(Grøn)
BC 107
Fra D/A-konv.
(on ≅ 1.8 V)
2N2102
3.3 K
(Rød)
Til A/D-Konv.
+5V
start/stop

Figur 4.

Opgave:

Lav et program der kan indlæse det højeste og laveste tilladelige væskeniveau i beholderen (i cm), og som herefter styrer magnetventilen i overensstemmelse med de indlæste grænser. Endvidere skal man kunne starte og stoppe styringen v.h.a. en trykknap.

Til brug for løsningen kan man anvende procedurerne ADCONV og DACONV fra biblioteket ADDABIB, idet man først skriver LOAD ADDABIB og dernæst starter sin egen programmering fra linie 100 og fremefter.

Proceduren ADCONV aflæser A/D-Converteren, og DACONV sætter en spænding på en af D/A-Converter udgangene. Vil man f.eks. aflæse A/D kanal 2 skriver man blot

```
EXEC ADCONV (VPOT, 2)
```

hvorefter VPOT vil indeholde spændingen på kanal 2.

Vil man sende en spænding på 7.4 V ud på D/A-kanal 1 skrives blot

```
EXEC DACONV (7.4, 1)
```

Det skal bemærkes at argumenterne i et procedurekald skal have en værdi tildelt inden kaldet (må ikke være udefineret).

I Opstillingen benyttes A/D-kanal 0 til registrering af spændingen over skydemodstanden.

A/D-kanal 1 bruges til trykknappen.

D/A kanal 0 bruges til at give spænding til magnetventilen.

Forsøgsopstillingen er koblet sammen, så det drejer sig altså blot om at lave et program til styring af magnetventilen.

Indholdet af ADDABIB er vist herunder:

```
0001 //-----------------------------------------------------------------
0002 //  Procedure til aflæsning af Analog/Digital-Converteren
0003 //
0004 PROC ADCONV(REF V, KANAL#) CLOSED
0005  IF KANAL#<0 OR KANAL#>7 THEN KANAL#:=0
0006  V:=INP(KANAL#)*10.0/255
0007  IF KANAL#>3 THEN V:=V/100.0
0008 ENDPROC ADCONV
0009 //  Procedure til aktivering af Digital/Analog-Converteren
0010 //
0011 PROC DACONV(V, KANAL#) CLOSED
0012  K#:=KANAL#
0013  IF KANAL#<0 OR KANAL#>1 THEN K#:=0
0014  IF V<-10.24 OR V>10.24 THEN V:=0
0015  V#:=ROUND(ABS(V*200.0))
0016  VL#:=V# MOD 256
0017  VH#:=V# DIV 256
0018  IF V>=0 THEN VH#:=VH#+8
0019  OUT K#*2+8, VL#
0020  OUT K#*2+9, VH#
0021 ENDPROC DACONV
0022 //  Procedure til tidsforsinkelse (1..1000) sec.
0023 //
0024 PROC FORSINK(TID) CLOSED
0025  HTID:=TID
0026  IF HTID<1 OR HTID>1000 THEN HTID:=1
0027  FOR SEKUND:=1 TO HTID DO
0028   FOR TÆLLER:=1 TO 1000 DO
0029   NEXT TÆLLER
0030  NEXT SEKUND
0031 ENDPROC FORSINK
0032 //-----------------------------------------------------------------
```

Ved udformning af løsningen skal man huske på, at styringen skal kunne startes og stoppes uanset hvor i programmet man befinder sig.

L.B.

INDHOLDSFORTEGNELSE.

		Side
2.	INDLEDNING	2.1
3.	INTRODUKTION	
3.1	Hvorfor lære EDB?	3.1
3.2	Hvad betyder EDB?	3.1
3.3	Beskrivelse af et datamatisk system	3.2
3.4	Anvendelsesområder for EDB	3.2
3.5	Eksempel på anvendelse af EDB i laboratoriet	3.5
4.	DATAMATENS PRINCIPIELLE OPBYGNING	4.1
5.	YDRE ENHEDER OG DATAMEDIER	5.1
6.	LØSNINGSBESKRIVELSESMETODER	
6.1	Blokdiagrammer	6.1
6.2	Trinvis programforbedring	6.10
7.	PROGRAMMERING OG AFPRØVNING	
7.1	Programmering i COMAL-80	7.1
7.1.1	Sætninger	7.3
7.1.2	Variable	7.5
7.1.3	Konstanter	7.13
7.1.3.1	Talkonstanter	7.14
7.1.3.2	Tekstkonstanter	7.15
7.1.4	Sætningstyper	7.15
7.1.4.1	Tildelingssætninger og aritmetiske udtryk	7.15
7.1.4.2	Tildelingssætninger med tekstvariable	7.20
7.1.4.3	Læsesætninger	7.21
7.1.4.4	Skrivesætninger	7.28
7.1.4.5	Betingede sætninger	7.31
7.1.4.6	Repeterende sætninger	7.41
7.1.4.7	Procedurer og funktioner	7.47
7.1.4.8	Eksempel på større program	7.53

		Side
7.2	Afprøvning af programmer	7.55
7.2.1	Skrivebordstest	7.55
7.2.2	Indsættelse af skrivesætninger	7.60
8.	PROBLEMANALYSE	8.1
8.1	Løsningsmuligheder og fakta	8.3
8.1.1	Brugerens nuværende situation	8.4
8.1.2	Problemformulering	8.5
8.1.3	Eksempel på problemanalyse	8.5
8.2	System forslag	8.8
8.3	Interview-teknik	8.8
9.	FILBEHANDLING	9.1
9.1	Hvad er en fil	9.2
9.2	Hvor gemmes en fil	9.2
9.3	Ydre enheder som filer	9.3
9.4	Opdeling af filer i fil-typer	9.4
9.5	Filmanipulering	9.5
9.5.1	Oprettelse af filer	9.6
9.5.1.1	Filnavne	9.7
9.5.1.2	Navnekonflikt ved oprettelse	9.7
9.5.1.3	Pladsmangel på baggrundslager	9.8
9.5.2	Skrivning af filer	9.8
9.5.2.1	Hvordan skrives en fil	9.9
9.5.3	Læsning af filer	9.10
9.5.3.1	Hvordan læses en fil	9.11
9.5.4	Sletning af filer	9.12
9.5.4.1	Hvordan slettes en fil	9.13
9.5.5	Ændring af filers indhold	9.14
9.5.5.1	Hvordan ændres en fils indhold	9.14
9.5.6	Kopiering af filer	9.15
9.5.6.1	Hvordan kopieres filer	9.17
9.5.7	Beskyttelse af filer	9.19
9.5.7.1	Hvordan beskyttes filer	9.20
9.5.8	Aflæsning af filstatus	9.21
9.5.8.1	Hvordan aflæses filstatus	9.21

		Side
9.5.9	Aflæsning af filkataloget	9.22
9.5.10	Oversættelse af programmer	9.23
9.5.10.1	Hvordan oversættes programmer	9.24
9.5.11	Sammenkædning af flere programmer	9.25
9.5.11.1	Hvordan sammenkædes programmer	9.27
9.5.12	Kørsel af programmer	9.28
9.6	Fejludskrift ved filbehandling	9.29
10.	PRAKTISKE ANVENDELSER	10.1
10.1	Potentiometrisk titrering	10.4
10.2	Dosering af væsker	10.5
10.3	Niveaumåling	11.1
11.	Stikordsregister	
APPENDICES		
A:	Opstart af Comet 3000	
B:	Opgaver	
C:	Opgaveløsninger	
D:	Ordliste	
E:	Regning med binære tal	
F:	Kodeformer på datamedier	
G:	Reserverede ord i Comal-80	

2. INDLEDNING.

Indholdet i dette kompendium har til formål at give læseren en forståelse for de grundlæggende principper/begreber ved elektronisk databehandling og sætte vedkommende i stand til at udforme, afprøve og køre enkle programmer, med henblik specielt på skrivning og læsning af data på baggrundslager, som fortrinsvis tager sigte på anvendelse inden for laboratoriearbejdet. Endvidere vil læseren få en rimelig viden om, hvilke faciliteter man kan forvente i et datamatisk system. Gennem deltagelse i de til kompendiet hørende øvelser, bliver deltagerne fortrolige med det at kommunikere med en datamaskine via en terminal.

Som eneste forudsætning for at læse kompendiet, kræves et vist kendskab til de aritmetiske grundbegreber.

Kapitel 3 vil ganske kort introducere begrebet EDB og datamatisk system, og gennem eksempler vise, hvad EDB bl.a. bruges til rundt omkring i verden.

Kapitel 4 beskriver en datamaskines principielle opbygning og virkemåde gennem en simpel skuffe-model, hvor en skuffe vil være identisk med det senere introducerede begreb en "variabel". Har man først forstået skuffe-modellen, vil det være en stor hjælp til indlæring og forståelse af et sætningsopbygget sprog som COMAL-80.

Afsnittet vil endvidere beskrive de forskellige program-niveauer man arbejder med på en datamaskine (mikrokode, maskinkode, assembler o.s.v.). Til sidst vil der være en sammenligning mellem skuffemodellen og en virkelig datamat.

Kapitel 5 omtaler de almindeligste ydre enheder og medier, der er tilkoblet datamaten. Enhedernes virkemåde gennemgås ganske kort, og eksempler på kodning og de til enhederne hørende medier angives. Hvis man er mere end almindelig interesseret i kodeformerne, kan man ved at slå op i appendix F studere fænomenerne nærmere.

Kapitel 6 vil indføre læseren i metoder til beskrivelse af en løsning, således at denne beskrivelse vil kunne danne grundlag for udarbejdelse af et program. Der er her tale om beskrivelse v.h.a. et blokdiagram og beskrivelse v.h.a. et pseudo-programmeringssprog hvor programmet fra at udtrykke en komplex handling (f.eks. sortér data), forfines trinvis til at udtrykke simplere entydige handlinger.

Kapitel 7 beskriver dele af programmeringssproget COMAL-80 og omfatter tal og tekstvariable samt sætningstyperne: tildelingssætninger (A:=), læsesætninger (READ-DATA, INPUT og INPUT FILE), skrivesætninger (PRINT, PRINT FILE), betingede sætninger (IF, IF-THEN-ELSE, IF-THEN-ELIF.....ELSE-ENDIF, CASE-OF-WHEN... OTHERWISE-ENDCASE) og repeterende sætninger (FOR-TO-STEP-DO-NEXT , WHILE-ENDWHILE og REPEAT-UNTIL). Gennemgangen af stoffet lægger vægt på indlæringen og forståelsen for programmeringssprog, hvorfor raffinerede eksempler m.v. må henlægges til selvstudium af COMAL-80 manualen, der forøvrigt er et udmærket opslagsværk når først man er blevet fortrolig med sprogets opbygning og virkemåde.

Kapitel 8 behandler problemanalyse, d.v.s. vi her prøver at belyse hvordan man griber en given EDB-opgave i videste forstand an. De forskellige faser i problemanalysen/problemformuleringen stilles systematisk op, og belyses ved gennemgang af et eksempel. Forståelsen for et EDB-systems opbygning og virkemåde, erhvervet gennem de foregående afsnit, vil være en fordel ved læsning af dette afsnit.

Kapitel 9 omhandler filbehandling, hvor først begreberne en "fil" og "data" defineres, hvorefter forskellige måder for filhåndtering gennemgås (læsning af filer, skrivning af filer, beskyttelse af filer m.v.). Herudover giver kapitlet læseren en orientering om hvad man med rette kan forvente af faciliteter i et brugervenligt EDB-system, og forklarer den principielle virkemåde og brug af disse faciliteter. Til sidst er tydning af EDB-systemers fejlmeddelelser angivet.

Kapitel 10 beskriver nogle eksempler på anvendelse af en datamat til styring af en proces. Eksemplerne er skematiske og skal blot fungere som inspiration til brugen af datamater i laboratoriemilieuet. For hvert eksempel er der en skitse og et pseudoprogram, der antyder hvorledes et program til styring af den pågældende proces kunne opbygges. Programmeringsdetaljer må henlægges til studium af forhandlerspecifikke manualer, over det forskellige apparatur.

Kapitel 11 indeholder et stikordsregister med sidehenvisninger.

Bagest i kompendiet er indsat appendix A til F der omhandler følgende:

A: Opstart af cometen.
B: Opgaver til kapitlerne.
C: Løsningsforslag til opgaverne.
D: Ordliste med EDB-fagord.
E: Regning med binære tal.
F: Kodeformer på data-medier.

3. INTRODUKTION.

3.1. Hvorfor lære EDB?

Udviklingen af EDB er i løbet af de sidste 10 år foregået så hurtigt, at EDB-begrebet for mange har fået et skær af mystik om sig. Denne mystik har medført, at mange mennesker føler, at vi bevæger os mod et samfund, hvor det ikke længere er menneskene, men maskiner, der styrer udviklingen.

For at eliminere denne mystik er det vigtigt, at så mange mennesker som muligt får indsigt i EDB-verdenen med alle dens nye begreber og arbejdsmetoder.

Flere og flere mennesker anvender EDB på deres arbejdsplads, men uden egentlig at vide hvad det er, der foregår, og dette kan let blive frustrerende. Selv for en teknisk minded person som en laborant, der anvender EDB til styring af analyseringsapparatur og til beregninger, kan det være uigennemskueligt, hvad der egentlig foregår. Derfór er det vigtigt for en laborant at have en baggrundsviden og en indsigt i EDB, således at laboranten evt. selv kan gribe styrende ind i processerne.

3.2. Hvad betyder EDB?

Forkortelsen EDB betyder: Elektronisk Data Behandling.

For at forstå,hvad denne betegnelse står for, er det nødvendigt først at se på, hvad der menes med "data".

Inden for EDB vil data udelukkende gælde oplysninger, der er omsat til tal og bogstaver, altså oplysninger, som på den ene eller anden måde ønskes behandlet.

Elektronisk databehandling dækker over databehandling, som udføres af maskiner baseret på elektronik til forskel fra f.eks. manuel databehandling, hvor databehandlingen sker ved brug af papir og blyant.

De elektroniske maskiner som udfører databehandlingen går under mange navne: datamaskine, datamat, edb-maskine, elektronhjerne, minidatamat og microdatamat. I denne bog bruges den generelle betegnelse "datamat" for disse maskiner.

3.3. Beskrivelse af et datamatisk system.

Et simpelt datamatisk system er opbygget af mindst 4 bestanddele: en inddata-enhed, en datamat, en uddata-enhed og programmer.

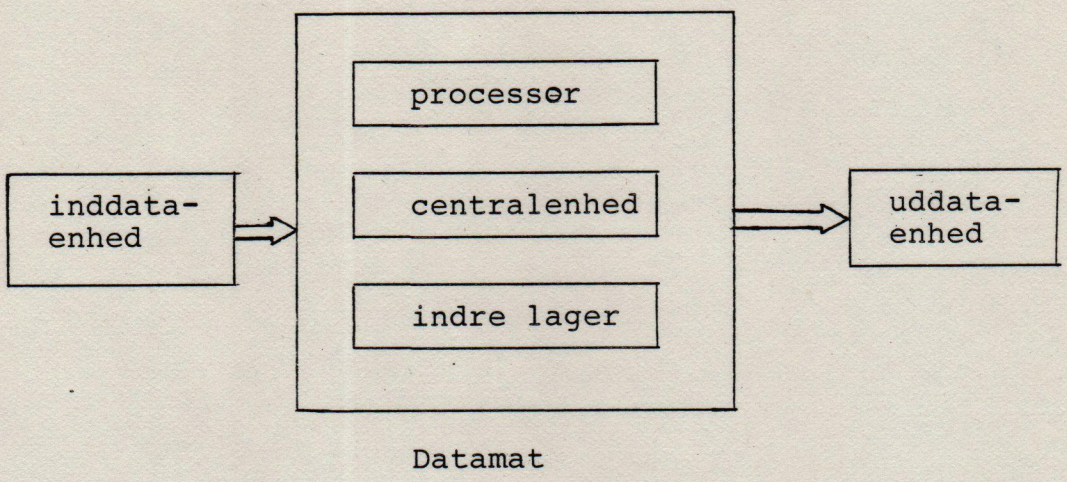


Fig.3.1. Et datamatisk system.

Inddata-enheden modtager vores data, hvorfra de hentes af datamatens processor. Derefter sendes de til centralenheden og til sidst gemmes de i det indre lager.

Databehandlingen styres af centralenheden. Først henter centralenheden inddata fra lageret, hvorefter den foretager databehandlingen og sender resultatet (uddata) tilbage til lageret.

Til sidst henter centralenheden uddata fra lageret, og de sendes via processoren til uddata-enheden.

Den sidste bestanddel i systemet er de programmer, som er nødvendige for at den netop beskrevne proces med indlæsning, databehandling og udlæsning kan foregå. Programmerne indlæses og lagres i det indre lager og kan evt. udlæses igen. De består af en række ordrer opstillet på en måde, som løser vores problemer.

Da programmer skal indlæses og lagres, betyder det, at data ikke bare er tal, der skal foretages en beregning på, men at data også kan være programmer.

3.4. Anvendelsesområder for EDB.

Når vi betragter EDB, er det klart, at det tekniske udviklingsstade betinger hvilke former for databehandling, der kan

komme på tale. Men hvis man i dag skal vurdere, hvor langt den teknologiske udvikling er nået inden for edb-området, kan man vel sige at udviklingen er kommet så langt, at det,der sætter begrænsningen for anvendelse af EDB-teknikken, er brugerens fantasi. Maskinerne i sig selv sætter ingen begrænsninger; hvis et problem kan formuleres i ord eller formler, vil det også kunne løses af en datamaskine. Det eneste der kræves er, at der findes en fastlagt løsningsmetodik.

Nogle vigtige områder, hvor man i dag anvender EDB, er følgende:

1. Administration
2. Kommunikation
3. Styring
4. Simulering
5. Kunstig intelligens
6. Grafisk databehandling
7. Undervisning
8. Spil

Administrativ databehandling omfatter blandt andet: lønberegning, skatteregnskab, momsopgørelse, lageropgørelse, føring af bogholderi, ressourceberegning m.m.. En af årsagerne til,at EDB har vundet indpas ved den administrative databehandling, er, at der er tale om store mængder af ens data, som skal underkastes den samme behandling. Desuden kan anvendelse af EDB bevirke en reduktion i antallet af ansatte.

Ved kommunikation mellem to fjerntliggende punkter fremkommer der meget let støj, så det er svært eller umuligt for modtageren at tyde meddelelsen. Men ved brug af EDB er det muligt af frasortere støjen og få tydet meddelelsen. Et eksempel på dette er kommunikation mellem en satellit og en jordstation.

Ved styringsanvendelser er det vel nok rumfarten, der tydeligst illustrerer anvendelse af EDB. Både startraketten og månelandingsfartøjet er forsynet med datamater, der på brøkdele af sekunder kan korrigere kursen m.v.. Desuden spiller automatisk styring af maskiner en vigtig rolle inden for industrien, da det med-

fører præcisionsarbejde, personalereduktion og evt. nedsat berøring med sundhedsskadelige stoffer.

Den seneste udvidelse af styringsområdet er sket med de såkaldte processtyringsopgaver, hvor EDB anvendes til at overvåge en industriel proces. Processtyringen bygger på, at der bliver foretaget en række fortløbende målinger, hvor resultatet af målingerne bliver behandlet via EDB, som derefter styrer eventuelle indgreb, hvis processen antager et utilsigtet forløb.

EDB kan også anvendes til simulering, f.eks. flysimulering ved optræning af piloter. Piloten anbringes i en kabine udstyret med instrumentbræt og en grafisk skærm, der viser situationsbilleder, som svarer til pilotens udsigt. Når instrumentbrættet betjenes, modtages dets signaler af et EDB-anlæg, som afhængig heraf danner en serie nye billeder svarende til flyets reaktion.

Det er også muligt at anvende en datamat som kunstig intelligens. For eksempel kan man få differentieret et udtryk symbolsk indeholdende ubekendte.

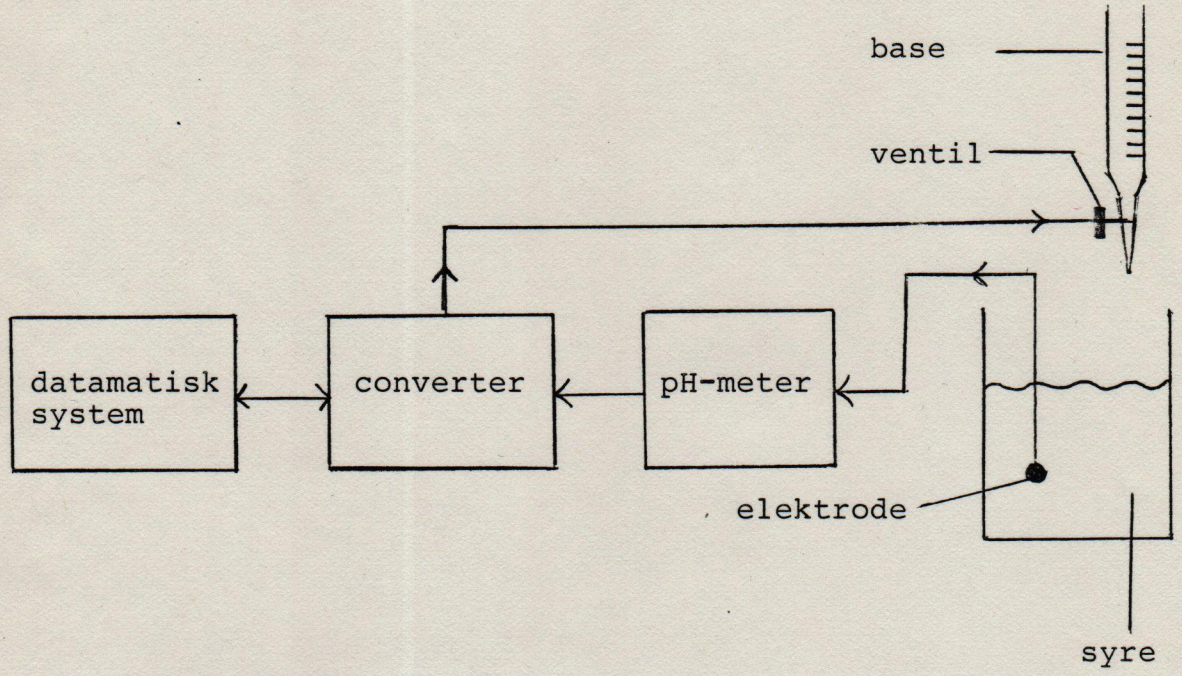
Grafisk databehandling er et stort område inden for EDB. Et grafisk databehandlingssystem består af en række komponenter, som giver systemet mulighed for at danne, opbevare og tegne grafiske informationer. For eksempel kan grafiske kommandoer omsættes til tegning på en tv-skærm. Det er vigtigt at bemærke, at et billede, som i virkeligheden er 3-dimensionel kun afbildes 2-dimensionelt med en 3D-simulering, f.eks. ved perspektivisk afbildning, rotation, skyggelægning m.v..

Undervisning ved hjælp af datamaskiner er baseret på en dialog mellem elev og datamat. Datamaten er programmeret til at behandle det område, der ønskes undervist i, og undervisning vil så foregå på den måde, at maskinen stiller eleven et spørgsmål, eleven svarer, maskinen modtager svaret og kontrollerer det, og afhængig af svarets rigtighedsgrad stilles et nyt spørgsmål, o.s.v..

Inden for underholdningsbranchen anvendes EDB til forskellige spil, f.eks. tennis og nedskydning af flyvemaskiner. Oprindelig blev de forskellige krigsspil benyttet til at træne soldaters reaktionsevner, men er så blevet overtaget af forlystelsesbranchen.

3.5. Eksempel på anvendelse af EDB i laboratoriet.

Figur 3.2. viser, hvordan man i laboratoriet kan benytte et datamatisk system til styring af et fysisk system.



Figur 3.2.

Det fysiske system består her af en simpel opstilling til en potentiometrisk titrering bestående af: et bægerglas med syre, en kombielektrode forbundet til et pH-meter samt en burette indeholdende en basisk titrervæske.

pH-meteret er forbundet med det datamatiske system via en converter, som er i stand til at oversætte analysesignalerne til digitale signaler og omvendt.

Inden titreringen starter,giver vi det datamatiske system besked om, at titreringen f.eks. skal stoppe, når pH $\geq$ 8,7. Så startes titreringen, som udføres efter følgende trin:

1: Visning af pH-værdi på pH-meter.
2: Converteren oversætter pH-værdien til en form, der kan opfattes af det datamatiske system.
3: Datamatiske system sammenligner den målte pH-værdi med den opgivne slutværdi og svarer,om burettehanen skal åbnes

eller lukkes.

4: Converteren oversætter svaret, hvorefter hanen åbnes eller lukkes.

I en almindelig potentiometrisk titrering sker styringen ved hjælp af en titratorenhed, som indeholder en microprocessor.

4. DATAMATENS PRINCIPIELLE OPBYGNING.

En datamat er en maskine, som kun kan bringes til at fungere ved at vi "fortæller" den hvad den skal lave, i den rækkefølge instruktionerne (ordrerne) skal udføres.Det,at fortælle en datamat hvad den skal gøre, kaldes at programmere den, hvilket for de fleste mennesker kan volde lidt problemer i starten, idet man normalt ikke tænker over tilværelsen i små detaljer. Tænk blot på det at gå til købmanden efter 4 øl. Hvis dette skulle beskrives for en datamat ville det blive noget i retning af:

1) Rejs dig fra lænestolen.
2) Gå ud i køkkenet.
3) Er der nogle tomme flasker?
4) Hvis ja! tag dem.
5) Gå ned på gaden.
6) Har du husket pengene?
7) Hvis ja! Hop til punkt 11.
8) Gå op igen.
9) Tag pengene.
10) Hop til punkt 6.
11) Gå til købmanden.
12) Køb 4 øl.
13) Gå tilbage til lejligheden.
14) Sæt dig i lænestolen
15) Drik dine øl.
16) Hop til punkt 1 .

Ja, sikken tilværelse!

Ovenstående "program" skal vise, at en datamat for at fungere skal have alle nødvendige informationer til den mindste detalje. Hvis vi bevæger os ind i apparatet en datamat, vil vi se en masse elektroniske "dimser" der alle er med til at danne grundstammen i enhver datamat. Nede på dette niveau vil al kommunikation foregå v.h.a. elektriske signaler, hvis udseende f.eks. kunne være som vist i fig. 4.1.

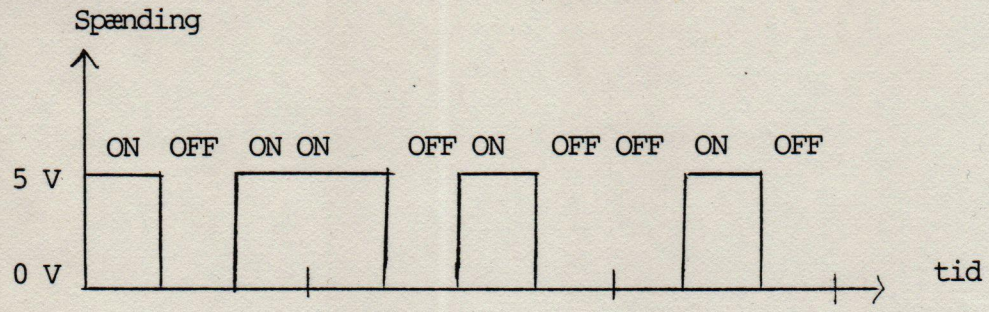


fig. 4.1. Elektrisk signal.

Karakteristisk for disse signaler er, at spændingen enten er 5 volt eller 0 volt vekslende i tidens løb. Der er altså tale om signaler, hvor kun to tilstande er mulige nemlig "ON" svarende til 5 volt og "OFF" svarende til 0 volt. Grunden til at man kun arbejder med to tilstande er, at de elektroniske kredse er opbygget af transistorer på en sådan måde at der kun eksisterer to stabile logiske tilstande (som et relæ).

Tilstanden "ON" kaldes også for "1" og tilstanden "OFF" for "0", hvorfor signalet i fig. 4.1 kan oversættes til en række nuller og et-taller som følger

lolloloolo

hvilket repræsenterer et tal i et talsystem der kaldes to-talssystemet eller det binære talsystem, der til forskel fra ti-talssystemet udmærker sig ved kun at anvende cifrene 0 og 1. Hvert ciffer i et binært tal kaldes en bit, og forekomster af bit kaldes derfor også for bitmønstre. Med andre ord kan man sige at kommunikation nede på elektronikniveau foregår v.h.a. bitmønstre. Et bitmønster kan til enhver tid omregnes til et tal i lo-talssystemet idet 2-talssystemet er opbygget af 1'ere, 2'ere, 4'ere, 8'ere , 16'ere, 32'ere o.s.v., hvor 10-talssystemet er opbygget af 1'ere, 10'ere, 100'ere, 1000'ere o.s.v.. F.eks. vil det binære tal 10111 kunne omskrives til

$$1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 23$$

i ti-talssystemet. Der vil i appendix E være en detaljeret gennemgang af det binære talsystem, samt regning med binære tal.

For at få en fornemmelse af hvad kommunikation med binære tal er for noget og for, hvad der egentlig er inde i mysteriet en datamat, har vi i fig. 4.2a skitseret den principielle opbygning af en datamat og i fig. 4.2b angivet hvilke simple ordrer den er

i stand til at udføre. Der er selvfølgelig her tale om en forsimpling af virkeligheden, idet der f.eks. mangler angivelse af hvorledes de forskellige ordrer bliver effektueret i datamaten, hvilket dog ikke betyder noget for forståelsen af det at programmere.

Den skitserede datamat kan opfattes som bestående af et antal "skuffer" (angivet med fede typer på figuren) nemlig skufferne I, R, U, X, Y og skufferne nummereret fra 0-7, samt af en indlæsnings-, en udlæsnings, en regne- og en nulstillingsenhed. Skufferne og de nævnte enheder er forbundet med hinanden som vist med pile, hvilket betyder, at transport mellem de forskellige komponenter kun kan foregå i pilens retning, og da kun når bestemte af de i cirkel med små tal angivne porte åbnes. Hvad transporten angår kan man forestille sig, at der her er tale om ilægning og udtagning af sedler med tal på fra skufferne, hvor en seddel kun kan indeholde et tal af gangen. Der gælder endvidere den regel at en seddel der en gang er lagt i en skuffe forbliver i skuffen uanset hvor mange gange der tages kopier af den, indtil en ny seddel lægges i den pågældende skuffe, hvorved det gamle indhold destrueres. Det vil med andre ord sige at en skuffe kun kan indeholde én seddel af gangen.

Lad os prøve at se på et par eksempler vedrørende programmering af vores datamatmodel. Hvis vi kalder skuffe 2 for A, og gerne vil lægge tallet 19 i A (altså operationen A=19) kan dette gøres med følgende ordresæt:

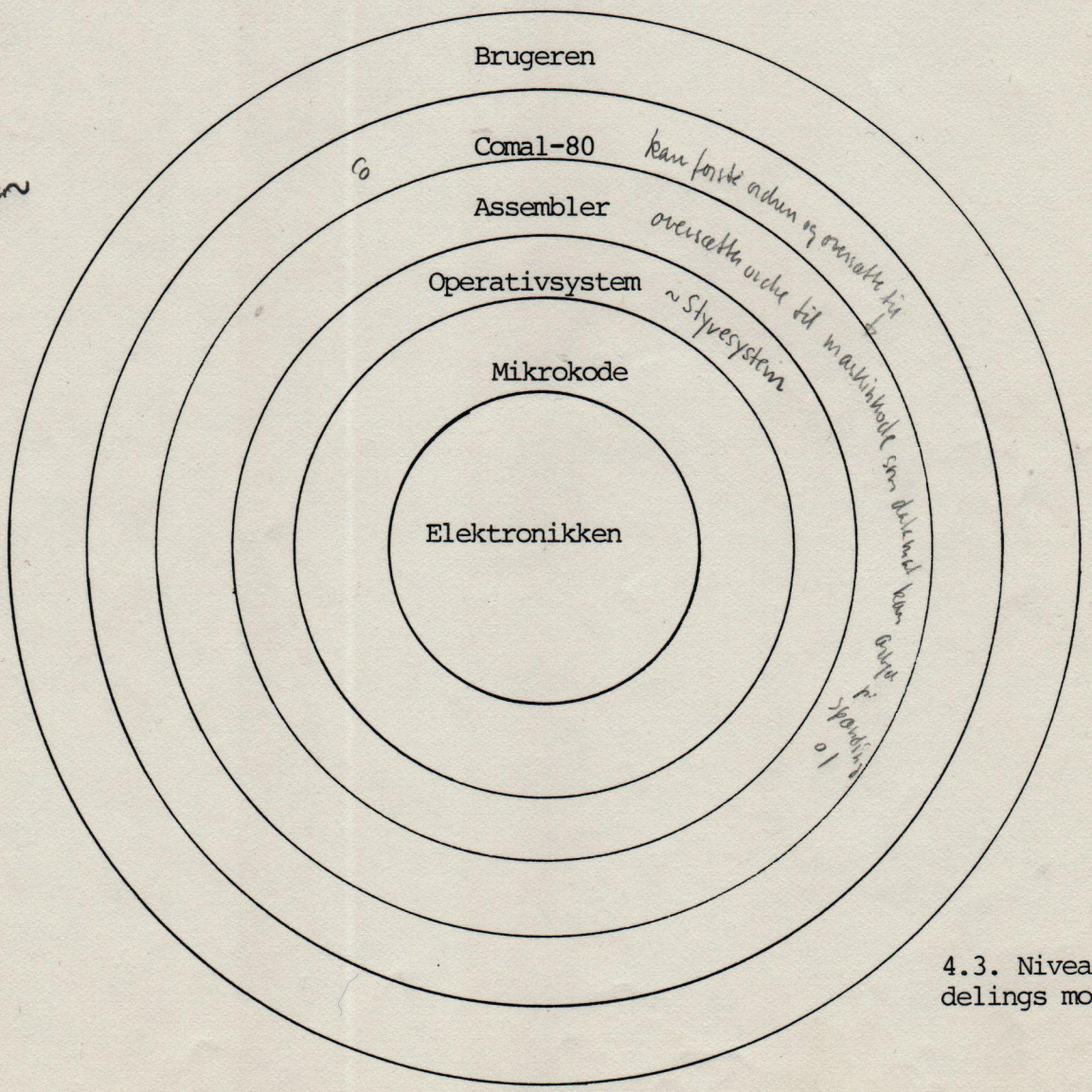
```
1: MVI,   19  svarende til 1,19
2: MIS,    2  svarende til 3,2
```

hvor MVI, 19 siger læg 19 i skuffe I og MIS, 2 siger læg indholdet af skuffe I i skuffe 2. Et andet eksempel kunne være, at vi ønskede at udføre operationen C=A+3 hvor C f.eks. er skuffe 4. Dette kunne gøres med ordrefølgen:

```
1: MVI, 3    ; læg 3 i I skuffen.
2: MIS, 0    ; læg indholdet af I i skuffe 0.
3: LDX, 0    ; læg indholdet af skuffe 0 i skuffe X.
4: LDY, 2    ; læg indholdet af skuffe 2 i skuffe Y.
5: ADD       ; læg X og Y sammen i skuffe R.
6: MRS  4    ; læg indholdet af R i skuffe 4 (=C).
```

I dette eksempel skulle der altså 6 ordrer til at udføre den ønskede operation. Det er ikke på nuværende tidspunkt svært at forestille sig, at selv et ganske simpelt problems løsning kunne gå hen og fylde en hel del ordrer i dette primitive sprog som det jo er. Da de første datamater så dagens lys havde man ikke engang mulighed for at angive ordrene som f.eks. ADD, LDX, o.s.v., men måtte programmere v.h.a. lange rækker af tal. Dette er man forskånet for i nutidens Danmark, idet man har konstrueret nogle avancerede sprog hvor man f.eks. blot behøver at skrive C=A+3 hvorefter en tolk (der er et program) oversætter udtrykket til den mere simple notationsform som før angivet. På en måde kan man jo sige, at det minder om de ting der sker, når f.eks. en englænder og en russer der ikke forstår hinanden skal tale sammen via en tolk. Af disse mere avancerede sprog skal her blot nævnes 3 som eksempel nemlig: FORTRAN, PASCAL og COMAL-80, hvoraf vi senere vil beskæftige os indgående med sidstnævnte.

Hvis vi kigger på niveau-opdelingen (hierakiet) i en datamat kan det skitseres som vist i fig. 4.3.



4.3. Niveau-delings model

Her har vi for at starte med det inderste niveau elektronikken, der arbejder med bitmønstre som tidligere nævnt. Næste niveau microkoden er en samling af ordrer, der som regel anvendes til simulering af manglende egenskaber ved elektronikken (f. eks. kan der mangle en "dims" til multiplikation af to tal, hvis virkning så simuleres i mikrokoden). Operativsystemet er et program, der styrer al kommunikation mellem datamaten og dens ydre enheder, (f.eks. en linieskriver), således at man f.eks. ved udskrivning af en tekst på linieskriveren ikke behøver at bekymre sig om, hvornår en linie er skrevet til ende så næste linie kan udskrives o.s.v.. Det operativsystem vi i dette kompendium vil referere mest til vil være CP/M-operativsystemet. Næste niveau Assembleren er et program, der samler et programs forskellige dele. F.eks. vil det være sådan, at en skriveordre i COMAL-80 i virkeligheden vil bestå af en række småprogrammer, som assembleren automatisk vil indsætte som byggeklodser i konstruktionen af et køreklart program. Dette indebærer at operativsystem og assembler er tæt knyttet til hinanden. COMAL-80 betyder som regel programmeringssproget COMAL-80 Men i forbindelse med omstående tegning skal det opfattes som det program, der oversætter brugerens program til et simplere programmeringssprog (Assembler).

På yderste niveau har vi så brugeren, der selvfølgelig ikke er noget program, men er den person der skriver sætninger i programmeringssproget COMAL-80.

Ja, det var sandelig en ordentlig omgang! men tag det blot roligt; de niveauer vi skal arbejde med i de følgende afsnit, er de to yderste, altså brugeren og COMAL-80.

Når det ovenfor nævnes, at man skriver programmer, er det fordi teknikken endnu ikke er nået så vidt, at man blot ved talens brug kan kommunikere sikkert med en datamat. Forøvrigt kunne man derved få problemer som: "Doktor, min datamat forstår mig ikke".

Lad os endnu engang vende tilbage til fig. 4.2a hvor der ialt er angivet 24 små cirkler med tal i. Disse cirkler repræsenterer altså nogle porte, der på passende tidspunkter i en programkørsel skal lukkes op og i. Denne lukken op og i samt beregninger, flytning af data m.v. udføres altsammen af en enhed man kalder centralenheden. Denne enhed set i lidt større sammenhænge er vist i fig. 4.4 og for sig selv i fig. 4.5.

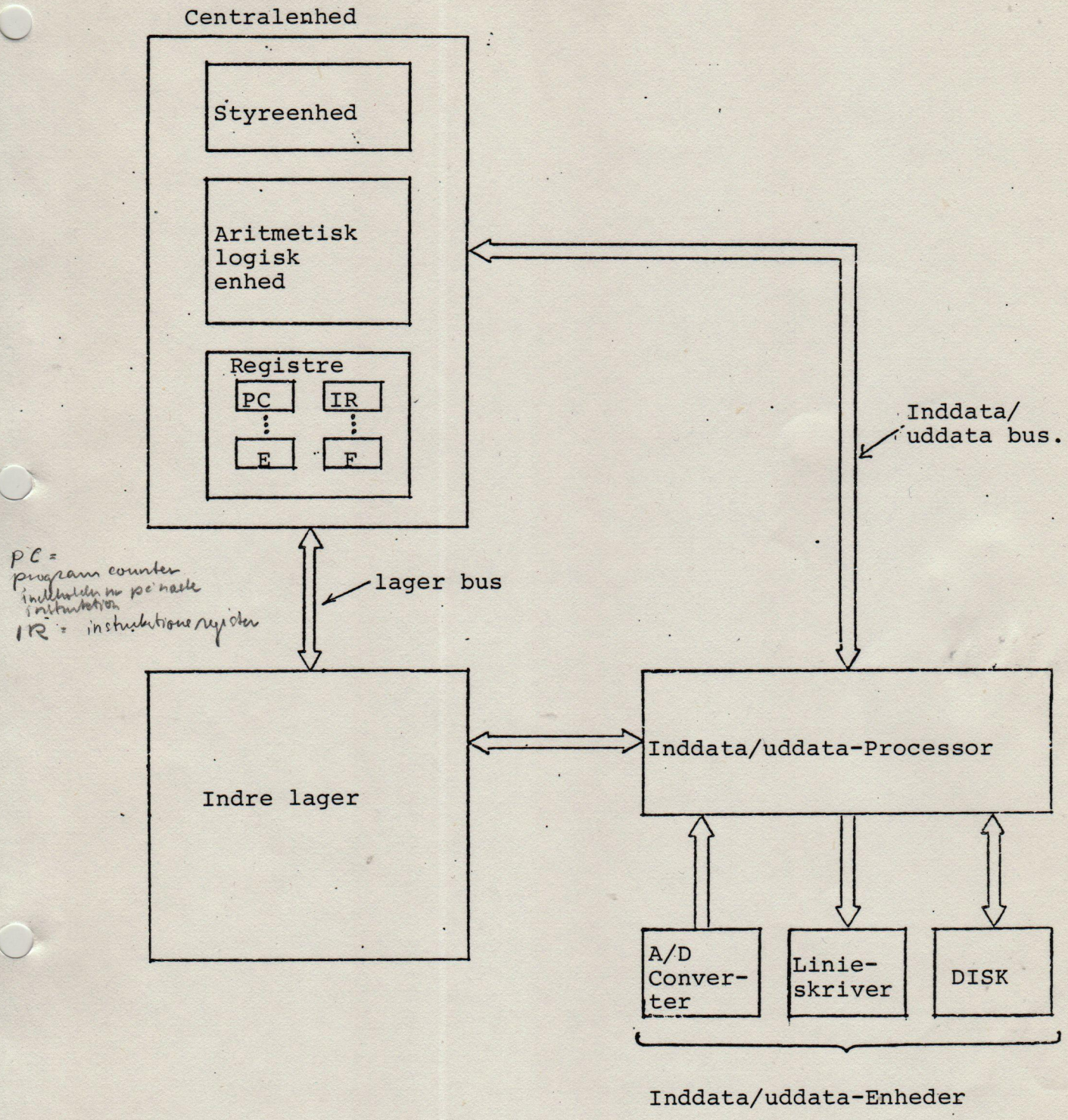


Fig. 4.4 viser hvordan en simpel datamat er organiseret med én centralenhed og én inddata/uddata-Processor.

Centralenheden er datamatens "hjerne", og har til opgave at udføre de programmer (i maskinkode) der ligger i det indre lager. Dette foregår ved først at hente (læse) en instruktion fra lageret, dernæst at undersøge om instruktionen er i orden og til sidst at udføre denne. Centralenheden består af en styreenhed der er ansvarlig for læsning af instruktioner samt fastlæggelse af instruktionernes type (bl. a. afhængig af antal operander), en aritmetisk-logisk enhed der udfører alle beregninger logiske som aritmetiske (f.eks. logisk AND og addition) og endelig et antal registre der anvendes til midlertidig lagring af resultater samt kontrol informationer. Af kontrolregistrene skal her blot nævnes to, nemlig programtælleren kaldet PC (eng. Program Counter), der er det vigtigste af registrene og har til opgave altid at pege på den næste instruktion, der skal udføres, og instruktions registret kaldet IR, der altid vil indeholde den instruktion, der er ved at blive udført. Det at udføre en instruktion er i virkeligheden en serie af ganske små trin der udføres af centralenheden på følgende måde:

1: Hent næste instruktion til instruktionsregistret (IR).
2: Sæt programtælleren (PC) til at pege på næste instruktion.
3: Bestem typen af instruktionen i IR.
4: Undersøg om instruktionen skal bruge data fra lageret, og hvis dette er tilfældet, find da ud af hvor i lageret dette data befinder sig.
5: Hvis instruktionen skal bruge data fra lageret, da hent det over i de interne registre.
6: Udfør instruktionen.
7: Gem resultatet hvor det nu skal lagres.
8: Gå til punkt 1 og begynd på næste instruktion.

Ja, der er jo ikke stor forskel på opbygningen af ovenstående "program" og opbygningen af programmet om hentning af 4 øl.

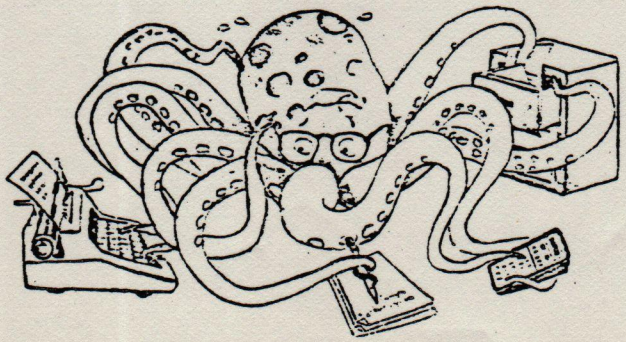


Fig. 4.5. Centra(v)l-enhed

Den på fig. 4.4 angivne inddata-/uddataprocessor er en elektronisk enhed, der letter arbejdet for centralenheden i dennes kommunikation med de ydre enheder. Betegnelsen "bus" er en betegnelse for de ledninger, der transporterer data mellem datamatens forskellige komponenter.

Hvis vi til slut sammenligner skitserne i fig. 4.4 og fig. 4.2a vil følgende dele fra de to tegninger være identiske.

fig. 4.4	fig. 4.2 a
Aritmetisk/logisk enhed	Regneenheden
Registre	X,Y og R skuffen
Indre lager	Skuffe 0-7
Noget af ind-/uddata processoren	Ind-/uddata skufferne
Inddataenhed	Inddata-enhed
Uddata-enhed	Uddata-enhed
Styreenhed	...(ikke angivet)....

Datamatmodellen i fig. 4.2a vil der blive refereret til mange gange i det øvrige kompendium.

5. YDRE ENHEDER OG DATAMEDIER.

Som det fremgik af fig. 4.2 b sker kommunikationen mellem datamaten og dens omgivelser gennem to ydre enheder: En inddata enhed og en uddata-enhed. Vores data skal derfor være repræsenteret på en sådan form, at de kan omsættes til elektriske impulser i inddata-enheden, og i uddata-enheden skal de elektriske impulser fra datamaten omsættes til en repræsentationsform, som vi kan bruge. Det er altså nødvendigt at anvende maskinlæsbare databærere, også kaldet datamedier, for at få indlæst data.

I det følgende gennemgås de mest anvendte datamedier og de tilsvarende enheder. Angående de koder, som nævnes, henvises til appendix.

Hulkortet er et stykke karton, som normalt indeholder 80 lodrette kolonner og 12 vandrette rækker. Som det fremgår af fig. 5.1 giver kombinationenaf huller en kode for et tegn (et bogstav eller et ciffer). Hullerne er anbragt efter Holleriths system (Hollerith er navnet på kodeformens opfinder), og øverst på kortet kan koderne oversættes til tegn af hensyn til de mennesker, der benytter kortet. Som inddata enhed anvendes en hulkortlæser, som omformer hullekoderne til elektriske signaler.

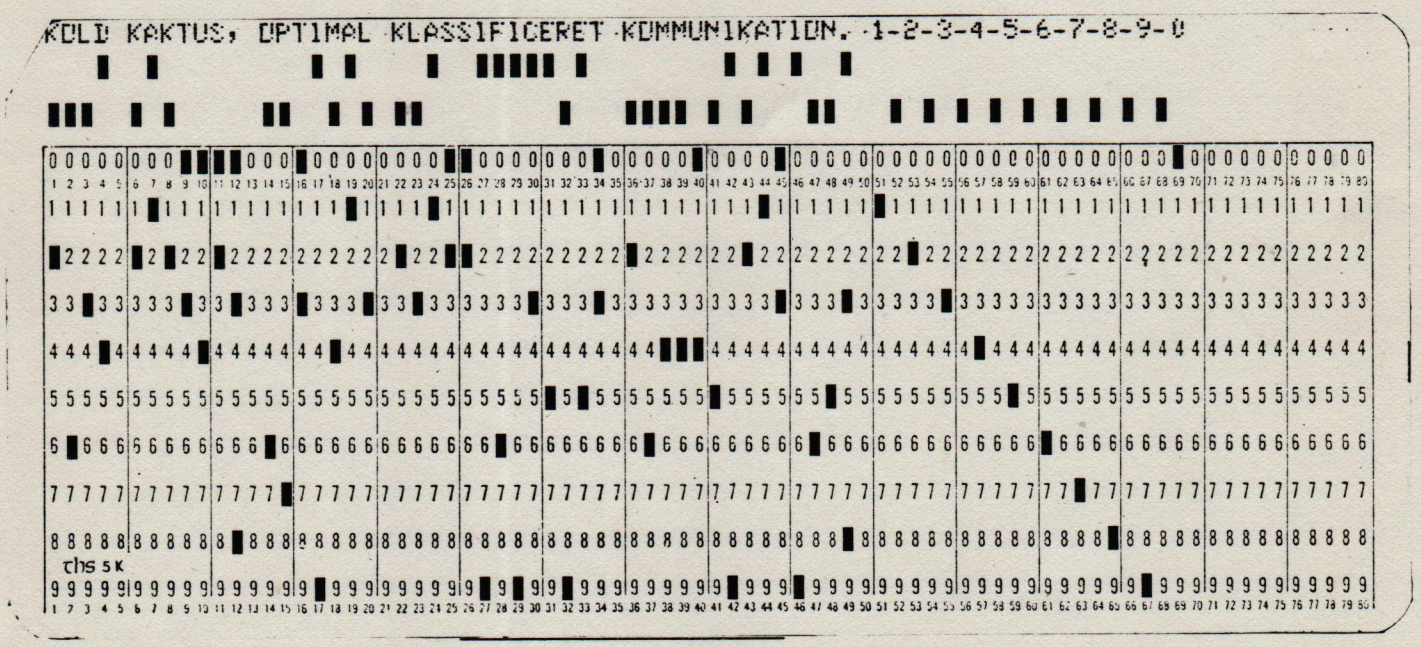
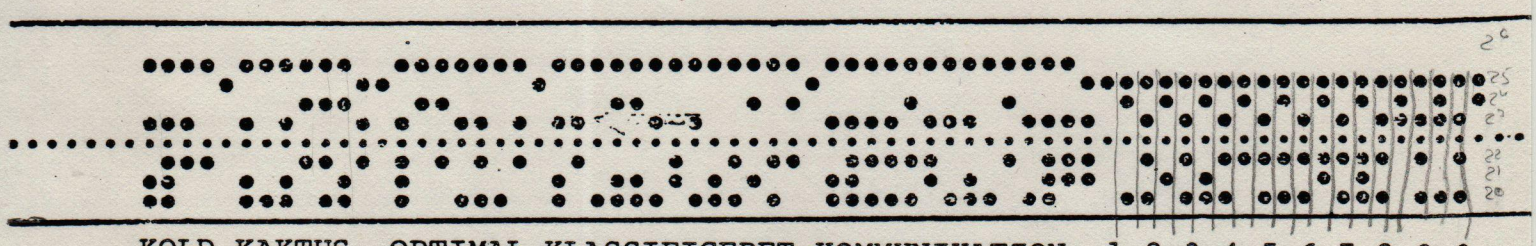


fig. 5.1. hulkort.

Det er også muligt at benytte hulkort som udlæsningsmedium. Den tilsvarende udlæsningsenhed kaldes en hulkorthuller, idet den omsætter datamatens elektriske signaler til huller.

Hulstrimmelen er en lang papirstrimmel. Ved hjælp af en række huller tværs over papiret dannes hulkombinationer, som repræsenterer et tegn (ciffer, bogstav og specialtegn). Kodningsformen kan være ASCII-kode eller EBCDIC-kode.

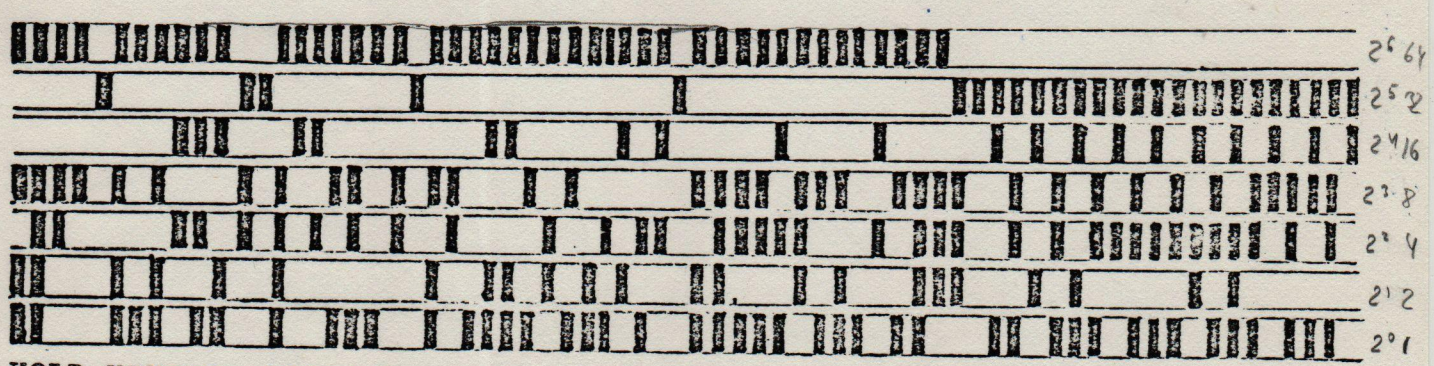


KOLD KAKTUS, OPTIMAL KLASSIFICERET KOMMUNIKATION. 1-2-3-4-5-6-7-8-9-0

Fig. 5.2. Hulstrimmel

Indlæse-enheden kaldes en strimmellæser, og uddata-enheden kaldes en strimmelhuller.

Magnetbåndet er lavet af plastik, hvor den ene side er belagt med et tyndt magnetiserbart materiale bestående af jernoxid. Data repræsenteres, ved at felter på båndet er magnetiserede. Som det fremgår af fig. 5.3. ligger disse felter på tværs af magnetbåndet, således at der til et bestemt magnetiseringsmønster på tværs af båndet, svarer et bestemt tegn. Princippet er altså det samme som ved hulstrimler, og der anvendes også de samme koder.



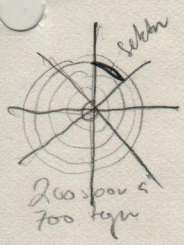
KOLD KAKTUS, OPTIMAL KLASSIFICERET KOMMUNIKATION. 1-2-3-4-5-6-7-8-9-0

Fig. 5.3. Magnetbånd.

Et magnetbånd af denne type er ca. dobbelt så bredt som et almindeligt båndoptagerbånd og er af en kraftigere kvalitet. Indlæsningen til datamaten foregår v.h.a. en båndstation, hvor båndet føres forbi et læsehoved og over på en "tom" spole. Båndstationen er også forsynet med et skrivehoved, hvilket betyder, at man også kan få skrevet uddata over på et magnetbånd.

Magnetbåndet anvendes også i en mindre udgave, som kaldes kassettebånd, fordi de af udseende ligner almindelige musikkasettebånd. Kasettebåndene er meget kortere end magnetbåndet, og kan derfor ikke lagre så mange tegn, som magnetbåndet.

Magnetplader ligner en almindelig LP-grammofonplade, men er belagt med et magnetiserbart lag på begge sider. Data lagres på koncentrisk beliggende spor og efter samme princip, som gælder for magnetbåndet. De store magnetplader anvendes både enkeltvis og i stabler (typisk 6-8 plader oven på hinanden), der så udgør et såkaldt magnetpladelager.



Ved brug er pladestabelen anbragt i en pladelagerstation, hvor der er et læsehoved for hver pladeside. Ved siden af læsehovederne er anbragt skrivehoveder, så magnetpladerne også kan anvendes som uddata-medie.

Disketter er en lille magnetplade på størrelse med en single grammofonplade og anvendes især i forbindelse med mindre datamater, og altid enkeltvis. Ind- og udlæsningen sker som ved magnetpladelageret, men maskinenheden kaldes en floppydisk, der ofte er bygget sammen med datamaten.

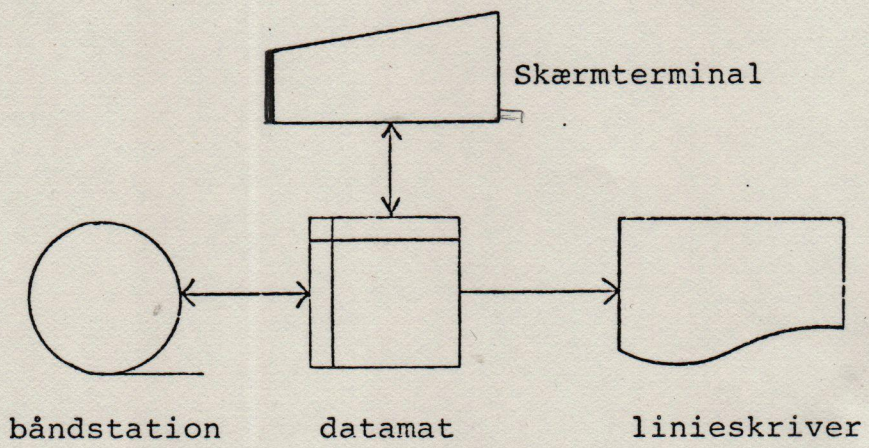
De nævnte datamedier har den ulempe, at koderne ikke umiddelbart kan læses af mennesker. Det betyder, at man f.eks. ikke kan kontrollere skrivefejl før indlæsningen, og at man ikke kan tolke uddata. For at bøde på dette kan man f.eks. benytte en TV-skærms overflade som medie.

Terminalen er forsynet med en TV-skærm og et tastatur, der minder om tastaturet på en skrivemaskine. Data skrives v.h.a. tastaturet og overføres til TV-skærmen hvor de fremkommer som almindelige tegn. Har man lavet en skrivefejl, kan man se det på skærmen og rette fejlen ved hjælp af en slettetast på tastaturet. Efter endt skrivning sendes data til datamaten ved at taste en bestemt ordre på tastaturet.

Kombinationen med TV-skærm og tastatur som indlæseenhed kaldes skærmterminal, hvor mediet altså er skærmens overflade. Skærmterminalen kan også anvendes som udlæseenhed, hvorved uddata direkte kan læses af mennesker.

Linieskriveren er en anden udlæsningsenhed, som gør det muligt, at uddata kan læses direkte af mennesker. Linieskriveren skriver uddata på papir (dokument) ligesom en skrivemaskine, men den er i stand til at skrive en hel linie ad gangen. Linieskriveren kaldes ofte for en printer (eng: print = trykke).

Det er værd at lægge mærke til at, kombinationen af ind- og uddataenheder kan vælges næsten frit, alt efter hvilke behov brugeren har. I figur 5.4. er skitseret med standardsymboler, hvordan et datamatisk system f.eks. kan være opbygget. Der henvises til næste side angående standardsymboler.

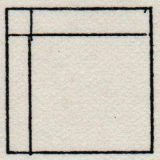
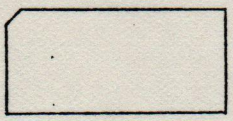
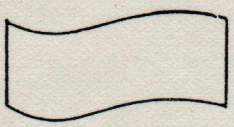
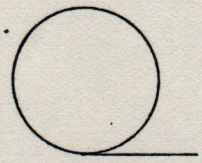
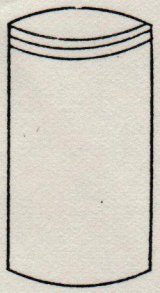
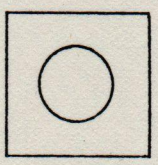
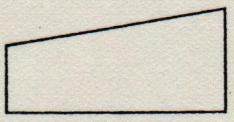
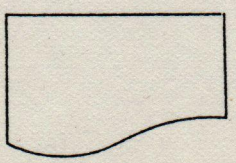


Figur 5.4.

Indlæsningen kan f.eks. foretages via skærmterminalen. Hvis skærmterminalen anvendes til udlæsning, er man nødt til at skrive uddata ned på papir, hvis en anden person står og venter på at komme til. Dette kan undgås ved at benytte skriveren til udlæsning, og samtidig har man mulighed for at gemme uddata på papiret.

Hvis uddata senere skal benyttes som inddata, vil man overføre dem til magnetbånd, hvorfra de senere kan indlæses igen.

STANDARDSYMBOLER FOR YDRE ENHEDER OG MEDIER.

SYMBOL	MEDIE	ENHED	
		INDDATA	UDDATA
	RAM og ROM lager	Centralenhed	Centralenhed
	Hulkort	Hulkortlæser	Hulkorthuller
	Hulstrimmel	Hulstrimmel-læser	Hulstrimmel-huller
	Magnetbånd	Båndstation	Båndstation
	Magnetplader	Pladelager-station	Pladelager-station
	Diskette	Floppydisk	Floppydisk
	TV-skærm	Skærmterminal	Skærmterminal
	Dokument	-	Linieskriver (printer)

6. LØSNINGSBESKRIVELSESMETODER.

Når man skal analysere sit problem, er det vigtigt, at problemet beskrives på en sådan måde, at man både har et overblik over den logiske sammenhæng mellem de enkelte delproblemer og har mulighed for at afprøve rigtigheden af de forskellige sammenhæng. De to mest anvendte metoder til løsningsbeskrivelse er blokdiagrammet og trinvis programforbedring, som bliver beskrevet i de følgende afsnit.

6.1. BLOKDIAGRAMMER.

Blokdiagrammet (=rutediagram=procesdiagram) kan udarbejdes i flere detaljeringsgrader. Først kan man udarbejde et diagram, der gengiver problemet i grove træk og dernæst lave diagrammer, som beskriver de enkelte delproblemer i alle detaljer.

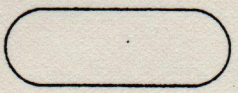
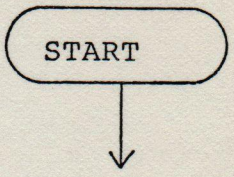
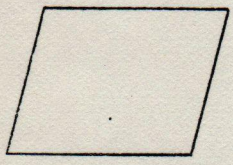
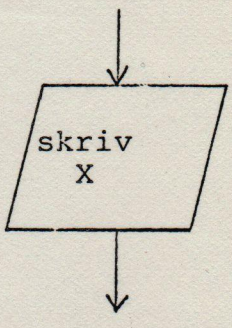
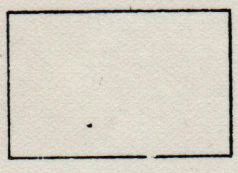
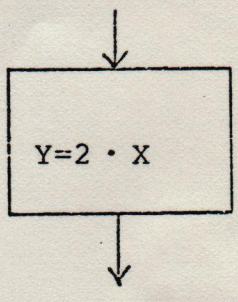
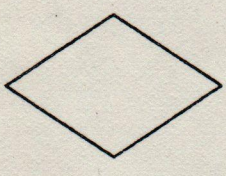
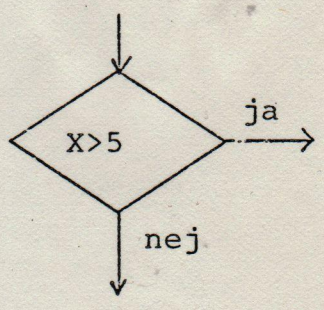
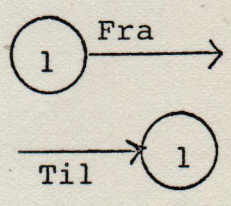
Ved udarbejdelsen af blokdiagrammer anvendes særlige symbolsæt, der er vist i figur 6.1.

Symbolerne forbindes med pile, som viser, hvordan diagrammet skal gennemløbes.

Ved konstruktion af et blokdiagram er der en del regler, som skal overholdes:

1: Et blokdiagram skal indledes med et startsymbol, og der må kun være en udgangspil fra startsymbolet.
2: Symbolerne for indlæsning/udlæsning og handling/beregning kan have en eller flere indgangspile, men kun én udgangspil.
3: Symbolet for logisk valg kan have en eller flere indgangspile, men har altid 2 udgangspile: en vandret ja-pil og en nedadrettet nej-pil.
4: Blokdiagrammet afsluttes med et stopsymbol, der kun må forekomme et sted. Til stopsymbolet kan der gå mange indgangspile, men ingen udgangspile.

Det kan ofte være svært at opbygge et blokdiagram, fordi man tvinges over i en logisk opbygning af problemløsningen i en detaljeringsgrad, man ikke rigtigt fornemmer på forhånd. Dette vil blive

SYMBOL	ANVENDELSE	EKSEMPEL
	Start eller slut	
	Indlæsning eller udlæsning	
	Beskrivelse af handling eller beregning	
	Logisk valg eller beslutning	
	Forbindelsesled på samme side	

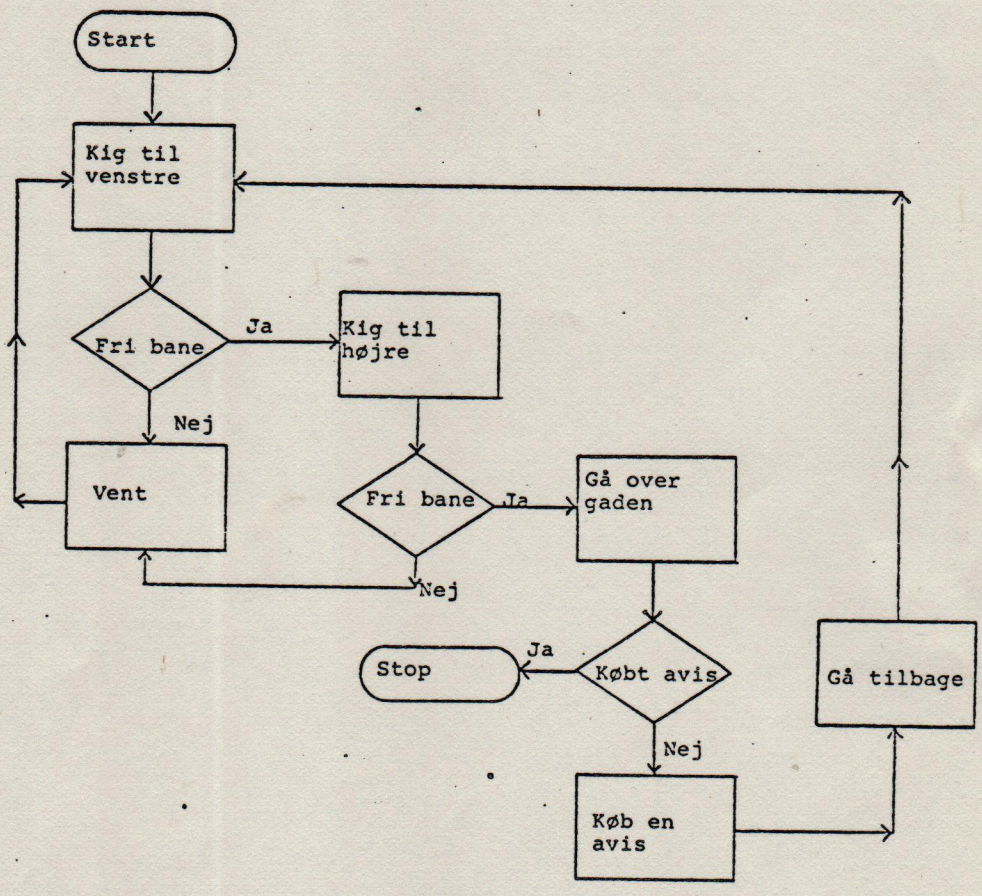
Figur 6.1. Symbolsæt for blokdiagrammer.

illustreret i det følgende ved hjælp af eksempler.

EKSEMPEL 6.1.

Vi ønsker at beskrive, hvad man skal foretage sig, når man skal over på den anden side af gaden for at købe en avis og herefter skal tilbage igen.

Problemet virker meget simpelt, fordi handlingen "at gå over gaden" udføres mere eller mindre ubevidst, men faktisk foretager man flere beslutninger undervejs, hvilket fremgår af nedenstående blokdiagram.



Figur 6.2. Blokdiagram til eksempel 6.1.

Det kan måske undre, at spørgsmålet "Fri bane?" stilles flere gange, men der må man huske på, at forudsætningerne for at stille de enkelte spørgsmål er forskellige.

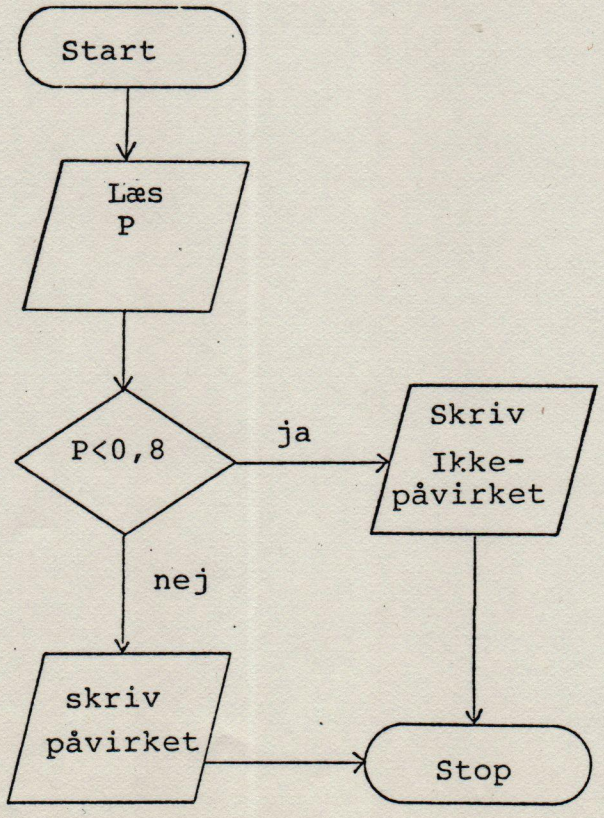
Når blokdiagrammet er udarbejdet, er det vigtigt, at det kontrolleres virkelig grundigt. Her kan det gøres ved at gennemløbe diagrammet med alle kombinationer om fri bane fra venstre og højre. Kontrollen af blokdiagrammet er så vigtig, fordi dette diagram danner det direkte grundlag for programmeringen, og det ved denne metode ofte er svært at foretage en rettelse i det senere program uden at ødelægge programmets samlede struktur.

EKSEMPEL 6.2.

Nu vil vi vise et diagram, der beskriver en udvælgelse af spritbilister i "påvirket" og "ikke-påvirket" tilstand på baggrund af promillen P. Hvis

P er mindre end 0,8 °/₀₀ er man "ikke-påvirket".
P er større end eller lig med 0,8°/₀₀ er man "påvirket."

Som resultat af udvælgelsen udskrives teksten "påvirket" eller teksten "ikke-påvirket".



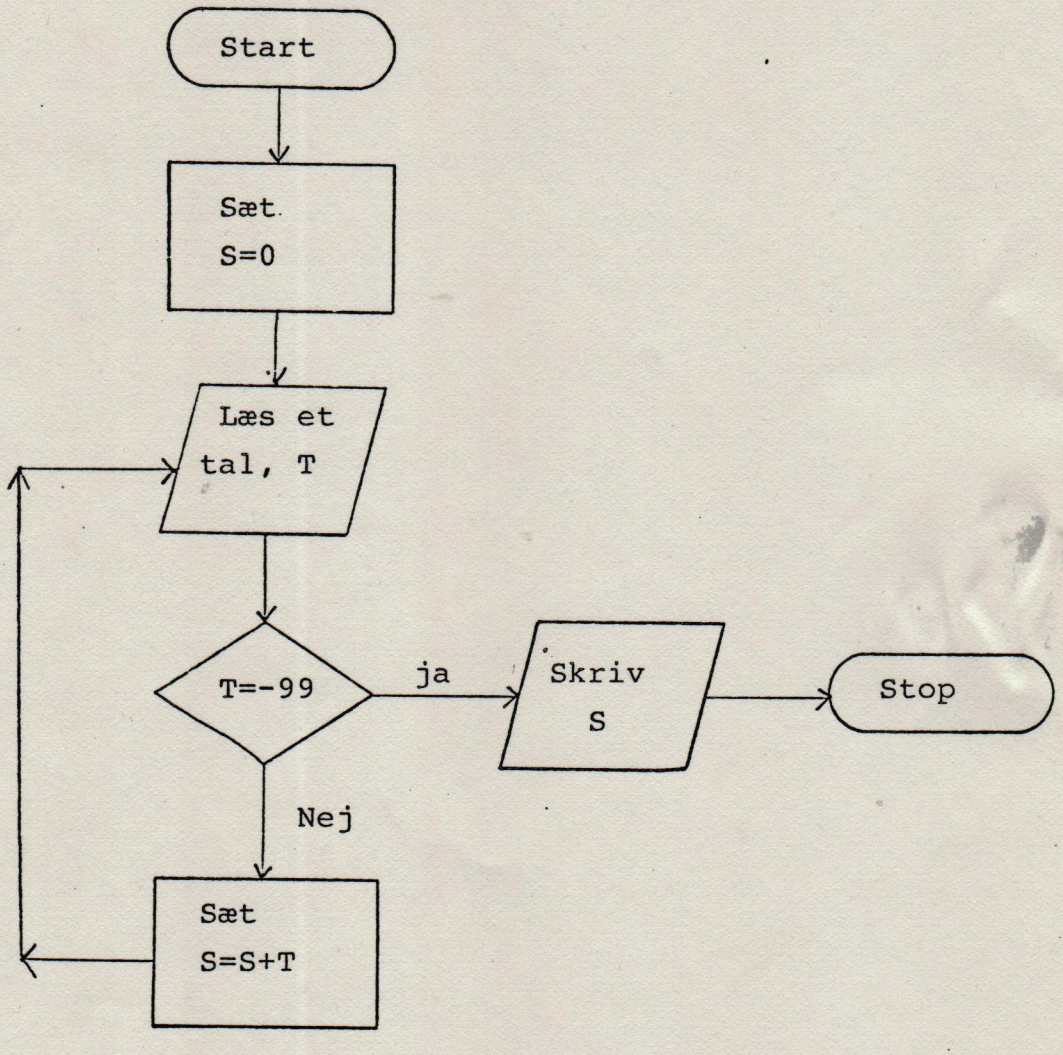
Figur 6.3. Blokdiagram for udvælgelse af spritbilister.

Til forskel fra eksempel 6.1. er det her nødvendigt først at foretage en indlæsning af den kendte promille, som gemmes i variablen P (P svarer til en af skufferne 0-7 i figur 4.2.a).

Desuden skal det bemærkes, at ved udvælgelsen "P<0,8" angiver NEJ-pilen tal, som er større end eller lig med 0,8.

EKSEMPEL 6.3.

Det diagram, der vises i dette eksempel, vil meget ofte indgå som en del af større diagrammer. Blokdiagrammet beskriver, hvorledes en række af tal læses, og efterhånden som de læses lægges sammen i en variabel S. Først når tallet -99 læses, stoppes optællingen, og summen af tallene udskrives.



Figur 6.4. Blokdiagram over summering.

Det første der sker i diagrammet er, at sumvariablen S nulstilles. Dette skal ses i sammenhæng med beregningen af summen: S=S+T, hvilket lettest forklares med et taleksempel.
Vi indlæser f.eks. tallene 7 og 5.

1. gennemløb: S=0, T=7, S=0+7=7
2. gennemløb: T=5, S=7+5=12

Hvis S ikke havde startværdien nul, ville slutresultatet blive forkert, da der så ville ligge et tilfældigt tal i S.

Ser vi på taleksemplet igen, sluttede vi med S=12 og skal derefter hoppe til indlæsningssymbolet igen, hvor vi har givet datamaten besked på at foretage en indlæsning. Men da der ikke er flere inddata, ender det med, at datamaten udskriver, at der er en fejl, og at programudførelsen stopper.

For at undgå en sådan fejlmeddelelse indlæser man tilsidst et tal, der aldrig vil forekomme i talrækken, her -99, hvorefter programudførelsen fortsætter, som angivet i blokdiagrammet. Tallet -99 er et eksempel på det man kalder stopdata, altså data der bevirker at et program stopper.

De tre viste eksempler var lette at overskue, især i forhold til opgaver fra det virkelige liv. Det næste eksempel er mere realistisk og måske derfor mere uoverskueligt. Som en hjælp til løsning vil vi først opstille en række punkter, som er nyttige ved en forhåndsanalyse af en opgave, før blokdiagrammet tegnes.

ANALYSE.

1) Læs opgaven grundigt (selv om den skulle fylde 1-1,5 A4 side), og noter evt. spørgsmål ned.
2) Sorter uvæsentlige oplysninger fra. Eks.: - Ifølge færdselslovens paragraf 79 stk. 8 punkt 15a 4.afsnit fra oven kan den idømmes bøde, hæfte eller.....hvis promillen er 0,8. Lav et blokdiagram der ud fra promillen..... o.s.v.-. I dette eksempel er alle de indledende oplysninger - paragraf 79...- overflødige!!
3) Hvad skal beregnes?

4) Hvad er inddata (hvad skal indlæses?)
5) Hvad skal udlæses (forlanges udskrifter?)
6) Har de angivne størrelser bogstavforkortelser? Hvis nej, lav forkortelser (f.eks. skrives: P=promillen, Sl=skat, A=antal, o.s.v.).
7) Skal blokdiagrammet gennemløbes flere gange? (f.eks. 100 piller indlæses, stopdata er 0,0,0, o.s.v.). Hvis ja, skal der sandsynligvis anvendes en eller flere tællevariable (T=T+1, A=A+1, o.s.v.).
8) Er der grænsetilfælde, der ikke er taget stilling til? Hvis ja tag stilling! og nedskriv beslutning.
9) Er der beregninger efter angiven formel? Hvis ja, indsæt de aktuelle forkortelser i formlen.
10) Kan der opstilles formler for nogle beregninger? Hvis ja, så anvend formlerne med indsættelse af forkortelser.
11) Er du i tvivl om, hvordan en tekst skal opfattes, så skriv ganske kort hvordan du vælger at opfatte den pågældende tekst, dog uden at den oprindelige opgavetekst derved bliver uigenkendelig.
12) Er det muligt på en simpel måde at begrænse opgavens omfang, så gør det (f.eks. ved at antage at en families medlemmer ved en rejse skal have ens billetter (retur/enkel). Dog uden at den oprindelige opgavetekst derved bliver uigenkendelig.
13) Ved udformningen af blokdiagrammet må du forestille dig, at du står over for en maskine, som kun fungerer ved, at du giver instruktioner om selv den mindste bevægelse.

I det følgende eksempel vil vi anvende de opstillede punkter, inden vi tegner blokdiagrammet.

EKSEMPEL 6.4.

Opgaven her går i al sin enkelthed ud på følgende:

100 ascorbinsyretabletter vejes enkeltvis på analysevægt. Der ønskes foretaget beregning og udskrift af middelvægt., samt en optælling og udskrift over fordelingen af tabletterne i forskellige vægtklasser. Desuden ønskes udskrevet, hvor mange tabletter, der er for store, og hvor mange, der er for små.

En ascorbinsyretablet skal veje mellem 0,1600 g og 0,2000 g.

Vægtklasserne er følgende: Klasse I vægt 0,1600 g - 0,1800 g.
Klasse II vægt 0,1801g - 0,2000 g.

Udarbejd et blokdiagram, der løser opgaven.

Benyttes der i blokdiagrammet forkortelser, bedes disse forklaret.

ANALYSE.

a) Hvad ønskes beregnet?

Svar:

1: Middelvægt af de 100 piller hvor middelvægten=

$$\frac{\text{pillernes samlede vægt}}{100}$$

2: Antal piller i vægtklasse I
3: Antal piller i vægtklasse II
4: Antal piller, der er for små
5: Antal piller, der er for store

b) Hvad ønskes indlæst?

Svar:

Pillens vægt (100 ialt)

c) Hvad ønskes udskrevet?

Svar:

1: Middelvægt
2: Antal piller i vægtklasse I
3: Antal piller i vægtklasse II
4: Antal piller der er for små
5: Antal piller der er for store

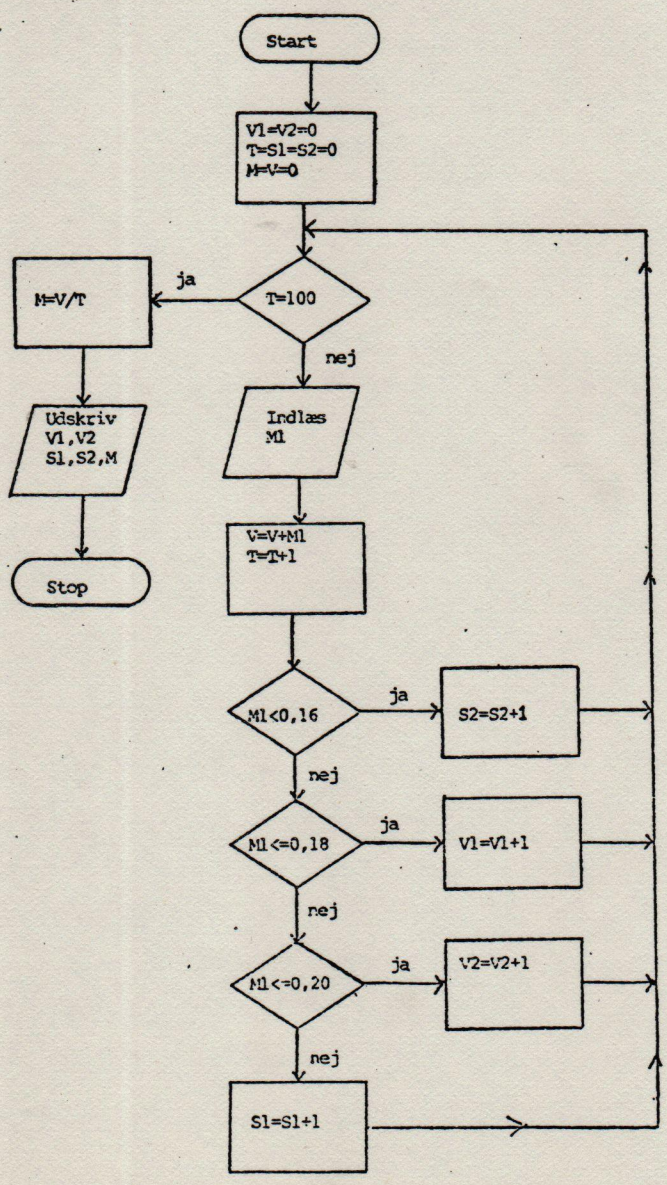
d) Hvilke variable er der brug for?

Svar:

T = pilletæller, startværdi: T=0
V1= pilletæller for vægtklasse I, startværdi: V1=0
V2= pilletæller for vægtklasse II
startværdi: V2=0
S1= pilletæller for for store piller,
startværdi: S1=0
S2= pilletæller for for små piller,
startværdi: S2=0
M = middelvægt
V = indlæste pillers samlede vægt,
startværdi: V=0
M1= den enkelte pilles vægt.

Med de indførte betegnelser bliver middelvægten nu: $M=\frac{V}{T}$

Blokdiagram.



6.2. TRINVIS PROGRAMFORBEDRING.

Den anden metode til løsningsbeskrivelse er trinvis programforbedring, der er en metode til struktureret programmering, hvor det strukturerede går på at programmet skal ligne problemets struktur, hvor løsningen med det samme beskrives i et pseudo-programmeringssprog. Ved denne metode benytter man altid følgende fremgangsmåde.

1: Grov skitse
2: Mindre grov skitse
3: Fin skitse
4: Helt fin skitse

Metoden bliver illustreret i det følgende eksempel.

Eksempel 6.5

Vi ønsker at fremstille et program, der kan sammenlægge en række tal (f.eks. periodens lønudbetalinger) og udskrive resultatet. Rækken af tal der skal lægges sammen (f.eks. 11.800, 12.850, 10.956, 13.417, - 99) vil være afsluttet af tallet -99, der indikerer at talrækken slutter her.

Opgaven er jo simpel nok, og kan løses med følgende program:

Trin 1:

1. Læs en række tal og læg dem sammen i S.
2. Udskriv resultatet S.
3. Stop

Nu kunne man jo læse alle tallene og lægge dem i hver sin skuffe i det indre lager, og herefter hente indholdet frem til sammenlægning. Hvis der er få tal fylder det jo ikke så meget i lageret, men kommer vi op på 10.000-vis af beløb der skal sammenlægges, er det jo ikke sikkert at de alle kan være i datamatens indre lager på én gang. Lad os prøve at lave programmet så der kun er brug for to skuffer (variable), én til det tal T vi lige har læst, og én til summen S af alle de tal vi har set tidligere. Med dette som udgangspunkt kan ovenstående program udformes som følger:

Trin 2

```
1. Gentag følgende:
2.    Læst et tal T
3.    sæt S = S + T
4. Indtil T = -99
6. Stop
```

Programmet i 2. trin ser jo ud til at fungere ved første øjekast. Det lider desværre bare af de mangler, at tallet -99 bliver talt med i summen af tallene (hvilket jo ikke var meningen), og variablen S har jo ikke nogen veldefineret værdi første gang udtrykket S = S + T mødes.

Dette kan vi råde bod på ved at trække den sidst indlæste værdi (nemlig -99) fra den endelige sum S, og ved at starte programmet med at nulstille S (S = 0), som vist i fase 3.

Fase 3

```
1. Sæt S = 0
2. Gentag følgende:
3.    Læs et tal T
4.    Sæt S = S + T
5. Indtil T = -99
6. Sæt S = S - T
7. Udskriv S
8. Stop
```

I det viste eksempel kunne man måske nok være nået frem til løsningen uden at opdele programmeringen i de 3 viste faser. Eksemplet skal jo også blot demonstrere fremgangsmåden ved trinvis programforbedring.

Havde vi f.eks. adgang til en banks lønkonti, og fik til opgave at lave et program der kunne kontrollere om en konto var spærret, om kontoen blev overtrukket, og om et af kunden opgivet kodeord var korrekt, måtte vi f.eks. starte med en råskitse som følger:

```
1. Indlæs kontonummer KTO
2. Indlæs beløb til hævning K
3. Indlæs sikkerhedskode SK
4. Åben kartoteket KTONR over kontonumre

5. Find og læs kontokort nr. KTO
6. Hvis konto KTO er spærret så
7.       udskriv "Alarm! kontoen er spærret"
8. Ellers hvis SK ≠ rigtig sikkerhedskode så
9.       udskriv "Forkert sikkerhedskode. Prøv igen"
10. ellers hvis Saldo - K < 0 så
11.      udskriv "Ingen dækning på kontoen"
12. ellers
13.      udbetal K kroner
14. End IF
15. Luk kartoteket KTONR
16. Stop
```

og bygge videre ud fra denne skitse. Prøv at gøre programmet færdigt, idet vi antager at et kartotekskort har følgende udseende:



```
Kontonummer    : KTO
Sikkerhedskode : SK
Navn           :
Adresse        :
Kundekode      :
Saldo          :
Renter         :
Spærrekode     :
```

7. PROGRAMMERING OG AFPRØVNING.

Det at fremstille et program for en maskine, betyder at udarbejde en liste af primitive operationer, der kan udføres af maskinen. Hvis der om programmet samtidig gælder at det på en utvetydig måde fører til løsning af et problem i et endeligt antal skridt, kaldes det for en algoritme.

Selve skabelsesprocessen, altså det at finde på en måde at løse et problem på, kan der ikke opstilles nogen model for. Man er her overladt til sin kreative sans, med hjælp i de afsnit 6 angivne metoder til løsningsbeskrivelse. Men er man nået så vidt, at et givet problem er blevet formaliseret på den ene eller anden måde, kan man starte på den egentlige programmering. Hvis man om et program skal kunne sige, at det udfører hvad man forventer det skal udføre, kræver det en eller anden form for dokumentation. Hertil benyttes i vid udstrækning test af programmer med på forhånd fastlagte værdier af inddatavariable, der så skal give anledning til bestemte værdier for uddata-variable, for at programmet kan siges at fungere. Inden for datalogien arbejder man også med at bevise at et program virker. Det største problem her er at få formuleret en præcis beskrivelse af forventningerne til et program.

Det vi skal beskæftige os med i dette afsnit er programmering i COMAL-80 samt metoder til afprøvning af et program.

7.1. PROGRAMMERING I COMAL-80.

COMAL-80 er et ud af mange programmeringssprog, men har den fordel at det er enkelt i sin opbygning, og kan danne en let overgang til f.eks. PASCAL, der er udråbt til at være et af fremtidens sprog. COMAL står for COMmen Algorithmic Language og er en kraftig udvidelse af programmeringssproget BASIC (Beginners All-purpose Symbolic Instruction Code).

Som ethvert andet sprog, har et programmeringssprog også en grammatik der skal følges og en betydning. Hvis vi på dansk siger: "Ræk mig et håndklæde", er meningen jo helt klart den, at man efter at have udsagt ordene, forventer at få et håndklæde. Grammatisk set er sætningen korrekt efter dansk retskrivnings-ordbog. Havde sæt-

set således ud: "Ræk mig en håndkledde", ville vi stadig fatte meningen, vel vidende at grammatikken er forkert. Dette skyldes, at man er i besiddelse af en portion fantasi, der hjælper en til at tænke, at meningen jo nok er den og den. Det er på dette punkt at vejene skilles mellem det menneskelige sprog og programmeringssprog, idet et programmeringssprog dels ikke indeholder nogen form for tvetydighed og dels grammatisk set, skal skrives 100% korrekt før en sætning accepteres og forstås. Dette faktum bør man bide sig fast i, da blot angivelsen af f.eks. et punktum i stedet for komma eller omvendt, kan gøre en sætning total uforståelig for datamaten. Nu er vi mennesker jo nogle besynderlige nogen, for hvis man på nogen måde kan kalde en ting for noget andet end man sædvanligvis benævner den pågældende ting, så gør man det. Således forholder det sig med ordene "grammatik" og "betydning", der inden for programmeringssprog kaldes for "syntaks" og "semantik". Man snakker altså om et sprogs syntaks (dets opbygning) og dets semantik (hvordan det skal opfattes).Nu kunne man jo konstruere et sprog hvor f.eks. "F3Q81" betød "find kvadratroden af 81 og læg 3 til". Det er jo nemt at indse, at dette ikke er nogen hensigtsmæssig notationsform for den angivne handling, at finde kvadratroden af 81 og lægge 3 til, idet "F3Q81" er svært at huske blandt måske 100 andre lignende former. En bedre notationsform vil f.eks. være "KVR(81) + 3" hvor KVR mindede en om ordet KVadrat Rod og + er man fortrolig med i forvejen. Det er da også således, at man for at lette brugen af programmeringssprog, i langt de fleste tilfælde bevidst har tilstræbt at gøre sproget let at huske, ved netop at bruge ord, der minder en om et eller andet. Desværre er det således fat, at disse mnemotekniske ord som de kaldes, næsten altid er udfærdiget til engelsk. I COMAL-80 vil ovenstående eksempel ikke hedde "KVR(81)+3". men derimod "SQR(81)+3", hvor SQR så skulle minde om det engelske SQuare Root. Det kan med andre ord betale sig at tænke på engelske ord, når man forsøger at forstå et program. Prøv f.eks. at betragte følgende program der er skrevet i COMAL-80.

```
0001 PRINT "Dette er 2-tabellen."
0002 FOR TÆLLER:=1 TO 10 STEP 1 DO
0003   TABEL:=TÆLLER*2 ↑Stopværdi efter TO
0004   PRINT TABEL
0005 NEXT TÆLLER
0006 PRINT "Slut for denne gang."
0007 END
```

Tallene til venstre er linienumre, og hver linie er en sætning. PRINT på engelsk betyder skriv, så betydningen af linierne 1,4 og 6 er såmænd nok at der skal skrives på terminalens skærm. I linie 1 udskrives teksten- Dette er 2-tabellen-. I Linie 4 udskrives indholdet af en variabel (=skuffe) TABEL, og i linie 6 udskrives teksten- Slut for denne gang.-

Betydningen af linie 3 kan man fornemme til at være: gang variablen TÆLLER med 2 og læg resultatet i variablen TABEL. Linie 2 og 5 giver umiddelbart den mening at man skal lægge 1 i variablen TÆLLER og udføre linie 3 og 4 for dernæst at lægge 1 (STEP 1) til variablen TÆLLER (NEXT TÆLLER) for derpå at udføre linie 3 og 4 igen o.s.v. indtil TÆLLER er lig med 10 (TO 10). Dette var blot et indledende eksempel på et lille COMAL-80 program, der udregner og udskriver totabellen på skærmen på følgende måde:

```
Dette er 2-tabellen.
2
4
6
8
10
12
14
16
18
20
Slut for denne gang.
```

Skærmbillede for ovenstående program.

For at få hold på de forskellige variable og sætningstyper, vil vi gennemgå COMAL-80 systematisk og starte med byggeklodserne, i dette sprog, nemlig sætningerne.

7.1.1.SÆTNINGER.

Et COMAL-80 program er opbygget af en række sætninger, hvor hver sætning består af et linienummer og en ordre. Linienumrene skal være heltallige værdier mellem 1 og 9999 begge inklusive. En sætnings linienummer bestemmer dens placering i programmet, idet programsætningerne automatisk ordnes i voksende linienummerorden uanset indtastningsrækkefølgen.

Umiddelbart efter indtastningen af en sætning foretages en syntaksanalyse, altså en kontrol af sætningens opbygning. Hvis sætningen er fejlbehæftet, udskrives en fejlmeddelelse, og markøren (den lysende plet på skærmen = cursor), placeres ud for det tegn, hvor fejlen blev opdaget. Herefter kan sætningen rettes ved brug af tastaturets rettefaciliteter, og en ny syntaksanalyse kan foretages. Ved godkendelse af en sætning udskrives en stjerne "*" på skærmen.

Det skal for god ordens skyld lige bemærkes, at indtastning af en sætning med et linienummer der eksisterer i forvejen, bevirker at den nye sætning vil erstatte den allerede eksisterende.

Når man skriver et program, er det en god ide at give sætningerne linienumre i spring på f.eks. 10, da det så på en nem måde er muligt at indføje evt. glemte sætninger. Hvis vi har skrevet et program

```
1   PRINT  " at være,"
2   PRINT  "eller være endnu værre,"
3   END
```

og her gerne vil indføre en manglende linie mellem linie 2 og 3, kan det jo ikke lade sig gøre med den anvendte linienummerering, da der ikke findes et heltal mellem 2 og 3. Udfører vi derimod nummeringen som

```
10  PRINT  "at være"
20  PRINT  " eller at være endnu værre,"
30  END
```

er der numrene fra 21 til 29 til at indføre nye sætninger. I dette tilfælde vælger vi nr. 25 til den manglende linie, og programmet kan nu fuldendes som

```
10  PRINT  "at være"
20  PRINT  "eller være endnu værre,"
25  PRINT  "det er spørgsmålet."
30  END
```

7.1.2. VARIABLE. ~ "skuffer"

Nu har vi snakket om sætninger og angivet eksempler herpå. Lad os nu gå videre med et eksempel på en sætning, som vi anførte i eksemplet med 2-tabellen, nemlig sætningen

```
4 TABEL:=TÆLLER*2
```

:= sættes lig med

Her er anvendt to navne TABEL og TÆLLER, der betegner de to variable. Når man snakker om "variable", kan man tænke på skufferne i datamatmodellen i fig. 4.2.a hvor en skuffe jo netop kunne have et varierende indhold i programforløbet. De to navne vil så svare til skuffernes navne. Lad os se på et lille eksempel, med disse to variable:

TABEL	4
TÆLLER	9

TABEL indeholder tallet 4 og TÆLLER tallet 9. Udfører vi nu ordren

```
TABEL:=TÆLLER*2
```

vil indholdet i skufferne ændres til

TABEL	18
TÆLLER	9

idet de to tegn ":=" skal udtales "sættes lig med" og TÆLLER jo indeholder 9 der ganget med 2 giver 18, der så lægges over i TABEL. Bemærk at "*" betyder multiplikation.

Der eksisterer nogle regler for, hvilke navne man kan give variable (skuffer). Et navn (også kaldet en identifikator) må maximalt indeholde 16 tegn, hvoraf det første skal være et bogstav. De tilladte tegn i et navn er følgende:

- Store og små bogstaver
- Cifre
- Understregning

Samtidig skal man være opmærksom på at navne der starter med bogstaverne "FN" er reserveret til specialbrug, og at små bogstaver i variabelnavne automatisk omformes til store bogstaver af COMAL-80 oversætteren. Hvis vi f.eks. anvender de to navne

TÆLLER og
tæller

vil de af COMAL-80 opfattes som værende identiske. Følger vi ovenstående regler, vil følgende navne bl.a. være lovlige:

C17H35COOH
Delta_X
m_1
BANAN

og følgende navne ulovlige:

Kaliumpermanganat
34K14
_MONEY
§48_stk_4

Prøv at finde ud af hvorfor de er ulovlige.

Som et eksempel på anvendelse af variable, kan vi her angive et program der beregner og udskriver kvadratroden af tallene fra 1 til 10.

```
0010 // REM PROGRAM TIL BEREGNING AF KVADRATRODEN.
0020 // REM
0030 TAL:=0
0040 TAL:=TAL+1
0050 RODEN:=SQR(TAL)
0060 PRINT "Kvadratroden af ";TAL;" = ";RODEN
0070 IF TAL<10 THEN GOTO 0040
0080 PRINT "Sådan gøres det."
0090 END
```

Navnene "TAL" og "RODEN" er variable, og de understregede ord er sprogets faste elementer (reserverede ord), der ikke må benyttes som variabelnavne. Det er f.eks. forbudt at bruge en variabel med navnet PRINT. Prøv at finde ud af hvordan ovenstående program virker.

For at gøre livet besværligt, har man valgt at arbejde med forskellige typer af variable. En variabel kan således afhængig af typen, tildeles et indhold der kan bestå af

- et tal (som ovenstående).
- et talsæt.
- en følge af tegn (en tekst).
- et sæt af tegnfølger.

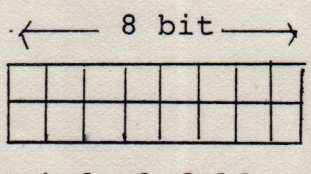
Den første variabeltype der kan indeholde et tal, kaldes for en numerisk variabel. En numerisk variabel kan igen underdeles i typerne

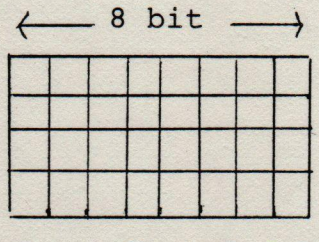
-reelle variable
-heltals variable

hvor den reelle variabel kan tildeles reelle talværdier der ligger i intervallet fra $\pm 2.9387359*10^{-39}$ til $\pm 1.7014117*10^{38}$ med 7 cifres præcision, eller fra $\pm 1.000000000000*10^{-128}$ til $\pm 9.999999999999*10^{126}$ med 13 cifres præcision, og heltalsvariablen kan tildeles heltallige værdier der ligger i intervallet fra

-32767 til 32767

Om der regnes med 7 eller 13 cifres præcision afhænger af, hvad det er for en oversætter man benytter. I COMAL-80 regnes der med 7 cifres præcision. Skal man arbejde med heltal>32767 kan man lagre dem som reelle tal, men skal så være opmærksom på, at det tager mindst dobbelt så lang tid at arbejde med reelle tal som med heltal. Dette skyldes at reelle tal fylder mere i lagret end heltal (se. fig. 7.1.).





Et heltal fylder 2 byte d.v.s. 16 bit.

Et reelt tal fylder f.eks. 4 byte d.v.s. 32 bit.

Fig. 7.1. Lagring af tal.

Skal man altså spare tid ved f.eks. mange ensartede beregninger og kan "nøjes" med heltal, kan man "forklare" datamaten at man ønsker at arbejde med heltalsvariable, ved efter variabelnavnet at tilføje et specielt tegn kaldet et nummertegn "#". Som eksempel er herunder angivet et program, der tæller fra 1000 og ned til 900 i spring på 9, samtidig med at summen af tællerværdierne udskrives på skærmen:

```
0010 // PROGRAM TÆLLER.
0020 SUM#:=0
0030 FOR TAL#:=1000 DOWNTO 900 STEP 9 DO
0040   SUM#:=SUM#+TAL#
0050 NEXT TAL#
0060 // Udskrivning
0070 PRINT "Summen = ";SUM#
0080 END
```

Som det ses er der efter de to variable "SUM" og "TAL" angivet et #, der således skal opfattes som heltalsvariable.

Prøv at finde ud af hvordan programmet virker, og forsøg herefter at finde ud af hvordan man kunne få skrevet alle mellemresultaterne af SUM# ud på skærmen.

Så vidt den numeriske variabel. Den anden variabeltype - et talsæt -, kan blot opfattes som en numerisk variabel med flere skuffer end én, altså med plads til flere tal. Hvis f.eks. en geolog har brug for på én gang at lagre information om forskellige boreprøvers indhold af jernmalm, og boreprøverne er identificeret ved deres beliggenhed på jordens overflade (altså ved en længdegrad og en breddegrad), kan han lagre oplysningerne om indhold af jernmalm i et talsæt, hvor de enkelte elementer i talsættet så udpeges af koordinaterne (længde, bredde). For nemheds skyld antager vi, at længde og bredde begge kun kan antage heltallige værdier mellem 1 og 10. Det talsæt vi således skal arbejde med, ser ud som vist i figur 7.2 , idet vi kalder talsættet for JERN.

	1	2	3	4	5	6	7	8	9	10
1										
2										
3		34.1							14.1	
4					45.0					
5					139.1					
6								9.3		
7			11.4							
8										
9										
10									6.5	

Fig. 7.2. viser talsættet JERN organiseret som et koordinatsystem, med værdier angivet i nogle af "skufferne".

Talsættet JERN, altså variablen JERN, har ialt 10*10 skuffer hvori der kan lagres informationer. Hvis vi gerne vil lægge tallet 117.4 ned i den skuffe der udpeges af længde = 6 og bredde = 9, kan det simpelthen gøres ved i COMAL-80 at skrive

10 JERN (6, 9):=117.4

der læses som: tallet 117.4 lægges i skuffen i søjle 6, række 9 i skuffe-arrangementet JERN. Dette kunne på et bibliotek svare til at man satte en bog ind på plads nr. 6 på 9. hylde i f.eks. reolen REJSE-LITTERATUR.

Hvis vi prøver at betragte et talsæt VIN med 3*4 skuffer indeholdende talværdier som vist i fig. 7.3 ,

	1	2	3 ⟶ Kvalitet
1	137.10	119.70	105.90
2	261.00	227.40	203.40
3	378.90	326.70	295.30
4	493.20	430.80	387.60

↓ Kvantum

Fig. 7.3. viser variablen VIN der indeholder priser for 3 kvaliteter af vin, i 4 forskellige kvanti.

hvor kvaliteten af vinen er betegnet med et tal fra 1 til 3 og antal kasser (kvantum) med et tal fra 1 til 4, kan vi få udskrevet samtlige priser fra variablen VIN v.h.a. følgende program:

Dimension - Reserverplads til variablen VIN(x,y) der har 3x4 skuffer

```
0010 DIM VIN(3,4)
0020 PRINT "Kvalitet  Kvantum  Pris"
0030 FOR KVALITET#:=1 TO 3 STEP 1 DO
0040  FOR KVANTUM#:=1 TO 4 STEP 1 DO
0050   PRINT KVALITET#;KVANTUM#;VIN(KVALITET#,KVANTUM#)
0060  NEXT KVANTUM#
0070 NEXT KVALITET#
0080 END
```

Indirect variabel - talvar

Udskriften fra kørsel af programmet kommer til at se ud som vist i tabel 7.1. Prøv at finde ud af hvordan programmet virker (ordet DIM er forklaret efter tabellen).

Kvalitet	Kvantum	Pris
1	1	137.1
1	2	261
1	3	378.9
1	4	493.2
2	1	119.7
2	2	227.4
2	3	326.7
2	4	430.8
3	1	105.9
3	2	203.4
3	3	295.3
3	4	387.6

Tabel 7.1. udskrift fra program vinpriser.

Det i programlinie 10 angivne ord DIM er en forkortelse for det engelske ord Dimension, hvormed man gør datamaten opmærksom på at en variabel har en udstrækning, en dimension. I det angivne tilfælde fortæller vi datamaten at variablen VIN har 3 søjler og 4 rækker. Arbejder vi således med talsæt, er vi nødt til inden vi bruger talsættet at beskrive størrelsen over for datamaten, da der i datamaten skal afsættes plads (skuffer) til det pågældende talsæt. I andre programmeringssprog kan man støde på betegnelsen "Array" for et sæt.

I de angivne eksempler har vi kun anvendt talsæt med to udstrækninger (søjlenummer, rækkenummer), men der er intet i vejen for, at man kan arbejde med 1 eller flere udstrækninger (f.eks. 4). Hvis man f.eks. havde sendt en raket til vejrs og man herfra målte strålingsintensiteten i rummet, kunne man registrere denne intensitet afhængig af rakettens position (længde, bredde, højde) og hvor lang tid raketten havde fløjet. Dette ville kræve et talsæt med 4 udstrækninger nemlig: længde, bredde, højde og tid.

Kalder vi talsættet for STRÅLING og vil gemme en målt intensitet 811, når raketten befinder sig i punktet 17,22,10 efter 3 timers flyvning, vil det tage sig ud som følger:

```
10 STRÅLING (17,22,10,3):=811
```

De tal vi udpeger en skuffe med, kaldes for indices. I eksemplet med stråling er 17,22,10 og 3 altså indices der udpeger en bestemt skuffe. Det vil derfor være nærliggende at kalde en skuffe i et talsæt for en indiceret variabel, hvilket den i realiteten også kaldes. I eksemplet med strålingen betegnes skuffen

stråling (17,22,10,3) altså som en indiceret variabel.

Den tredie variabeltype - en følge af tegn- kan indeholde en tekst, altså en følge af tegn. Denne variabeltype (tekst-variabel) er opfundet, da det mange gange kan være praktisk at arbejde med en tekst, i stedet for et tal. Principielt er der ingen forskel på at arbejde med tal og med tekster, bortset fra, at det måske kan lyde som noget vås at lægge to tekster sammen. Det vil vi se nærmere på.

Hvis man vil arbejde med en tekstvariabel, må man informere datamaten om,at en bestemt variabel skal opfattes som sådan. Dette gøres ved efter variabelnavnet at angive et dollartegn "$". Vil man f.eks. lægge teksten "Laborantskolen" over i en variabel der hedder STED, skal det gøres således:

```
0005 DIM STED$ OF 14
0010 STED$:="Laborantskolen"
0020 END
```

Grunden til at vi i ovenstående eksempel må erklære variablen STED$ v.h.a. en DIM sætning er, at datamaten skal reservere plads til variablen STED$'s maximale størrelse. I ovenstående eksempel har vi fortalt datamaten at STED$ er en tekstvariabel, der maximalt kan indeholde 14 tegn (v.h.a. OF 14). D.v.s. at tekster med længder større end 14, ikke kan gemmes i variablen STED . Det er også muligt at lægge tekster sammen. Hvis vi f.eks. har en tekstvariabel NAVN$, kan den f.eks. benyttes som vist i nedenstående program:

```
0010 DIM NAVN$ OF 10
0020 NAVN$:="ANDERS"
0030 NAVN$:=NAVN$+" AND"
0040 PRINT NAVN$
0050 END
```

der vil give udskriften:

ANDERS AND

Hvis vi illustrerer NAVN$'s indhold i linie 10 og 20 vil det se ud som følger

	1	2	3	4	5	6	7	8	9	10	
NAVN $	A	N	D	E	R	S					i linie 10
NAVN $	A	N	D	E	R	S		A	N	D	i linie 20

Læg mærke til at den tekst man ønsker at gemme i en tekstvariabel, skal omgives af to anførselstegn (f.eks. "ANDERS"), og at disse to tegn ikke hverken gemmes eller udskrives.

Den fjerde og sidste variabeltype - et sæt af tegnfølger (tekster) - virker principielt på samme måde som et talsæt. Hvis f.eks. en vicevært for en opgang med 4 etager og 3 lejligheder pr. etage, ønsker at gemme navnene på lejlighedernes indehavere (én pr. lejlighed) kan han f.eks. gøre det i et sæt af tegnfølger kalder EJERE$. Lad os antage at EJERE$ ser således ud:

	1	2	3
1	Knudsen	Olsen	Jensen
2	Petersen	Holm	Hertel
3	-ubeboet-	Svendsen	Bennicke
4	Gawanetski	Hansen	Møller

↓ etage

lejlighed

Fig. 7.4. Indholdet af EJERE $.

Hvis vi gerne vil udskrive navnene på samtlige ejere i opgangen kan det gøres med følgende program:

```
0010 // PROGRAM EJER-udskrift.
0020 DIM EJERE$(3,4) OF 10
0030 PRINT "Lejlighed  Etage  Ejer"
0040 FOR LEJLIGHED#:=1 TO 3 STEP 1 DO
0050  FOR ETAGE#:=1 TO 4 STEP 1 DO
0060   PRINT LEJLIGHED#,ETAGE#,EJERE$(LEJLIGHED#,ETAGE#)
0070  NEXT ETAGE#
0080 NEXT LEJLIGHED#
0090 END
```

Bemærk ligheden med program VINPRISER, der er vist tidligere i afsnittet. Udskriften fra program EJER-udskrift er vist i tabel 7.2.

Lejlighed	Etage	Ejer
1	1	Knudsen
1	2	Petersen
1	3	-Ubeboet-
1	4	Gawanetski
2	1	Olsen
2	2	Holm
2	3	Svendsen
2	4	Hansen
3	1	Jensen
3	2	Hertel
3	3	Bennicke
3	4	Møller

Tabel 7.2. Udskrift fra program EJER-udskrift.

Udskriften ligner jo i betænkelig grad udskriften fra program VINPRISER, hvilket vel ikke kan overraske nogen. Selvom det er nyt og måske ikke så nemt at kapere i første omgang, skal man ikke være bange for at eksperimentere med tekstvariable. Der kan højest komme noget volapyk ud af det. Prøv at overveje hvordan man kunne lave et program der lavede "sjove" ord, med en længde på 8 (TIP:Indlæs nogle stavelser i et sæt af tekstvariable og bland dem på forskellig vis).

OPGAVE

Grunden til at vi har gjort så meget ud af variable er, at variable er forudsætningen for at man overhovedet kan lave et fornuftigt program. Hvis ikke der var noget der hed variable, kunne man jo ikke gemme beregningsresultater, inddata m.v..

7.1.3. KONSTANTER

Konstanter er størrelser, der som navnet lader antyde, ikke ændrer sig i et programforløb. Man kan således heller ikke gemme resultater i en konstant. F.eks. er tallet 0.0821 en konstant (gaskonstanten). Der findes talkonstanter, der benyttes i aritmetiske udtryk, i inddata, i CASE- og DATA-sætninger (se afsnit 7.1.4.5. og 7.1.4.3.), og tekstkonstanter der benyttes som konstanter i tekstudtryk, i inddata, i ledetekster ved brug af INPUT-sætningen (se afsnit 7.1.4.3.), samt i CASE- og DATA-sætninger.

7.1.3.1. TALKONSTANTER.

Talkonstanter kan være reelle tal (altså tal med decimaler) eller heltal (tal uden decimaler). Et reelt tal kan være opbygget på følgende måder:

1: Som en række cifre (1191).
2: Som en række cifre efterfulgt af et punktum (967.).
3: Som en række cifre efterfulgt af et punktum, igen efterfulgt af en række cifre (33.1878).
4: Som punkterne 1-3, efterfulgt af et E og en række cifre (1191E9,967.E3,33.1878E10).
5: Som punkterne 1-3 efterfulgt af et E og tegnet plus (+) eller minus (-) samt en række cifre (1191E+3,967.E-2, 33.1878E+10).

Et heltal opgives som en række af cifre (11936).

Eksempler på reelle tal:

44.1	-6.18E-3	0	11.5E7	

Eksempler på heltal:

999	6830	-4	0	14040

Der må ikke forekomme blanktegn (altså mellemrum) i talkonstanter. Bemærk endvidere at der bruges decimalpunktum i stedet for decimalkomma. Det i ovenstående eksempler anvendte E betyder Exponent, altså 10'er exponent. Det betyder at f.eks. tallet 11.5 E7 i virkeligheden skal opfattes som $11.5 * 10^7 = 115000000$. Hvis der efter E'et står et plus (+) er det at opfatte som et E uden plus. Står der derimod et minus (-) efter E'et, betyder det at dividere med 10'er exponenten uden minus. Har vi f.eks. tallet 2.7E-2 er dette lig med $2.7 * 10^{-2} = 2.7/10^2 = 0,027$. Som eksempel på anvendelse af talkonstanter i et beregningsudtryk, er herunder anført beregningen af en kugles rumfang A når radius i kuglen R kendes:

```
10 A:=4/3*3.141593*R↑3
```

Den angivne pil (↑) betyder potensopløftning, og stjernerne (*) gange.

7.1.3.2. TEKSTKONSTANTER.

En tekstkonstant består af en følge af tegn omgivet med anførselstegn, hvorfor anførselstegnet i sig selv jo ikke kan indgå i en tekstkonstant. Hvis man alligevel ønsker anførselstegn i sin tekst, kan dette gøres ved at skrive to på hinanden følgende anførselstegn (""). Ønsker man f.eks. at udskrive teksten - Gummi Tarzan - gøres det på flg. måde:

```
10 PRINT "Gummi Tarzan"
```

og har man ønske om at udskrive teksten - kan du sige: "øllebrød"?-, skal det gøres på følgende måde:

```
10 PRINT"Kan du sige:""øllebrød""?"
```

altså med 2 anførselstegn i hver ende af ordet - øllebrød -. En tekstkonstant kan også indeholde blanktegn (altså mellemrum). Endvidere er det muligt at lægge tekstkonstanter sammen på følgende måde, idet resultatet udskrives:

```
10 PRINT "FJERN" + "SYN"
```

der giver udskriften:

```
FJERNSYN
```

7.1.4. SÆTNINGSTYPER

I de foregående afsnit har vi set på variable og konstanter. Nu skal vi til at se på hvorledes vi kan arbejde med disse størrelser. Vi er allerede stødt på forskellige sætningstyper i diverse eksempler.

7.1.4.1. TILDELINGSSÆTNINGER OG ARITMETISKE UDTRYK.

I afsnit 7.1.3.1. angav vi et eksempel med udregning af en kugles rumfang efter formlen $A=4/3\pi R^3$, idet vi skrev

```
10 A:=4/3*3.141593*R↑3
```

Denne sætningstype kaldes for en tildelingssætning, da vi tildeler variablen A værdien af udtrykket, der står på den højre side af ":=", der udtales "sættes lig med". Hvis vi laver et lille program der udregner kuglens rumfang for R=4, vil det se ud som følger:

```
0010 // PROGRAM KUGLE
0020 PI:=3.141593
0030 R:=4
0040 A:=4/3*PI*R^3
0050 END
```

I dette program starter vi i linie 20 med at tildele variablen PI værdien 3.141593 (konstant), og i linie 30 variablen R værdien 4. Dernæst beregner vi i linie 40 værdien af udtrykket, for de værdier af PI og R som disse variable har på dette tidspunkt. Resultatet af beregningen lægges derpå over i variablen A. Dette kan illustreres i følgende 3 trin:

TRIN	Variabel		
	PI	R	A
1	3.141593	?	?
2		4	?
3			~~268.0826~~ 27.162

De i skemaet angivne ? betyder, at vi ikke kender indholdet af en variabel før vi har lagt noget i den. Resultatet af beregningen bliver altså 268.0826 der lægges over i A. Når man altså skal tildele en variabel en værdi gøres det efter følgende model:

	Linienr.	Variabel der skal lægges i	:=	udtryk
eks:	20	NUMMER#	:=	2↑3

Denne model gælder selvfølgelig også for talsat indicerede variable. Lad os antage at vi skal beregne rumfanget for 5 kugler med hver deres radius. Dette kan gøres på en smart måde, uden at vi behøver at angive formlen for rumfanget 5 gange (en pr. kugle):

```
0010 // PROGRAM KUGLE-II.
0020 DIM RADIER(5)
0030 RADIER(1):=2.5; RADIER(2):=8.26; RADIER(3):=1.6
0040 RADIER(4):=11.35; RADIER(5):=7.0
0050 PI:=3.141593
0060 KUGLENR:=1
0070 R:=RADIER(KUGLENR)
0080 A:=4/3*PI*R^3
0090 PRINT "Rumfanget af kugle ";KUGLENR;" = ";A
0100 KUGLENR:=KUGLENR+1
0110 IF KUGLENR<6 THEN GOTO 0070
0120 END
```

Prøv at finde ud af hvordan programmet fungerer, evt. ved at tegne

et skema over de forskellige variable's indhold efterhånden som de forskellige programlinier aktiveres, og angiv udskriften fra programmet.

I forbindelse med beregning af udtryk, eksisterer der nogle ganske bestemte operatorer og nogle ganske bestemte regler for, hvordan et udtryk beregnes, idet det ikke er ligegyldigt i hvilken rækkefølge man anvender operatorerne. Af aritmetiske operatorer findes der følgende:

Operator	betydning	prioritet	eksempel
↑	Potensopløftning	1	X↑2
/	Division	2	SUM/TOTAL
*	Multiplikation	2	PRIS*ANTAL
DIV	Heltalsdivision	2	N#DIV K#
MOD	Modulus	2	N#MOD K#
+	Addition	3	SUM + BELØB
-	Subtraktion	3	BELØB-RABAT

Tabel 7.3. Operatorer.

Alle disse operatorer kalder man for Dyadiske, hvilket er betegnelsen for operatorer der kombinerer 2 operander til et resultat. Over for denne type operander står så de Monadiske operatorer, der kun har én operand og et resultat (f.eks. foranstillet minus).

Lad os kigge lidt på de forskellige operatorer. De fire operatorer /,*,+ og- er kendt af alle, og kræver som sådan ingen forklaring. Det skal dog lige bemærkes at tegnet for multiplikation er en stjerne (*), i stedet for punktum, kryds eller ingenting. Hvis vi f.eks. i en formel skal gange A med B vil det se ud som AB, hvil-

ket ved oversættelse til COMAL skal angives A*B. Operatoren DIV, altså heltalsdivision, betyder at man kan dividere to heltal med hinanden, og uanset om divisionen går op eller ej, får man et heltal som resultat. Dividerer vi f.eks. 4 med 3, der begge er heltal, får vi resultatet 1 (4 DIV 3=1). Har vi to heltal N og P, er resultatet af N DIV P defineret som den heltallige værdi af X, der får udtrykket (N-X*P) til at antage sin mindste ikke-negative værdi. Eksempler på heltalsdivision:

DIV arbejder på heltal

```
 17   DIV   4  =  4
 17   DIV (-4) = -4
-17   DIV   4  = -5
-17   DIV (-4) =  5
```

Prøv at beregne følgende udtryk:

```
 23   DIV (-8) = -2
-117  DIV  10  = -12
-19   DIV (-3) = +7
```

Operatoren MOD betyder den heltallige rest man får ved division af to heltal. Hvis vi f.eks. skal finde modulus af 11 DIV 3 bliver det 2, da 3 går op i 11 3 gange og 11-3*3=2. Sagt på en anden måde, er resultatet af N MOD P defineret som den mindste, ikke-negative værdi, som udtrykket (N-X*P) kan antage for heltallige X. Eksempler på modulus-beregning:

```
 17  MOD   4  =  1
 17  MOD (-4) =  1
-17  MOD   4  =  3
-17  MOD (-4) =  3
```

Prøv at beregne følgende udtryk:

```
 23  MOD  (-8)
-117 MOD   10
-19  MOD  (-3)
```

N MOD P kan også udtrykkes v.h.a. DIV således at N MOD P = N-P*(N DIV P).

Operatoren ↑ , altså potensopløftning, betyder at man skal gange tallet på venstre side af pilen med sig selv det antal gange der er angivet på højre side af pilen. F.eks. betyder 2 ↑ 3 2 ganget med sig selv 3 gange, altså 2*2*2 ($=2^3$).
Eksempler på potensopløftninger:

Matematisk	Comal	Resultat
10^3	10 ↑ 3	1000
2^5	2 ↑ 5	32
2.5^2	2.5 ↑ 2	6.25
-1.7^{-2}	(-1.7) ↑ (-2)	0.3460

Hvis potensen er negativ skal man stadig gange tallet med sig selv, men resultatet af dette skal man dividere op i 1. D.v.s. at f.eks. 3^{-2} i virkeligheden betyder $1/3^2$.

Nu har vi beskrevet operatorernes betydning. Når man sætter sådanne operatorer sammen i et beregningsudtryk, skal man ved udregning af udtrykkets værdi følge rækkefølgen som angivet i tabel 7.3 fra venstre mod højre. D.v.s. at man først (Prioritet=1) undersøger udtrykket for forekomster af potensopløftninger fra venstre mod højre, og udfører beregningerne. Dernæst undersøger man udtrykket for forekomster af *,/,DIV og MOD fra venstre mod højre og foretager beregningerne. Dernæst undersøger man så udtrykket for forekomster af + og - fra venstre mod højre og udfører beregningerne. Lad os prøve at beregne udtrykket

```
0:  3/5*7+13 MOD 3↑2 DIV 2
1:  3/5*7+13 MOD 9 DIV 2
2:  0.6*7+13 MOD 9 DIV 2
3:    4.2+13 MOD 9 DIV 2
4:        4.2+4  DIV 2
5:           4.2+2
6:             6.2
```

Man kan ændre ved udregningsrækkefølgen ved at indføje parenteser,

da parentesers indhold skal udregnes før noget andet. Hvis vi indføjer nogle parenteser i ovenstående udtryk, kan vi prøve at foretage beregningen på ny:

```
0:  3/(5*7) + (13 MOD 3) ↑ 2 DIV 2
1:    3/35 + (13 MOD 3) ↑ 2 div 2
2:          3/35+1 ↑ 2 DIV 2
3:          3/35+1 DIV 2
4:          0.086+1 DIV 2
5:            0.086+0
6:              0.086
                ======
```

Som det ses får vi to vidt forskellige resultater. Prøv at beregne værdien af udtrykkene:

```
1:  (3+4*6)/9 DIV 2*(4↑2+1)          =   17
2:  17 MOD (2↑3-19)*(16*(8/4-1))     =   96
3:  29 MOD 15 DIV 2 MOD 3            =   1
```

Nu kunne man jo spørge sig selv hvorfor man skal kunne beregne sådanne udtryk? Det er vel det vi har datamaten til? Det er da rigtigt, men vi må ikke glemme at det er os, der skal fodre datamaten med de rigtige formler, hvorfor vi er nødt til at vide hvordan datamaten opfatter et beregningsudtryk.

7.1.4.2. TILDELINGSSÆTNINGER MED TEKSTVARIABLE.

Principielt er der ingen forskel på tildeling af numeriske værdier og tildeling af tekst til en variabel. Den lille forskel der alligevel er, ligger i antallet og betydningen af de operatorer man har til rådighed.

Følgende tekstoperatorer findes:

Operator	betydning	eksempel
+	sammensætning	"KEMIK"+"ALIER"
&	sammensætning	"S"&"TOL"
IN	Forekomst af en tekst i en anden	"LE" IN "ÆBLER"

Alle operatorerne er dyadiske. De to første + og & er identiske, og bevirker sammensætning (konkatenering) af to tekster til én, idet resultatet bliver teksten på venstre side forlænget med teksten på højre side af operatoren. Hvis vi f.eks. sammensætter teksten "KOM" og teksten "PENDIUM" og lægger resultatet over i variablen EKSEMPEL$ på flg. måde

```
10 EKSEMPEL$:="KOM"+"PENDIUM"
```

vil EKSEMPEL$ nu indeholde teksten "KOMPENDIUM".

Operatoren IN giver som resultat en logisk værdi - sand eller falsk -, afhængig af om teksten på venstre side af operatoren forekommer i teksten på højre side af operatoren eller ej. F.eks. vil

```
"LOM" IN "BLOMSTER" være sand
"LOM" IN "BLOK" være falsk.
```

og IN-operatoren kan f.eks. anvendes til fra en adresseliste at udvælge adresser med bestemte egenskaber (f.eks. alle adresser hvorpå der boede personer med efternavnet "PETERSEN"). Overvej hvorledes man i program EJER-udskrift sidst i afsnit 7.1.2., ved anvendelse af IN-operatoren kunne udvælge og udskrive alle navne der ender på "SEN".

OPGAVE

7.1.4.3. LÆSESÆTNINGER.

Når man fra et program snakker om at læse, betyder det at man læser data, at man tilfører variable nogle værdier. Nu er der så spørgsmålet hvor man læser fra, for data må jo stå et eller andet sted. Det vi skal beskæftige os med her, er læsning af data fra

1: Programmet (READ-DATA)
2: Terminalen (INPUT)
3: Baggrundslageret (INPUT FILE)

Lad os prøve at kigge på programmet KUGLE II i afsnit 7.1.4.1. I programlinierne 30 og 40 tildeler vi variablen RADIER en værdi til hver af skufferne RADIER(1)- RADIER (5). Disse værdier kan tilføres RADIER på en lidt smartere måde, idet vi kan gøre brug af en DATA-sætning, hvor vi så skriver de 5 værdier, der skal lægges over i RADIER. Overførslen af værdierne sker v.h.a. en læsesætning

READ. Lad os prøve at indføre READ- og DATA-sætningen i programmet KUGLE-II:

```
0010 // PROGRAM KUGLE-II.
0020 DIM RADIER(5)
0030 READ PI
0040 DATA 3.141593, 2.5, 8.26, 1.6, 11.35, 7.0
0050 FOR KUGLENR:=1 TO 5 DO
0060  READ RADIER(KUGLENR)
0070 NEXT KUGLENR
0080 FOR KUGLENR:=1 TO 5 DO
0090  R:=RADIER(KUGLENR)
0100  A:=4/3*PI*R^3
0110  PRINT "Rumfanget af kugle ";KUGLENR;" = ";A
0120 NEXT KUGLENR
0130 END
```

Vi starter i linie 30 med at læse en værdi over i variablen PI. Den værdi PI tilføres, vil være den første i DATA-sætningen i linie 40, nemlig værdien 3.141593. Næste gang der skal læses, vil det være den næste værdi i DATA-sætningen der læses. Man læser altså videre fremad, akkurat som man kunne forestille sig en rulle toiletpapir, hvor der på hvert blad stod en værdi; når man havde taget

3.141593	2.5	8.26	1.6	1.35	7.0	/

et blad kunne man jo ikke sætte det på igen, men måtte gå videre til det næste.

I linie 60 møder vi igen ordet READ, der står inde i en løkke hvor KUGLENR løber fra 1 til 5 samtidig med at værdier fra DATA-sætningen tilføres de indicerede variable RADIER(1), RADIER(2),... RADIER(5). Når man således benytter READ-sætningen, skal der også være en DATA-sætning. Denne form for læsning, er læsning af data fra selve programmet. Selv om READ-DATA konstruktionen er rimelig smart, er den dog temmelig klodset, hvis man f.eks. har lavet et program til beregning af trykket P ud fra formlen P V=n R T, hvor man altså skal kende rumfanget V, temperaturen T og antal grammolekyler og således i programmet skal have disse størrelser i en DATA-sætning. Det ville betyde at man hver gang der kom et nyt sæt måleværdier, skulle ind og ændre i data-sætningerne i programmet.

Skal man gøre det mere end nogle få gange, bliver man hurtig træt af den spøg.

Heldigvis er der opfundet en anden type læsesætning, hvor man mens programmet kører,kan tilføre det data, via terminalens tastatur. Denne sætningstype hedderINPUT (uden D i), og bevirker udskrift på terminalens skærm af et spørgsmålstegn til orientering for brugeren af programmet om, at vedkommende skal indtaste data. Syntaksen for en INPUT-sætning er følgende:

Linienr INPUT textkonstant: variabel, variabel,.....

idet det dog skal bemærkes at tekst konstanten og kolon kan undværes. Det skal endvidere bemærkes at det kun er i tilfælde af, at tekstkonstant og (:) udelades, at der udskrives et (?), ellers udskrives tekstkonstanten. Lad os se på et par eksempler, idet første eksempel vil være programmet kugle-II fra afsnit 7.1.4.1, hvori der indføjes en INPUT-sætning:

```
0010 // PROGRAM KUGLE-II.
0020 PI:=3.141593
0030 INPUT "Indtast kuglens radius (0 betyder stop) ": R
0040 IF R<=0 THEN GOTO 0080
0050 A:=4/3*PI*R^3
0060 PRINT "Rumfanget af kuglen = ";A
0070 GOTO 0030
0080 PRINT "Slut for denne gang."
0090 END
```

Programmet får nu tilført data i linie 30, idet det nu er brugeren af programmet der bestemmer hvilken radius der skal indtastes. Hvis vi forestiller os terminalens skærm, vil følgende dialog med programmet kunne udspille sig, idet understreget tekst udskrives af programmet, og ikke understreget tekst, er brugerens (dit) svar til programmet:

<u>Indtast kuglens radius (0 betyder stop)</u> 17.1
<u>Rumfanget af kuglen = 20944.84</u>
<u>Indtast kuglens radius (0 betyder stop)</u> 6.25
<u>Rumfanget af kuglen = 1022.654</u>
<u>Indtast kuglens radius (0 betyder stop)</u> 11.35
<u>Rumfanget af kuglen = 6124.579</u>
<u>Indtast kuglens radius (0 betyder stop)</u> 0
<u>Slut for denne gang.</u>

Selve understregningerne kommer selvfølgelig ikke med på skærmen, men er kun anvendt for at lette på forståelsen. De svar brugeren taster ind på tastaturet (her tallene 17.1 , 6.25 og 0) bliver lagt over i variablen R. Efter hver beregning og udskrivning af A, hoppes til linie 30, og læsning af radius for en ny kugle kan begynde, med en efterfølgende beregning og udskrivning af A o.s.v., indtil R får værdien 0, hvorefter programmet stopper.

Som andet eksempel kan vi tage det samme program som før, og her undlade at angive en tekstkonstant, således at linie 30 nu skal udformes som

```
30 INPUT R
```

udskriften på skærmen vil nu blive følgende:

```
?17.1
Rumfanget af kuglen = 20944.84
?6.25
Rumfanget af kuglen = 1022.654
?11.35
Rumfanget af kuglen = 6124.579
?0
Slut for denne gang.
```

Fig. 7.6. Samme skærmdialog som i fig. 7.5., dog uden textkonstant i INPUT-sætningen.

Understregningen i fig.7.6. har samme betydning som i fig. 7.5.

Nu har vi set på tilførsel af data til programmet fra selve programmet og fra terminalen. Hvis man over længere tid skal gemme og bruge de samme data til forskellige ting, gemmer man dem på et baggrundslager (floppy-disk, pladelager, m.v.). Det vi nu skal se på, er hvordan man læser data fra et sådant baggrundslager via et program (skrivning kommer senere). Principielt er der ingen forskel på at læse data fra terminalen og fra baggrundslageret.

Den eneste forskel der alligevel er, er at en mængde data på et baggrundslager har et navn, så man kan skelne de forskellige datamængder fra hinanden (der kan til forskel fra terminalen, være mere end én datamængde på baggrundslageret). D.v.s. at man, inden man fra

En file kan være et program eller en datamængde

programmet kan læse fra baggrundslageret, må fortælle datamaten, hvad det er for en datamængde (eng.: file) man vil læse fra, ved at angive navnet på den. Denne handling kaldes at åbne filen (eng.: open). Skal man således åbne en fil med navnet RADIER.DAT (angående filnavne se afsnit 9), der f.eks. indeholder mængden af radier for hvilke man vil beregne rumfanget af en kugle, gøres dette med sætningen

```
10 OPEN # 3,"RADIER .DAT", READ
```

åben fil 3 du hedder Radier.Dat og skriv det ved

hvor følgende ting oplyses:

1: at navnet er RADIER.DAT
2: at alle senere referencer til denne datamængde skal ske ved nummeret 3 (#3)
3: at datamængden skal læses (READ), altså at den eksisterer i forvejen.

nummertegn her # file

Når dette er gjort, kan man læse fra datamængden (filen), som man læser fra terminalen ved f.eks. at skrive

```
10 INPUT # 3 : R
```

hvorved R får tildelt en værdi fra filen.Lad os eksempelvis se på hvorledes programmet KUGLE-II vil se ud ved læsning af data fra baggrundslageret:

3 er tilfældig valgt - skal blot være det samme

En

```
0010 // PROGRAM KUGLE-II.
0020 PI:=3.141593
0030 OPEN FILE 3, "RADIER.DAT", READ
0040 INPUT FILE 3: R
0050 IF R<=0 THEN GOTO 0090
0060 A:=4/3*PI*R^3
0070 PRINT "Rumfanget af kuglen = ";A
0080 GOTO 0040
0090 PRINT "Slut for denne gang."
0100 CLOSE FILE 3
0110 END
```

læs det første tal i filen og sætte ind for R

Bemærk ligheden med forrige version af programmet. Bemærk endvidere sætningen CLOSE # 3 , der frigiver nummeret 3 til evt.

andre filer der skal læses/skrives. CLOSE er engelsk og betyder "lukke". Man skal altså når man arbejder med filer, både åbne og lukke dem på den anviste måde. En fil kaldes også for et kartotek.

Kartoteker.

Når man skal arbejde med kartoteker, kan man vælge forskellige typer af kartoteker. Disse er følgende:

Type		Hand-ling	Udseende af handling i Comal-80
Tilgang	data repr.		
Sekventiel	ASCII	åbning	10 OPEN FILE 1, "LØN.OPL", READ/WRITE 1)
		lukning	20 CLOSE FILE 1
		Læsning	30 INPUT FILE 1: NAVN $, ADDR1 $,...
		skriv-ning	40 PRINT FILE 1: "Anders", ADD $,...
	BINÆR	åbning	10 OPEN FILE 7, "LAGER.DAT", READ/WRITE 1)
		lukning	20 CLOSE FILE 7
		læsning	30 READ FILE 7: NR, VARE $,.....
		skriv-ning	40 WRITE FILE 7: NUM, V$, ...
Direkte	ASCII	åbning	10 OPEN FILE 2, "SAGER.DAT, RANDOM 42 2)
		lukning	20 CLOSE FILE 2 direkte fil
		læsning	30 INPUT FILE 2 REC: SNR, TP, R$.... 3)
		skriv-ning	40 PRINT FILE 2, REC: N, K $, F, Z,.. 3)
	BINÆR	åbning	10 OPEN FILE 1, "ZAP.TMP", RANDOM, 89 2)
		lukning	20 CLOSE FILE 1
		læsning	30 READ FILE 1, REC: CPR, SKONR,... 3)
		skriv-ning	40 WRITE FILE 1, REC: VALGNR, NAVN$,... 3)

1) Man vælger kun et af ordene READ (ved læsning) eller WRITE (ved skrivning).
2) Tallet 42 (89) angiver hvor mange byte en post (et kartotekskort) fylder i dette tilfælde.
3) Variablen REC indeholder nummeret på den post (kartotekskort) man vil læse eller skrive.

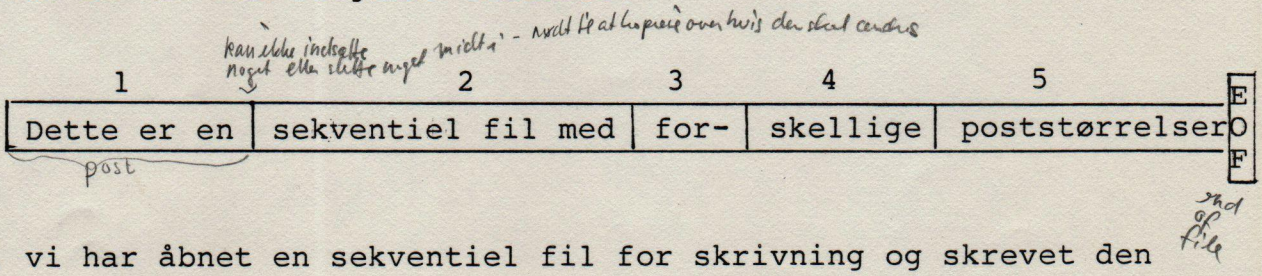
Fremgangsmåden ved anvendelsen af et kartotek (fil) er uanset typen den, at man:

1: Åbner kartoteket (filen)
2: Læser fra og/eller skriver i kartoteket (filen) det nødvendige antal gange.
3: Lukker kartoteket (filen)

Det er altså ikke nødvendigt at åbne og lukke kartoteket hver gang man skal læse eller skrive en post. Man åbner kartoteket og udfører diverse læsninger / skrivninger, hvorefter kartoteket lukkes. Lad os prøve at se på hvad det er der kendetegner de forskellige kartotekstyper.

Sekventielt kartotek (fil)

Et sekventielt kartotek er karakteriseret derved, at det efter åbning kun kan enten skrives eller læses, ikke begge dele. Som navnet lader antyde ligger data sekventielt(på hinanden følgende), og der er ikke plads imellem kartotekskortene til indsættelse af nye informationer. Sagt på en anden måde er sekventielle kartoteker kun velegnet til lagring af data, hvor data ikke skal ændres og der ikke skal indsættes poster midt i kartoteket. Når en sekventiel fil oprettes (skrives), vil der automatisk efter sidste post blive indsat et specielt tegn kaldet End-of-file (ofte forkortet til: EOF), der over for programmet tilkendegiver at filen slutter her. Tegnet EOF er sædvanligvis en byte med den decimale værdi 4. Lad os antage at vi har lavet en fil med følgende udseende:



d.v.s. vi har åbnet en sekventiel fil for skrivning og skrevet den

pågældende tekst i filen, og derefter lukket filen. Hvis vi igen åbner den samme fil til skrivning, vil vi få en fejlmeddelelse der siger, at filen eksisterer i forvejen, og programmet vil stoppe. Ovenstående fil kan fremstilles med flg. program

```
0010 OPEN FILE 3, "TEKST.TMP", WRITE
0020 PRINT FILE 3: "Dette er en"
0030 PRINT FILE 3: "sekventiel fil med"
0040 PRINT FILE 3: "for-"
0050 PRINT FILE 3: "skellige"
0060 PRINT FILE 3: "poststørrelser"
0070 CLOSE FILE 3
0080 END
```

idet vi har kaldt filen for "TEKST.TMP". Den samme fil som vi lige har skrevet kan vi så læse igen lige så mange gange vi ønsker det. Nedenstående program læser fra filen "TEKST.TMP" og udskriver indholdet på terminalens skærm:

```
0010 DIM LINE$ OF 80
0020 OPEN FILE 0, "TEKST.TMP", READ
0030 WHILE NOT EOF(0) DO
0040   INPUT FILE 0: LINE$
0050   PRINT LINE$
0060 ENDWHILE
0070 CLOSE FILE 0
0080 END
```

Vi starter med at åbne filen som nr. 0, hvorefter vi gentager linie 40 og 50 indtil der er End-of-File på fil nr. 0 (EOF(o)).

De indtil nu viste eksempler har alle været med filer af typen "ASCII-filer". Den anden filtype "BINÆR-filer" er pladsbesparende i forhold til 1. type, idet informationerne fylder mindre. Ellers fungerer filerne ens.

ASCII/BINÆR-fil

Den bedste måde at illustrere forskellen på de to måder at repræsentere data på, er ved et lille eksempel. Lad os antage, at vi skal gemme (skrive) tallet 738 på en fil, og lad os se på de to re-

præsentationsformer:

	1. byte (55)	2. byte (51)	3. byte (56)
ASCII:	"7"	"3"	"8"
Bitmønster	0 0 1 1 0 1 1 1	0 0 1 1 0 0 1 1	0 0 1 1 1 0 0 0

BINÆR:	738
Bitmønster	0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0

2 byte

I ASCII-filen gemmes de enkelte cifre i tallet som tegn (et ciffer f.eks. 3, er jo et tegn lige såvel som A eller Z er et tegn), idet hvert eneste tegn der skal gemmes i et elektronisk lager har fået tildelt en fast talværdi (se appendix E: kodeformer). I det viste eksempel ses det at tegnene "7", "3" og "8" ifølge ASCII-tabellen har fået tildelt talværdierne 55, 51 og 56. Tallet 738 fylder altså 3 byte i denne notationsform. I den binære notationsform fylder tallet kun 2 byte, idet 738 her blot er oversat til det tilsvarende binære tal, nemlig 0000001011100010 og dermed fylder 1 byte mindre end i ASCII-repræsentationen.

En ting der er værd at bemærke, er at ASCII-filer direkte kan udskrives på terminalens skærm eller på linieskriveren, hvorimod den binære fil først skal oversættes til en ASCII-fil, før dette kan lade sig gøre.

Direkte kartotek (fil)

Et direkte kartotek er karakteriseret derved, at det efter åbning både kan læses og skrives. I modsætning til det sekventielle kartotek, skal man ved oprettelse af et direkte kartotek angive en posts (et kartotekkorts) størrelse målt i antal byte som data fylder. Følgende tal kan benyttes ved beregning af, hvor meget data fylder:

- heltal (eks: A #) fylder 2 byte
- reelle tal (eks: SKAT) fylder 4 byte
- tekstvariable fylder 2 byte + længden af teksten

Det skal bemærkes, at det kun er ved brug af et binært kartotek disse tal gælder. Benyttes et ASCII-kartotek er hvert ciffer/tegn i et tal at opfatte som en del af en tekst, og skal beregnes som ved tekst.

Lad os antage, at vores kartotekskort skal rumme følgende oplysninger:

Fornavn:	max 20 tegn	fylder 20 + 2 =	22 byte
Efternavn:	max 20 tegn	fylder 20 + 2 =	22 byte
Gade:	max 30 tegn	fylder 30 + 2 =	32 byte
Postnr:	Heltal	fylder	2 byte
Postdistrikt:	max 20 tegn	fylder 20 + 2 =	22 byte
Fradrag:	Reelt tal	fylder	4 byte
Skatteprocent:	Heltal	fylder	2 byte
			106 byte

Alle disse oplysninger gemt i en binær direkte fil, vil ialt fylde 106 byte. Hvis vi vil læse et kort fra dette kartotek (f.eks. kort nr. 18), kan det gøres med flg. program:

```
0010 DIM NAVN$ OF 20, ENAVN$ OF 20, GADE$ OF 30, PDIST$ OF 20
0020 OPEN FILE 9, "STAMDATA.ANS", RANDOM , 106
0030 READ FILE 9, 18: NAVN$, ENAVN$, GADE$, POSTNR#, PDIST$, FRD, SPCT#
0040 PRINT "Fornavn          :";NAVN$
0050 PRINT "Efternavn        :";ENAVN$
0060 PRINT "Gade             :";GADE$
0070 PRINT "Postnummer       :";POSTNR#
0080 PRINT "Postdistrikt     :";PDIST$
0090 PRINT "Fradrag          :";FRD
0100 PRINT "Skatteprocent    :";SPCT#
0110 CLOSE FILE 9
0120 END
```



I programmet er det antaget at vi har kaldt kartoteket for "STAMDATA.ANS".

Når der tidligere står, at kartoteket både kan læses og skrives, betyder det, at man så længe kartoteket er åbent, kan læse og /eller skrive et vilkårligt (eng.: Random) kartotekskort i filen. Man kan med andre ord indsætte kartotekskort et vilkårligt sted i filen.

Hvis man åbner en direkte fil der ikke eksisterer i forvejen, bliver den oprettet, og kartoteket vil være tomt (hvad ellers!). Lad os afslutte dette afsnit med at vise hvordan man kan indsætte et kartotekskort på plads nr. 47:

```
0010 DIM NAVN$ OF 20, ENAVN$ OF 20, GADE$ OF 30, PDIST$ OF 20
0020 OPEN FILE 9, "STAMDATA.ANS", RANDOM , 106
0040 INPUT "Fornavn          :": NAVN$
0050 INPUT "Efternavn        :": ENAVN$
0060 INPUT "Gade             :": GADE$
0070 INPUT "Postnummer       :": POSTNR#
0080 INPUT "Postdistrikt     :": PDIST$
0090 INPUT "Fradrag          :": FRD
0100 INPUT "Skatteprocent    :": SPCT#
0105 WRITE FILE 9, 47: NAVN$, ENAVN$, GADE$, POSTNR#, PDIST$, FRD, SPCT#
0110 CLOSE FILE 9
0120 END
```

Igen gælder det her, at man ingen vegne kommer hvis man ikke prøver sig frem, og sålænge man eksperimenterer på f.eks. en diskette, der udelukkende er afsat til eksperimenter, kan man ikke komme galt af sted.

Prøv for træningens skyld at lave program EJER-udskrift sidst i afsnit 7.1.2. om, så navne på ejerne i stedet for at befinde sig i sættet af tegnfølger EJERE$, nu befinder sig på baggrundslageret under navnet "NAVNE.EJE", og således må læses her fra, før de kan udskrives i en liste.

andre filer der skal læses/skrives. CLOSE er engelsk og betyder "lukke". Man skal altså når man arbejder med filer, både åbne og lukke dem på den anviste måde.

Igen gælder det her, at man ingen vegne kommer hvis man ikke prøver sig frem, og sålænge man eksperimenterer på f.eks. en diskette, der udelukkende er afsat til eksperimenter, kan man ikke komme galt af sted.

Der findes andre formater for læsning af filer, men det overlades til den interesserede læser at rådslå sig med COMAL-80 manualen (evt. læreren). På næste side i tabel 7.4 er lavet en oversigt over samtlige læse- og skriveordrer i COMAL-80, hvor skrivesætningerne beskrives i afsnit 7.1.4.4.

Prøv for træningens skyld at lave program EJER-udskrift sidst i afsnit 7.1.2. om, så navne på ejerne i stedet for at befinde sig i sættet af tegnfølger EJERE$, nu befinder sig på baggrundslageret under navnet "NAVNE.EJE", og således må læses her fra, før de kan udskrives i en liste.

Tabel 7.4. Læsning og skrivning.

Gruppe	Nøgleord		Type	Kommentarer	Behandles i afsnit nr.
	Skrivning	Læsning			
Internt	POKE	PEEK	X	Maskinsprogs-funktioner	Behandles ikke
	DATA	READ	X	Internt i program	7.1.4.3.
Via Termi-nal	PRINT USING	INPUT	Ascii	1) Brug af komma og semikolon er ens for print og input 2) Format ved fil-brug er helt analogt med format ved teminal-brug	Behandles ikke
	PRINT				7.1.4.4.
Bruger-filer i program-mer	PRINT FILE USING	INPUT FILE			Behandles ikke
	PRINT FILE				7.1.4.4.
	WRITE FILE	READ FILE	Binært	Pladsbesparende	Behandles ikke
Filer med helpro-gram-mer	LIST	ENTER	Ascii	Hentning og gemning af hele programmer på baggrundslager	se appendix A
	SAVE	LOAD	Binært		

Bemærk at ved åbning af en brugerfil skal man bruge en OPEN-sætning afsluttet af et af nøgleordene, READ, WRITE eller RANDOM.

Disse nøgleord bruges også i helt andre forbindelser, nemlig h.h.v.:

READ-sætninger, READ-FILE-sætninger, WRITE FILE-sætninger samt RANDOM-sætninger eller RANDOM-kommandoer.

7.1.4.4. SKRIVESÆTNINGER

Når vi snakker om at skrive fra et program, betyder det at lagre data (analyseværdier, navnelister m.v.) på et eller andet medie. Man kunne forestille sig at programmet EJER-udskrift skrev en liste af ejere på et stykke papir (via linieskriveren) som vist i tabel 7.2. på side 7.13. Den samme liste kunne være skrevet ud på terminalens skærm. I begge tilfælde sker udskrivningen v.h.a. en skrivesætning med ordren PRINT (eng. PRINT = skriv). Der er i mange af de foregående eksempler vist brugen af PRINT, og som det mere eller mindre fremgår heraf, kan følgende ting udskrives:

1: Indhold af variable (både tekst-variable og numeriske variable).
2: Værdien af et udtryk (både tekstudtryk og numeriske udtryk).
3: Konstanter (både tekst- og talkonstanter).

Efter ordren PRINT kan der angives en række af variable, udtryk og konstanter i en "pærevælling", idet disse dog skal adskilles med komma (,) eller semikolon (;). Følgende eksempler på PRINT-sætninger kan angives:

```
0010 PRINT ANTAL#,VARE$
0020 PRINT PI*RADIUS^2,"KVADRAT"+ENHED$
0030 PRINT 48,"Kr."
0040 PRINT ANTAL#;"Kr. på";TYPE$(RENTE);
```

Linie 10 viser udskrivning af variabelindhold (ANTAL# og VARE$) der f.eks. kunne give udskriften:

17 JERNPLADER

hvis ANTAL# indeholdt tallet 17 og VARE$ indeholdt teksten "JERNPLADER".

Linie 20 viser udskrivning af udtryk, d.v.s. udtrykkene beregnes med de aktuelle værdier af de i udtrykkene indgående variable før der udskrives. Her kunne PI være 3.141593 og R=10 i første udtryk, og ENHED$="CENTIMETER" i andet udtryk, hvilket ville give udskrift som:

314.593 KVADRATCENTIMETER

Linie 30 viser udskrivning af konstanter hvor udskriften vil blive:

48 kr.

Linie 40 viser en "pærevælling" af de i linie 10,20 og 30 viste udskrivningstyper (altså variable, udtryk og konstanter blandet sammen). Hvis ANTAL = 19, RENTE = 1, TYPE$(1) = "6 MDR" og TYPE$(2)= "9 MDR" overlades det nu til læseren at vise udskriften fra skrivesætningen i linie 40. Der eksisterer nogle helt eksakte regler for, hvordan udskrifter kommer til at se ud, idet formatet dels er fastlagt ved standardregler og dels ved brugerens (dine egne) tabuleringsregler (som på en skrivemaskine). Det vil dog i denne afkortede beskrivelse føre for vidt, at pensle alle formatteringsreglerne ud i detaljer. Det eneste vi vil nævne her, er at semikolon ";" som afslutning på en PRINT-sætning, betyder at videre udskrivning vil foregå i forlængelse af det allerede skrevne, hvor videre udskrivning normalt vil fortsætte på næste linie. D.v.s. at afslutning af en skrivesætning uden ";" helt vil svare til en vognretur på en skrivemaskine. Hvis vi ser på nedenstående program:

```
0010 PRINT "ABE"
0020 PRINT "BJØRN"
0030 PRINT "OKSE"," HUND"
0040 PRINT "PØLSE";
0050 PRINT " MED";
0060 PRINT " DET HELE"
0070 PRINT "JA TAK !"
0080 END
```

vil vi få følgende udskrift på skærmen:

```
ABE
BJØRN
OKSE HUND
PØLSE MED DET HELE
JA TAK !
```

Prøv at finde ud af hvorfor udskriften ser således ud. I det følgende program skal du forsøge at vise hvilken udskrift der kommer på skærmen:

```
0010 DIM ENHED$ OF 5
0020 ENHED$:="Meter"
0030 ANTAL:=835.4
0040 PRINT "REKORDTAVLE:"
0050 PRINT "------------"
0060 PRINT "Rekorden i sækkeløb på tid = ";
0070 PRINT ANTAL," ";ENHED$
0080 PRINT "Hurra !!"
0090 PRINT "------------"
0100 END
```

Hvis man ønsker detaljerede oplysninger om skriveformater kan man kigge i COMAL-80 manualen s. 5-16.

Det er således, at med mindre andet er angivet, vil udskrivning ske på terminalens skærm. Dette kan man ændre på, ved at bruge en sætning der hedder SELECT OUTPUT (eng.: select output = vælg uddatamedie), hvor man efter de to ord angiver, hvor man vil udskrive. For linieskriveren skal man benytte betegnelsen "LP:" og for dataskærmen "DS:". Hvis man på en nem måde vil udskrive på baggrundslageret kan man blot angive f.eks. "RADIER.DAT". Som eksempel vil vi her angive et program hvor man først udskriver på dataskærmen, dernæst på henholdsvis linieskriveren, baggrundslageret og endelig igen på dataskærmen:

```
0010 // PROGRAM UDSKRIV.
0020 PRINT "Denne tekst udskrives på SKÆRMEN."
0030 SELECT OUTPUT "LP:"
0040 PRINT "Denne tekst udskrives på LINIESKRIVEREN."
0050 SELECT OUTPUT "MINFIL"
0060 PRINT "Denne tekst udskrives på BAGGRUNDSLAGERET"
0070 PRINT "under navnet -MINFIL- ."
0080 SELECT OUTPUT "DS:"
0090 PRINT "Nu er vi tilbage på SKÆRMEN."
0100 END
```

Det sidste vi vil nævne i forbindelse med skrivning er, at man på nøjagtig samme måde som man åbnede (OPEN) filer ved læsning, kan åbne filer ved skrivning, ved blot at angive ordet "WRITE", som nøgleord i stedet for "READ" på følgende måde:

```
10 OPEN FILE 4, "MINFIL",WRITE
```

Herefter kan man, ved efter PRINT at angive nummeret 4, skrive på filen "MINFIL", på følgende måde:

```
0010 OPEN FILE 4, "MINFIL", WRITE
0020 PRINT FILE 4: "Atomvægt ";19.38
0030 PRINT FILE 4: "Lagkage."
0040 CLOSE FILE 4
0050 END
```

Skrivesætningerne i dette program bevirker, at der i filen "MINFIL" skrives teksten:

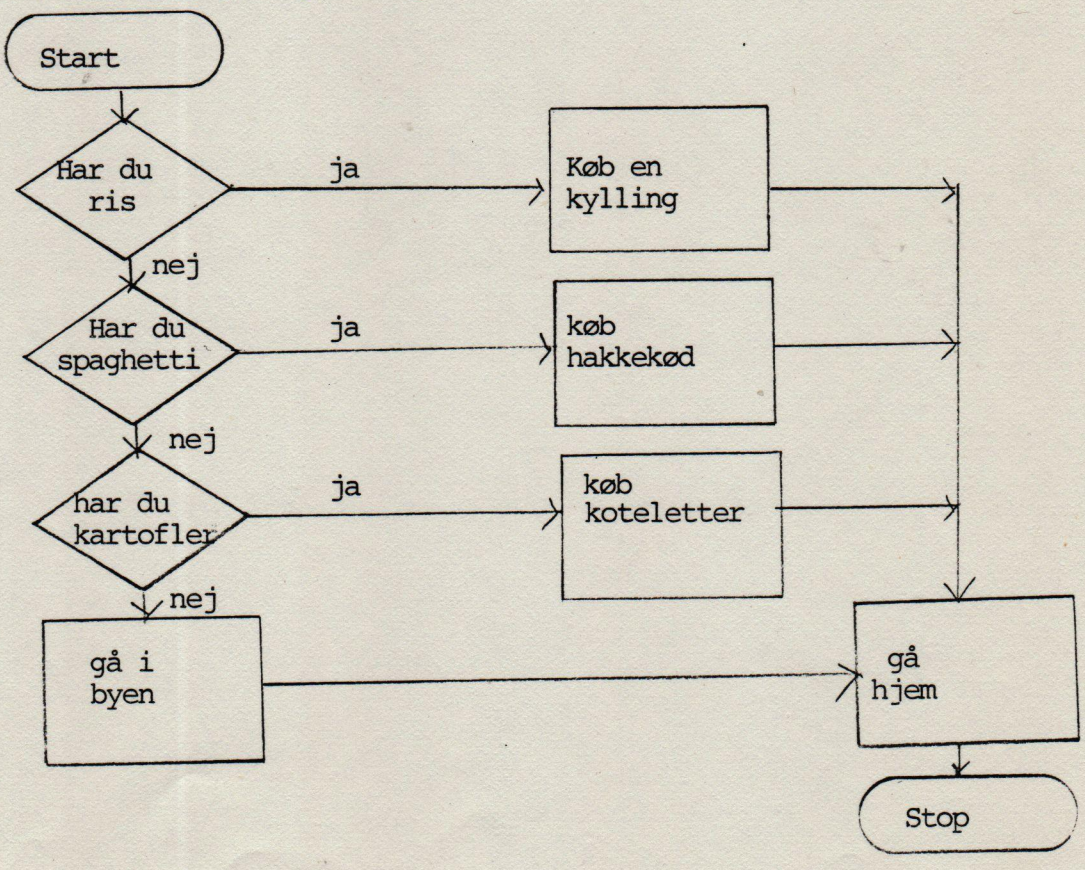
Atomvægt 19.38
Lagkage.

Sætningen CLOSE er omtalt under afsnittet "læsning".

For specielt interesserede, er der her igen mulighed for at konsultere COMAL-80 manualen for yderligere eksempler og detaljer.

7.1.4.5. BETINGEDE SÆTNINGER.

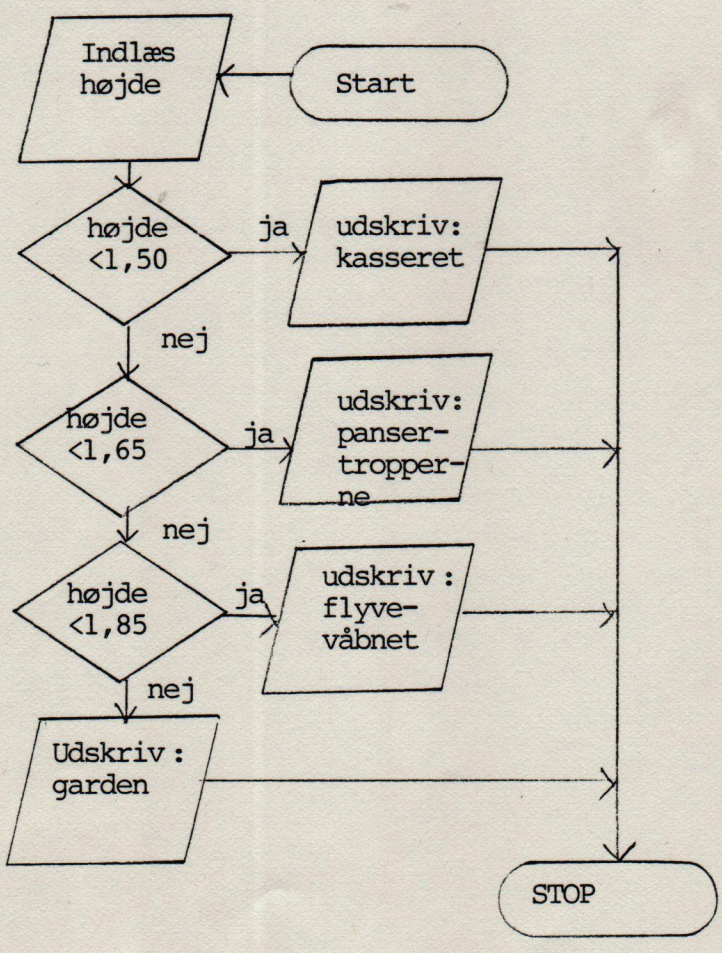
Hverdagen er fyldt med små og store beslutninger. Man må hele tiden stille sig selv spørgsmål (måske ubevidst) med, hvis nu det og det er opfyldt så gør jeg sådan, ellers gør jeg noget andet. Er man f.eks. hos slagteren og skal beslutte sig for noget kød til aftensmaden, kan man nemt komme i følgende situation: Man har lyst til f.eks. kotelet hakkekød eller kylling. For at træffe en afgørelse gør man sig måske flg. overvejelser: Hvis jeg har ris derhjemme tager jeg kylling, ellers hvis jeg har spaghetti tager jeg hakkekød ellers hvis jeg har kartofler tager jeg koteletter. Hvis jeg ikke har nogen af delene, går jeg i byen og spiser. Hvad man gør i den aktuelle situation afhænger af en masse hvis'er. Tegner vi denne beslutningsproces i et blokdiagram kan den antage følgende udseende:



Hvad man gør i den aktuelle situation er altså betinget af at bestemte forudsætninger er opfyldt. Man kan komme i en lignende situation når man skal lave et program, der skal løse et eller andet problem. Lad os antage at vi skal lave et program der kan afgøre, om en værnepligtig skal i garden, flyvevåbnet, pansertropperne eller skal kasseres. Det eneste der afgør hvad han skal, er hans højde efter følgende regler:

Max. højde	Min. højde	beslutning
ingen	1,85	garden
1,84	1,65	flyvevåbnet
1,64	1,50	pansertropperne
1,49	ingen	kasseres

Det eneste programmet skal vide inden beslutningen træffes, er altså personens højde som vi genialt nok kalder for HØJDE. Det eneste vi forlanger af det færdige program er en udskrift, der fortæller os om beslutningen. Løsningen i blokdiagramform kan antage følgende udseende:



Da vi nu skal oversætte dette blokdiagram til COMAL-80, er tiden kommet til at introducere sætningstyper, der ud fra en sammenligning af to eller flere værdier, kan dirigere programforløbet. Disse sætningstyper kaldes for betingede sætninger, da det videre programforløb jo er betinget af et valg.

Lad os starte med at kigge på en betinget sætning der kaldes en IF-sætning. I sin simpleste form ser den således ud:

IF betingelse THEN sætning

eks.: IF HØJDE<1.85 THEN PRINT "flyvevåbnet"

betingelse sætning

og kan oversættes med: Hvis (IF) indholdet af variablen HØJDE er mindre end (<) 1,85 så (THEN) udskriv teksten (PRINT) "flyvevåbnet".

Hvis vi oversætter blokdiagrammet v.h.a. denne sætningstype fås følgende program:

```
0010 // PROGRAM VÆRNEPLIGT.
0020 INPUT "Indtast højde:": HØJDE
0030 IF HØJDE<1.5 THEN GOTO 0120
0040 IF HØJDE<1.65 THEN GOTO 0100
0050 IF HØJDE<1.85 THEN GOTO 0080
0060 PRINT "Garden"
0070 GOTO 0130
0080 PRINT "Flyvevåbnet"
0090 GOTO 0130
0100 PRINT "Pansertropperne"
0110 GOTO 0130
0120 PRINT "Kasseret"
0130 END
```

~ til dansk nummerering af blokdiagrammet på 7.38

kan laves i basic

Hvis betingelsen i IF-sætningen ikke er opfyldt, udføres sætningen efter IF-sætningen, ellers udføres sætningen efter ordet THEN. Vi har her indført endnu en sætningstype GOTO, der bevirker et hop til det linienummer der står efter ordet GOTO. I linierne 30,40 og 50 hoppes der altså til linienumrene 120, 100 og 80 henholdsvis, hvis de indledende betingelser ellers er opfyldt. Det er jo lidt omstændigt med alle disse goto-sætninger. Heldigvis eksisterer der jo andre former for betingede sætninger.

Den næste vi vil se på har følgende udseende:

```
IF betingelse THEN
sætning
sætning
.
.
.
ENDIF
```

Denne sætning indebærer, at sætningerne der står mellem IF og ENDIF bliver udført hvis betingelserne er opfyldt, ellers fortsættes der i linien umiddelbart efter ENDIF. Vi vil nu se hvorledes program værnepligt kommer til at se ud ved anvendelse af denne konstruktion:

```
0010 // PROGRAM VÆRNEPLIGT.
0020 INPUT "Indtast højde:": HØJDE
0030 IF HØJDE<1.5 THEN
0040  PRINT "Kasseret"
0050  GOTO 0160
0060 ENDIF
0070 IF HØJDE<1.65 THEN
0080  PRINT "Pansertropperne"
0090  GOTO 0160
0100 ENDIF
0110 IF HØJDE<1.85 THEN
0120  PRINT "Flyvevåbnet"
0130  GOTO 0160
0140 ENDIF
0150 PRINT "Garden"
0160 END
```

Vi ser her at der er kommet nogle flere linier, men til gengæld er programmet blevet lidt "pænere", lidt mere struktureret.

En tredie type sætning vi skal se på, ser således ud:

```
IF betingelse THEN
sætning
sætning
.
.
.
ELSE
sætning
sætning
.
.
.
ENDIF
```

Til forskel fra den forrige type, er der her kommet et ELSE mere på, hvilket betyder at sætningerne mellem IF og ELSE udføres hvis betingelse er opfyldt, ellers udføres sætningerne mellem ELSE og ENDIF.

Med denne konstruktion kan programmet antage følgende udseende:

```
0010 // PROGRAM VÆRNEPLIGT.
0020 INPUT "Indtast højde:": HØJDE
0030 IF HØJDE<1.5 THEN
0040  PRINT "Kasseret"
0050 ELSE
0060  IF HØJDE<1.65 THEN
0070   PRINT "Pansertropperne"
0080  ELSE
0090   IF HØJDE<1.85 THEN
0100    PRINT "Flyvevåbnet"
0110   ELSE
0120    PRINT "Garden"
0130   ENDIF
0140  ENDIF
0150 ENDIF
0160 END
```

Vi ser her at der er det samme antal linier som før, men vi har helt undgået GOTO-sætningerne, og programmet følger mere en naturlig spørgemåde, nemlig som skitseret i indledningen med besøg hos slagteren. Herudover bliver programmeringen uafhængig af linienumrene, idet der jo ikke er en eneste reference (GOTO) til et linienummer.

Den sidste konstruktion vi skal på af denne art, har følgende opbygning:

```
IF betingelse 1 THEN
sætning
sætning
.
.
.
ELIF betingelse 2 THEN
sætning
sætning
.
.
.
ELIF betingelse 3 THEN
sætning
sætning
.
```

```
.
.
.
ELIF betingelse  THEN
.
.
.
ELSE
sætning
sætning
.
.
.
ENDIF
```

Hvis første betingelse i denne konstruktion er opfyldt udføres sætningerne mellem IF og ELIF (else if), ellers prøves næste betingelse om den er opfyldt, og er dette tilfældet udføres sætningerne mellem de to ELIF, ellers prøves næste betingelse om den er opfyldt o.s.v.. Med andre ord, så udføres sætningerne for den første betingelse der er opfyldt, hvorefter der hoppes til første sætning efter ENDIF.

Hvis vi laver program værnepligt med denne konstruktion vil den få følgende udseende:

```
0010 // PROGRAM VÆRNEPLIGT.
0020 INPUT "Indtast højde:": HØJDE
0030 IF HØJDE<1.5 THEN
0040  PRINT "Kasseret"
0050 ELIF HØJDE<1.65 THEN
0060  PRINT "Pansertropperne"
0070 ELIF HØJDE<1.85 THEN
0080  PRINT "Flyvevåbnet"
0090 ELSE
0100  PRINT "Garden"
0110 ENDIF
0120 END
```

Denne udgave af programmet viser klart, at vores problem var "skræddersyet" til denne konstruktion, eller snarere omvendt, konstruktio-

nen var skræddersyet til problemet. Jeg vil her benytte lejligheden til at sige, at mange problemer i virkeligheden ligner hinanden i udseende, og hvor det bare er nogle andre størrelser der afgør programforløbet, og nogle andre ting der skal beregnes. Denne sidste konstruktion kan derfor med fordel anvendes på lignende problemer.

Betingelsen der er angivet efter alle IF'erne, kan i virkeligheden være en serie betingelser der skal være opfyldt, før sætningen/sætningerne efter THEN udføres.

En serie af betingelser kan kobles sammen med de dyadiske operatorer AND og OR, og hertil kommer så, at man kan vende betingelsen om, altså negere den, med den monadiske operator NOT. Hvis vi f.eks. skal give udtryk for, at HØJDE <1,85 og der samtidig skal gælde at skonummeret SKONR<45, før en betingelse er opfyldt skriver vi blot

```
10 IF HØJDE <1.85 AND SKONR<45 THEN.
```

Hvis betingelsen havde været HØJDE <1.85 eller SKONR<45 skulle vi blot have skrevet

```
10 IF HØJDE <1.85 OR SKONR <45 THEN
```

Hvis betingelsen havde været, at HØJDE ikke måtte være mindre end 1.85 skulle vi blot skrive

```
10 IF NOT (HØJDE <1.85) THEN
```

NOT vender sandhedsværdien om

Enhver betingelse er enten opfyldt (SAND) eller ikke opfyldt (FALSK), så en betingelse er altså et logisk udtryk, der kan antage værdien SAND eller FALSK, hvilket i COMAL-80 hedder TRUE eller FALSE. Om operatorerne AND, OR og NOT, kan vi anføre følgende skema, idet vi tager udgangspunkt i to betingelser A og B der enten kan være TRUE eller FALSE:

A	B	A AND B	A OR B	NOT A	NOT (A AND B)
False	False	False	False	True	True
False	True	False	True	True	True
True	False	False	True	False	True
True	True	True	True	False	False

Lad os kigge på et par eksempler hvor vi antager at HØJDE=1.78 og SKONR=43 og beregne en betingelses logiske værdi:

h = 1,78
skonr = 43

Betingelse:

```
HØJDE < 1.65 AND SKONR=43 OR HØJDE>1.75
    FALSE    AND    TRUE  OR   TRUE
            FALSE  OR TRUE
                  TRUE
```

Resultatet blev altså at den givne betingelse var opfyldt idet den logiske værdi blev TRUE.

Lad os prøve en til:

Betingelse:

```
SKONR>45 OR  NOT (HØJDE=1.78 AND SKONR<44)
   FALSE                TRUE          TRUE
   FALSE  OR  NOT                TRUE
   FALSE     OR          FALSE
            FALSE
```

Denne betingelse var altså ikke opfyldt, da den logiske værdi blev FALSE. Prøv at beregne den logiske værdi af flg. betingelser (HØJDE og SKONR som før):

1: HØJDE>1.90 AND NOT (SKONR<42 OR SKONR>43)

2: NOT (SKONR>46 AND HØJDE<1.90) OR HØJDE >1.80

3: NOT (NOT (HØJDE>1.45 OR NOT (HØJDE <1.90)))

De operatorer der benyttes til at afgøre en relations logiske værdi kaldes for en relationsoperator. Følgende relationsoperatorer der alle er dyadiske, eksisterer i COMAL-80.

Operator	Matematisk	Betydning
>	>	Større end
<	<	Mindre end
>=	≥	Større end eller lig med
<=	≤	Mindre end eller lig med
=	=	Lig med
<>	≠	Forskellig fra

Efter denne gennemgang af forskellige operatorer, vil vi se på den sidste af de i COMAL-80 eksisterende betingede sætninger, nemlig CASE-sætningen der har følgende opbygning:

```
CASE udtryk
WHEN udtryk, udtryk,......
sætning
sætning
.
.
.
.
sætning
WHEN udtryk, udtryk.....
sætning
sætning
.
.
.
.
sætning
WHEN...........
.
.
.
.
OTHERWHISE
sætning
sætning
.
.
.
.
ENDCASE
```

I denne konstruktion beregnes værdien af udtrykket efter ordet CASE hvorefter udtrykkene i WHEN-sætningerne gennemløbes, indtil der findes en værdi, som er identisk med værdien af det først beregnede udtryk. Hvis en sådan identisk værdi er fundet, udføres sætningerne mellem WHEN og det næste WHEN, svarende til den pågældende WHEN-sætning.

Hvis der ikke blandt udtrykkene i WHEN-sætningerne findes en værdi, som er identisk med den beregnede værdi efter CASE, udføres de alternative sætninger der er angivet efter ordet OTHERWISE.
Hvis CASE-konstruktionen ikke indeholder alternative sætninger, vil

programmet i ovennævnte tilfælde stoppe med en fejlmeddelelse.

Som eksempel kan vi antage at vi har en kasse med kugler af forskellige farver. Vi ønsker nu at optælle antallet af RØDE, GULE og GRØNNE kugler fra kassen. Samtidig ønsker vi at tælle SORTE og BLÅ kugler under ét. Alle andre farve kugler tælles sammen under ét. Først når farven STOP indlæses stopper programmet:

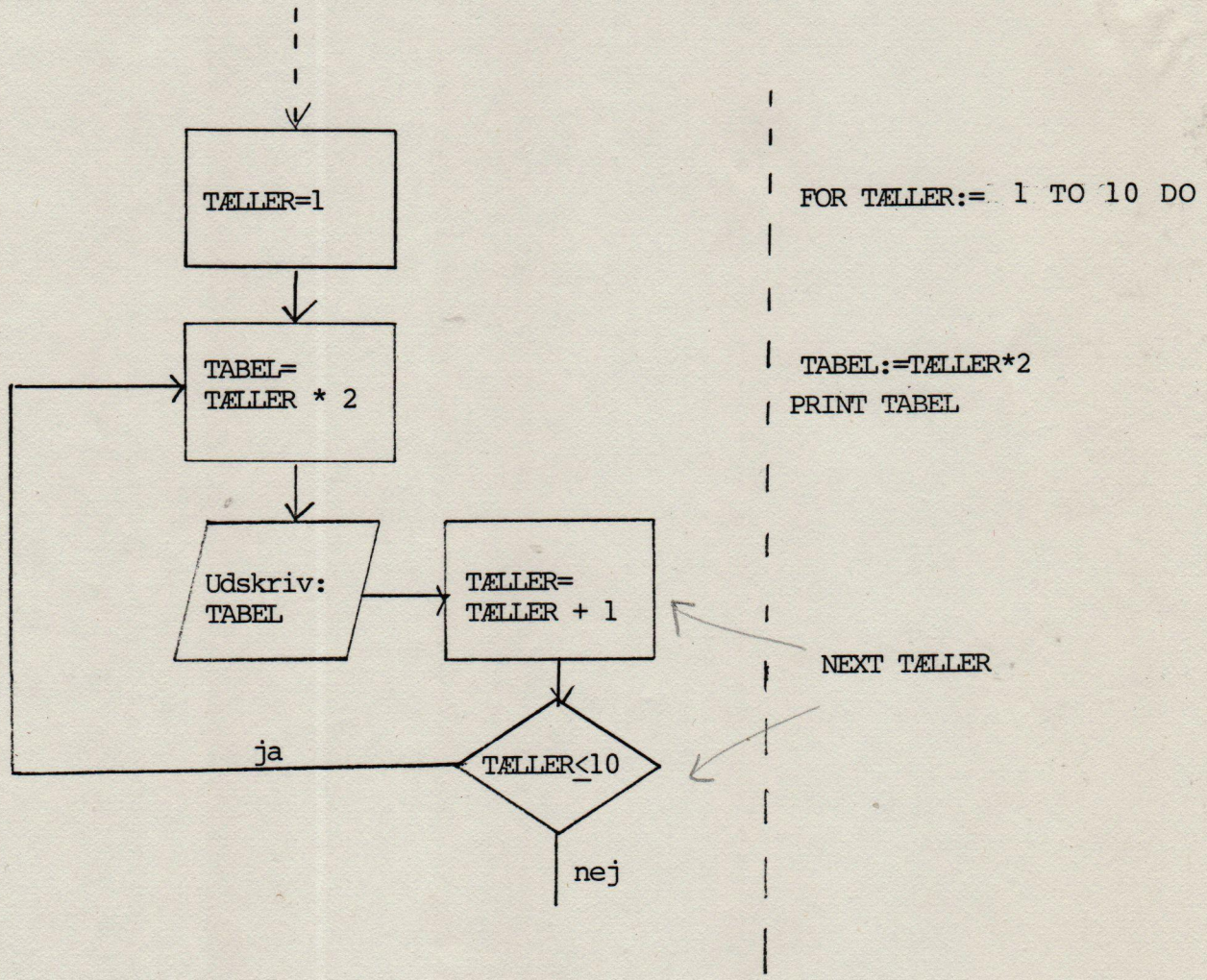
```
0010 // PROGRAM FARVER.
0020 DIM FARVE$ OF 10, TÆLLER(5)
0030 MAT TÆLLER:=0
0040 INPUT "Indtast farve:": FARVE$
0050 IF FARVE$<>"STOP" THEN
0060  CASE FARVE$ OF
0070  WHEN "RØD"
0080   TÆLLER(1):=TÆLLER(1)+1
0090  WHEN "GUL"
0100   TÆLLER(2):=TÆLLER(2)+1
0110  WHEN "GRØN"
0120   TÆLLER(3):=TÆLLER(3)+1
0130  WHEN "SORT", "BLÅ"
0140   TÆLLER(4):=TÆLLER(4)+1
0150  OTHERWISE
0160   TÆLLER(5):=TÆLLER(5)+1
0170  ENDCASE
0180  GOTO 0040
0190 ELSE
0200  PRINT "RØDE ";TAB(14);TÆLLER(1);" stk."
0210  PRINT "GULE ";TAB(14);TÆLLER(2);" stk."
0220  PRINT "GRØNNE ";TAB(14);TÆLLER(3);" stk."
0230  PRINT "SORTE + BLÅ ";TAB(14);TÆLLER(4);" stk."
0240  PRINT "ANDRE FARVER ";TAB(14);TÆLLER(5);" stk."
0250 ENDIF
0260 END
```

Udskriften på skærmen kunne forløbe således, idet understreget tekst er det brugeren (dig selv) indtaster:

```
Indtast farve: SORT
Indtast farve: BLÅ
Indtast farve: TYRKIS
Indtast farve: RØD
Indtast farve: BLÅ
Indtast farve: VIOLET
Indtast farve: GRØN
Indtast farve: GRØN
Indtast farve: LILLA
Indtast farve: RØD
Indtast farve: STOP
RØDE           2  stk.
GULE           0  stk.
GRØNNE         2  stk.
SORTE + BLÅ    3  stk.
ANDRE FARVER   3  stk.
```

7.1.4.6. REPETERENDE SÆTNINGER

Programmer laves tit til løsning af problemer hvor en beregning /proces skal gentages et vist antal gange eller indtil en eller anden betingelse er opfyldt, hvorefter programmet går videre i forløbet. For at kunne lave programmerne så de virker strukturerede og let overskuelige, har man konstrueret nogle sætningstyper, hvor et antal sætninger gentages indtil en betingelse er opfyldt. Der eksisterer 4 typer af repeterende sætninger som de kaldes, hvor forskellen ligger i hvornår det afgøres om der skal fortsættes, og hvad der afgør det. Den første type vi vil omtale, er FOR-TO-STEP-NEXT sætningen, som vi jo allerede har set eksempler på i det foregående stof. Den nemmeste måde at forklare sætningens virkemåde på, er at tegne et blokdiagram for sætningen, idet vi tager udgangspunkt i programmet i afsnit 7.1 der beregnede og udskrev 2-tabellen:



I denne konstruktion angiver man hvor mange gange løkken skal gennemløbes, idet variablen efter ordet FOR (styrevariablen), får tildelt en startværdi (her 1), der så efter hvert gennemløb af løkken tælles op (ned) med trinværdiens størrelse, der er angivet efter ordet STEP (her 1). Først når stopværdien der er angivet efter ordet TO (her 10) overskrides stopper løkken, og der fortsættes i sætningen umiddelbart

efter NEXT sætningen. Et gennemløb af løkken starter i sætningen umiddelbart efter FOR-sætningen og slutter med at styrevariablen bliver talt op med trinværdien når sætningen NEXT nås, hvorefter løkken gennemløbes igen hvis førnævnte stopbetingelse ikke er opfyldt.

I stedet for TO kan ordet DOWNTO benyttes, hvilket blot betyder at trinværdien negeres, d.v.s. trinværdi:=-trinværdi. Hvis trinværdien er 1 kan ordet STEP og trinværdi udelades.

For god ordens skyld skal det nævnes, at man sagtens kan have flere repeterende løkker inden i hinanden (gælder også de øvrige typer). Som eksempel kan vi angive et program der beregner og udskriver tabellerne fra 2 til 10:

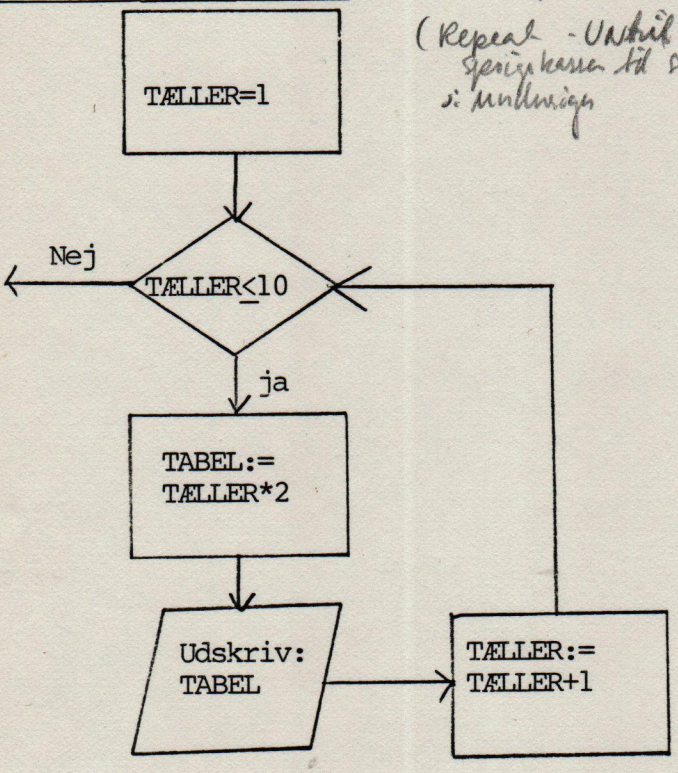
```
0010 // PROGRAM TABELLER (2-10).
0020 PRINT "TABEL:";
0030 FOR TABEL:=2 TO 10 DO
0040   PRINT TAB((TABEL-1)*4+6);TABEL;
0050 NEXT TABEL
0060 PRINT "-----------------------------------------"
0070 FOR FAKTOR:=1 TO 10 DO
0080   PRINT TAB(5-INT(FAKTOR/10));FAKTOR;":";
0090   FOR TABEL:=2 TO 10 DO
0100     VÆRDI:=FAKTOR*TABEL
0110     PRINT TAB((TABEL-1)*4+6);VÆRDI;
0120   NEXT TABEL
0130   PRINT
0140 NEXT FAKTOR
0150 PRINT "-----------------------------------------"
0160 END
```

```
TABEL:    2    3    4    5    6    7    8    9   10
-----------------------------------------------------
    1 :   2    3    4    5    6    7    8    9   10
    2 :   4    6    8   10   12   14   16   18   20
    3 :   6    9   12   15   18   21   24   27   30
    4 :   8   12   16   20   24   28   32   36   40
    5 :  10   15   20   25   30   35   40   45   50
    6 :  12   18   24   30   36   42   48   54   60
    7 :  14   21   28   35   42   49   56   63   70
    8 :  16   24   32   40   48   56   64   72   80
    9 :  18   27   36   45   54   63   72   81   90
   10 :  20   30   40   50   60   70   80   90  100
-----------------------------------------------------
```

Den næste repeterende sætning vi vil se på er en WHILE-sætning hvor man til forskel fra før, i starten af løkken angiver, om den skal gentages. Opbygningen er simpel:

```
WHILE betingelse
sætning
sætning
  :
sætning
ENDWHILE
```

og i blokdiagramform



Program

```
WHILE TÆLLER<=10 DO
TABEL:=TÆLLER*2
PRINT TABEL
TÆLLER:=TÆLLER+1
ENDWHILE
```

Når en WHILE-sætning udføres, udregnes værdien (sand eller falsk) af betingelsen efter WHILE. Hvis værdien er TRUE fortsættes programudførelsen i den efterfølgende sætning. Er værdien derimod FALSE fortsættes programudførelsen i sætningen umiddelbart efter ENDWHILE-sætningen.

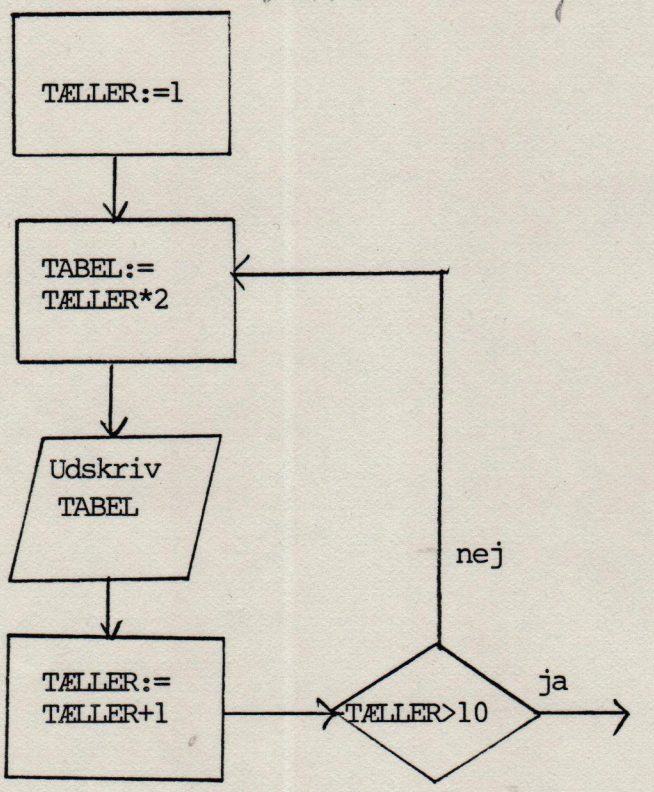
Lad os prøve indføre WHILE-konstruktionen i programmet FARVER der er angivet sidst i afsnit 7.1.4.5: (se næste side)

```
0010 // PROGRAM FARVER.
0020 DIM FARVE$ OF 10, TÆLLER(5)
0030 MAT TÆLLER:=0
0035 FARVE$:=" "
0037 WHILE FARVE$<>"STOP" DO
0040  INPUT "Indtast farve:": FARVE$
0060  CASE FARVE$ OF
0070  WHEN "RØD"
0080   TÆLLER(1):=TÆLLER(1)+1
0090  WHEN "GUL"
0100   TÆLLER(2):=TÆLLER(2)+1
0110  WHEN "GRØN"
0120   TÆLLER(3):=TÆLLER(3)+1
0130  WHEN "SORT", "BLÅ"
0140   TÆLLER(4):=TÆLLER(4)+1
0150  OTHERWISE
0160   TÆLLER(5):=TÆLLER(5)+1
0170  ENDCASE
0180 ENDWHILE
0190 TÆLLER(5):=TÆLLER(5)-1
0200 PRINT "RØDE ";TAB(14);TÆLLER(1);" stk."
0210 PRINT "GULE ";TAB(14);TÆLLER(2);" stk."
0220 PRINT "GRØNNE ";TAB(14);TÆLLER(3);" stk."
0230 PRINT "SORTE + BLÅ ";TAB(14);TÆLLER(4);" stk."
0240 PRINT "ANDRE FARVER ";TAB(14);TÆLLER(5);" stk."
0260 END
```

Den 3. type vi vil se på er en REPEAT-UNTIL -sætning, hvor man i slutningen af løkken angiver, om man skal undlade at fortsætte i løkken. Konstruktionen er lige så simpel som den foregående:

```
REPEAT
sætning
sætning
:
:
sætning
UNTIL betingelse
```

Og som blokdiagram



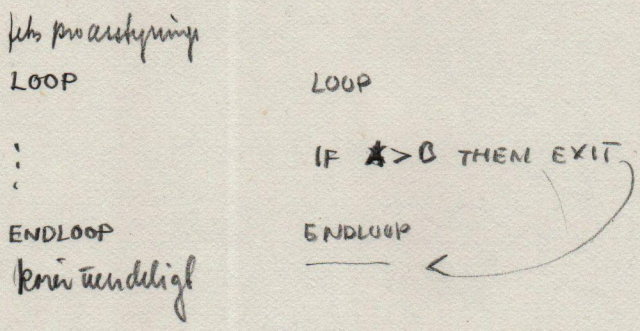
Program

```
REPEAT
TABEL:=TÆLLER*2

PRINT TABEL
TÆLLER:=TÆLLER+1

UNTIL TÆLLER>10
```

Når en REPEAT-UNTIL sætning udføres vil løkken gennemløbes indtil betingelsen efter UNTIL-sætningen er TRUE, hvorefter programudførelsen fortsætter umiddelbart efter UNTIL-sætningen. Er betingelsen FALSE fortsætter løkken i sætningen lige efter REPEAT-sætningen. REPEAT-sætningen har altså ingen anden funktion end at markere starten på løkken.

Den sidste type af repeterende sætninger hedder LOOP-ENDLOOP, og vil blot gentage sætningerne mellem LOOP og ENDLOOP, indtil man v.h.a. en GOTO-sætning eller EXIT hopper ud af den. EXIT-sætningen bevirker at programudførelsen fortsætter umiddelbart efter ENDLOOP.

Prøv at lave programmet med 2-tabellen (se afsnit 7.1.) v.h.a. denne konstruktion.



7.1.4.7 Procedurer og funktioner

Dette afsnit er kun en orientering om procedurer (og funktioner) og deres anvendelse. For en detaljeret beskrivelse af procedurer og funktioner, henvises til Comal-80 manualen i afsnit 5.11 side 5.49.

Hvem har ikke prøvet at skifte gear i en bil eller i hvert fald set andre gøre det, eller hvem har ikke prøvet at ringe til venner og bekendte over telefonen? Det at skifte gear eller at dreje/trykke et nummer på telefonen, er begge handlinger der igangsætter en serie af på forhånd fastlagte transaktioner. Ved gearskiftet ændres nogle tandhjuls stilling i forhold til hinanden, og ved opringningen er der en masse relæer hos telefonvæsenet der ændrer tilstand. Hvilke tandhjul i gearkassen eller hvilke relæer der ændrer tilstand er skjult for den der udfører handlingen. Tænk hvis man hver gang man f.eks. skulle ringe op, skulle fortælle telefonvæsenet hvilke relæer der skulle kobles til og fra, så kunne man få nok at lave med at sætte sig ind i hvordan et telefonsystem fungerer!

Sådan er det heldigvis ikke, idet vi jo blot drejer nummeret, eller skifter gear, hvorefter der sker en masse ting "bag ryggen" på os. Vi behøver altså blot at vide, hvad resultatet af en bestemt handling bliver, for at vi kan anvende systemet (gearkasse/telefon), uden at bekymre os om de "skjulte handlinger".

Noget lignende gør sig gældende inden for programmering, idet man over for personer uden indgående kendskab til f.eks. en Analog/Digital-konverter tilsluttet EDB-systemet, kan stille underprogrammer til rådighed, hvor brugeren så blot aktiverer underprogrammet og får udført en bestemt handling (f.eks. aflæs spændingen på A/D-konverterens kanal 2). Underprogrammet vil så udføre nogle ordrer der er skjult for brugeren, og levere det ønskede resultat.

Det vi her kalder et underprogram betegnes også som en procedure eller som en funktion. En procedure kan altså bruges som et let betjenbart "stykke værktøj", til opnåelse af avancerede faciliteter. Dermed er ikke sagt at procedurer i al almindelighed er eller skal være avancerede, idet en procedure meget vel kan anvendes til simple beregninger, hvor disse beregninger f.eks. benyttes flere steder i det samme program.

Inden vi går over til et eksempel, vil vi lige se på hvordan man laver en procedure i COMAL-80. Opbygningen er som følger:

```
PROC     navn (evt. parametre)
  .
  .
  .
sætninger
  .
  .
  .
ENDPROC navn
```

hvor "navn" er det navn, hvormed proceduren kaldes (aktiveres). PROC og ENDPROC afgrænser proceduren og sætninger er almindelige Comal-80 sætninger. Det der hedder "(evt. parametre)" er foreløbige navne for nogle variable (kaldet formelle parametre). Parentesen medtages kun hvis der optræder sådanne formelle parametre. De formelle parametre erstattes af de aktuelle parametre på det tidspunkt hvor proceduren faktisk udføres.

Lad os v.h.a. et eksempel prøve at belyse begreberne procedure, formelle og aktuelle parametre. Hvis vi kigger på programmet FARVER på side 7.40 kan vi se at vi 5 forskellige steder i programmet udfører en regneoperation af typen T:=T+1. Denne beregning kan vi foretage i en procedure som vi f.eks. kan kalde ADDONE, idet proceduren opbygges på følgende måde:

```
PROC ADDONE (REF T)
T:=T+1
ENDPROC ADDONE
```

Hvis vi laver programmet FARVER med denne procedure kommer det til at se ud som følger:

CALL BY VALUE
CALL BY NAME

hvis REF mangler bliver vardien ikke lagt tilbage til den aktuelle
kan også hedder vardioverføring
elle adresseføring

```
0010 // PROGRAM FARVER.
0012 PROC ADDONE(REF T)
0014  T:=T+1
0016 ENDPROC ADDONE
0018 //
0020 DIM FARVE$ OF 10, TÆLLER(5)
0030 MAT TÆLLER:=0
0040 INPUT "Indtast farve:": FARVE$
0050 IF FARVE$<>"STOP" THEN
0060  CASE FARVE$ OF
0070  WHEN "RØD"
0080   EXEC ADDONE(TÆLLER(1))
0090  WHEN "GUL"
0100   EXEC ADDONE(TÆLLER(2))
0110  WHEN "GRØN"
0120   EXEC ADDONE(TÆLLER(3))
0130  WHEN "SORT", "BLÅ"
0140   EXEC ADDONE(TÆLLER(4))
0150  OTHERWISE
0160   EXEC ADDONE(TÆLLER(5))
0170  ENDCASE
0180  GOTO 0040
0190 ELSE
0200  PRINT "RØDE ";TAB(14);TÆLLER(1);" stk."
0210  PRINT "GULE ";TAB(14);TÆLLER(2);" stk."
0220  PRINT "GRØNNE ";TAB(14);TÆLLER(3);" stk."
0230  PRINT "SORTE + BLÅ ";TAB(14);TÆLLER(4);" stk."
0240  PRINT "ANDRE FARVER ";TAB(14);TÆLLER(5);" stk."
0250 ENDIF
0260 END
```

PROCEDUREN

variabel T
– den formelle eller den forelsbøje parameter

den aktuelle parameter
– vordi lases over i PROC – beregn og lases tilbage

Resultat af programkørsel:

```
Indtast farve: SORT
Indtast farve: BLÅ
Indtast farve: TYRKIS
Indtast farve: RØD
Indtast farve: BLÅ
Indtast farve: VIOLET
Indtast farve: GRØN
Indtast farve: GRØN
Indtast farve: LILLA
Indtast farve: RØD
Indtast farve: STOP
RØDE          2  stk.
GULE          0  stk.
GRØNNE        2  stk.
SORTE + BLÅ   3  stk.
ANDRE FARVER  3  stk.
```

Som det kan ses i programmet er alle sætninger af typen

```
TÆLLER (I):=TÆLLER(I)+1
```

erstattet af

```
EXEC ADDONE(TÆLLER(I))
```

hvor ordet EXEC (eng: execute = udfør), aktiverer den procedure hvis navn er angivet bagefter (her ADDONE).

At en parameter er formel betyder at navnet på den kun er symbolsk, hvilket så igen betyder at denne formelle parameter først får et aktuelt navn når proceduren kaldes med EXEC, og får da navnet på den variabel der er angivet i EXEC-kaldet (aktuel parameter). D.v.s. at en formel parameter kan antage alle de variabelnavne det skal være (et for hvert kald af proceduren).

I det angivne eksempel er T en formel parammeter og TÆLLER(1), TÆLLER(2),.....TÆLLER(5) er alle aktuelle parametre. F.eks. ved kaldet

```
EXEC ADDONE (TÆLLER(3))
```

vil T i proceduren antage navnet TÆLLER(3) således, at det vil i virkeligheden tæller op ved ordren

```
T:=T+1
```

er TÆLLER(3):=TÆLLER(3)+1

Lad os prøve at stille variablene op i et skema og se hvilke værdier de antager for en programkørsel med de samme inddata som vist på side 7.40:

Linie	Farve $	Tæller					T
		1	2	3	4	5	
30		0	0	0	0	0	
40	SORT						
140							0
14					1		1
40	BLÅ						
140							1
14					2		2
40	TURKIS						
160							0
14						1	1
40	RØD						
80							0
14		1					1
40	BLÅ						
130							2
14					3		3
40	VIOLET						
160							1
14						2	2
40	GRØN						
120							0
14				1			1
40	GRØN						
120							1
14				2			2
40	LILLA						
160							2
14						3	3
40	RØD						
80							1
14		2					2
40	STOP						

Som vi kan se antager T den variabels værdi, som proceduren bliver kaldt med, og ændringer i T vil overføres til den selvsamme variabel. Denne kalde-form kaldes for adresseoverføring, idet alle variable i et program jo har en adresse i datamatens indre lager, og det er på denne adresse, ændringerne foregår ved procedurekaldet. Som alternativ til denne kaldeform findes værdioverføring, hvor man blot overfører en værdi til proceduren, og altså ikke kan overføre ændring af en variabels værdi i proceduren til uden for proceduren. Denne forskel i kaldemåde gøres der opmærksom på i proceduren, ved foran variabelnavnet i parameterlisten at angive ordet REF (eng: Reference=henvisning) hvis en variabel skal adresseoverføres ellers udelades dette ord (ved værdioverføring).

Der kan sagtens være mere end én parameter i en procedures parameterliste. Hvis vi f.eks. skulle lave en procedure til beregning af et individs normalvægt ud fra højde og køn, skulle normalvægt adresseoverføres og højde og køn skulle værdioverføres, da normalvægt jo er en variabel hvis indhold bestemmes af beregninger i proceduren, og højde og køn er konstanter, der bruges i beregningerne.

Laver vi dette eksempel lidt mere skematisk kunne proceduren se ud som følger:

```
10   PROC VÆGT (REF NORMVÆGT, HØJDE, KØN$)
20
30    Beregninger
 .
 .
 .
200 ENDPROC VÆGT
```

og et kald af proceduren som følger

```
300 EXEC VÆGT (NV,181,"mand")
```

hvor højden altså er 181 og køn="mand". Efter procedurekaldet vil variablen NV indeholde den beregnede normalvægt.

Prøv at lave en procedure, der ud fra en kugles radius R beregner kuglens rumfang A, og indsæt proceduren i programmet kugle-II på side 7.23.

7.1.4.8. EKSEMPEL PÅ STØRRE PROGRAM.

```
0010 // ****************************
0020 // *  Kurvetegnings program   *
0030 // ****************************
0040 //
0050 // L e o n  B e n n i c k e  04.03.82  Version 1.0
0060 //
0070 DIM VALUE$ OF 14
0080 DIM INDFIL$ OF 20, SVAR$ OF 4, VALCUT$ OF 9, PRT$ OF 9
0090 LV:=1
0100 //
0110 //
0120 PROC CUT(V, REF VALCUT$, REF LV)
0130  VALCUT$:=SPC$(9)
0140  VALUE$:=STR$(V)
0150  LV:=LEN(VALUE$)
0160  IF "E" IN VALUE$ THEN
0170   VALCUT$(6:9):=VALUE$(LV-3:LV)
0180   VALCUT$(1:5):=VALUE$(1:5)
0190   IF LV)9 THEN LV:=9
0200  ELIF LV(=9 THEN
0210   VALCUT$(10-LV:9):=VALUE$(1:LV)
0220   IF LV()9 THEN VALCUT$(1:9-LV):=SPC$(9-LV)
0230  ELSE
0240   VALCUT$(1:9):=VALUE$(1:9); LV:=9
0250  ENDIF
0260 ENDPROC CUT
0270 //
0280 // Start program...
0290 CLEAR
0300 INPUT "Inddata filnavn:)": INDFIL$
0310 OPEN FILE 1, INDFIL$, READ
0320 INPUT "Ønskes automatisk skalering af X og Y akser (JA/NEJ):)": SVAR$
0330 IF SVAR$(1:2)="NE" OR SVAR$(1:2)="ne" THEN
0340  INPUT "Min. X-værdi:)": MINX
0350  INPUT "Max. X-værdi:)": MAXX
0360  INPUT "Min. Y-værdi:)": MINY
0370  INPUT "Max. Y-værdi:)": MAXY
0380 ELSE
0390  IF NOT EOF(1) THEN
0400   INPUT FILE 1: X, Y
0410   MAXX:=X; MINX:=X; MAXY:=Y; MINY:=Y
0420  ENDIF
0430  WHILE NOT EOF(1) DO
0440   INPUT FILE 1: X, Y
0450   IF X)MAXX THEN MAXX:=X
0460   IF X(MINX THEN MINX:=X
0470   IF Y)MAXY THEN MAXY:=Y
0480   IF Y(MINY THEN MINY:=Y
0490  ENDWHILE
0500  CLOSE FILE 1
0510  OPEN FILE 1, INDFIL$, READ
0520 ENDIF
0530 CLEAR
0540 DELTAX:=(MAXX-MINX)/70
0550 DELTAY:=(MAXY-MINY)/20
0560 CLEAR
0570 CURSOR 10, 1
0580 PRINT CHR$(16);
0590 CURSOR 80, 1
0600 PRINT CHR$(17)
```

```
0610 CURSOR 80, 20
0620 PRINT CHR$(18)
0630 CURSOR 10, 20
0640 PRINT CHR$(15)
0650 EXEC CUT(MINX,VALCUT$,LV)
0660 XC:=10-TRUNC(LV/2)
0670 PRT$:=VALCUT$(10-LV:9)
0680 CURSOR XC, 21
0690 PRINT PRT$
0700 FOR J:=1 TO 20 STEP 19 DO
0710  FOR I:=10 TO 78 DO
0720   CURSOR I+1, J
0730   IF ((I+1) MOD 5=0) AND (J=20) THEN
0740    PRINT CHR$(22)
0750    IF (I+1) MOD 10=0 THEN
0760     V:=MINX+(I-10)*DELTAX
0770     EXEC CUT(V,VALCUT$,LV)
0780     XC:=I+1-TRUNC(LV/2)
0790     PRT$:=VALCUT$(10-LV:9)
0800     CURSOR XC, 21
0810     PRINT PRT$
0820    ENDIF
0830   ELSE
0840    PRINT CHR$(5)
0850   ENDIF
0860  NEXT I
0870 NEXT J
0880 FOR J:=19 TO 2 STEP -1 DO
0890  CURSOR 10, J
0900  IF (J MOD 2=0) THEN
0910   PRINT CHR$(6)
0920   Y:=(20-J)*DELTAY+MINY
0930   EXEC CUT(Y,VALCUT$,LV)
0940   CURSOR 1, J
0950   PRINT VALCUT$
0960  ELSE
0970   PRINT CHR$(4)
0980  ENDIF
0990  CURSOR 80, J
1000  PRINT CHR$(4)
1010 NEXT J
1020 EXEC CUT(MINY,VALCUT$,LV)
1030 CURSOR 1, 20
1040 PRINT VALCUT$
1050 WHILE NOT EOF(1) DO
1060  INPUT FILE 1: X, Y
1070  IF (X<=MAXX) AND (X>=MINX) AND (Y<=MAXY) AND (Y>=MINY) THEN
1080   XPOS:=TRUNC((X-MINX)/DELTAX)+10
1090   IF XPOS>80 THEN XPOS:=80
1100   IF XPOS<10 THEN XPOS:=10
1110   YPOS:=20-TRUNC((Y-MINY)/DELTAY)
1120   IF YPOS<=0 THEN YPOS:=1
1130   IF YPOS>20 THEN YPOS:=20
1140   CURSOR XPOS, YPOS
1150   PRINT "*"
1160  ENDIF
1170 ENDWHILE
1180 CLOSE FILE 1
1190 CURSOR 1, 22
1200 END
```

7.2. AFPRØVNING AF PROGRAMMER.

Som nævnt i starten af afsnit 7, er det en god ide at teste sine programmer for, om de nu også udfører hvad man forventer af dem. Det er jo ikke nok at programmerne er syntax-fejlfri (grammatisk korrekte), de skal også fungere. Afprøvning af programmer kræver disciplin, da man efterhånden traditionelt bliver mere og mere sløset med sin afprøvning efterhånden som man bliver bedre til at programmere. Sagt med andre ord, stoler man mere på sig selv, hvilket kan få katastrofale følger for brugeren af programmet, der måske bliver ført bag lyset, p.g.a. en endnu ikke opdaget fejl. En anden ting man skal være opmærksom på, er at de mennesker man laver programmer til, måske ikke kan forstå at det skal tage så lang tid at fremstille et program, hvorfor man kan komme ud for at der bliver skyndet på en, med bortfald af programtesten til følge. I en sådan situation, må man prøve at få vedkommende til at forstå vigtigheden af at få afprøvet programmet. Det er specielt ved store programmer (mange siders kode) at afprøvningen kommer til sin ret, da store programmer kan være vanskelige at overskue.

Hvordan kan man så afprøve programmer?

Ja, der eksisterer forskellige metoder hertil.

7.2.1. SKRIVEBORDSTEST.

Den første metode vi vil beskrive, er den såkaldte skrivebordstest, hvor man simpelthen sætter sig ved sit skrivebord eller hvad man nu har, og leger datamat. D.v.s. at man nedskriver navnene på samtlige variable i programmet, (i en tabel), starter forfra i programmet og udfører alle ordrer i den rækkefølge der angives i programmet, og angiver eventuelle udskrifter herfra på papiret. Alle de værdier de forskellige variable gennemløber i programforløbet, nedskrives under de pågældende variabelnavne. Når man så er færdig og forhåbentlig har fulgt programmet korrekt og nedskrevet og beregnet de forskellige variable korrekt, kan man se om indholdet af de variable og udskrifterne svarer til forventningerne. Man må altså vælge testdata til programmet der dels er nemt at regne manuelt på, og dels bevirker at alle dele af programmet gennemløbes mindst én gang.

Som eksempel på en skrivebordstest vil vi teste et program der beregner om et personnummer er gyldigt eller ej.

Inden vi kigger på programmet, må vi lige gøre os klart hvordan man kontrollerer et personnummer:

Et CPR-nummer på 10 cifre giver fødselsdato (ciffer 1 og 2), fødselsmåned (ciffer 3 og 4), og fødselsår (ciffer 5 og 6). Ciffer 7,8 og 9 er et løbenummer, og ciffer 10 er et såkaldt kontrolciffer, der desuden angiver kønnet.

Kontrolcifferet er ikke et tilfældigt tal, men afledt af de første 9 cifre. Det sker på følgende måde:

CPR-nummer	xx xx xx xx x
Vægttal	43 27 65 43 2

1. Multiplicer hvert af de første 9 cifre i CPR-nummeret med det tal (vægttal), der står opført lige nedenunder. Disse vægttal gælder for alle CPR-numre.
2. Læg resultaterne (produkterne) sammen.
3. Divider den fremkomne sum med 11 og find resten.
4. Træk denne rest fra 11. Resultatet er kontrolcifret. D: Sidste ciffer

   Tag f.eks. CPR-nummer 07 04 41 0123

   Det giver summen 0+21+0+28+24+5+0+3+4 = 85.

   Summen af 85 divideres med 11. Det gi'r en 7-gang med 8 til rest.

   Resten 8 trækkes fra 11. Resultatet er 3. Netop det sidste ciffer.

   Nr. 12 født den 7. april 1941 får altså føjet et 3-tal til som sidste ciffer.

Hvis kontrolcifret beregnes til at være 11 sættes det til 0. Hvis kontrolcifret beregnes til at være 10 kan det ikke være et CPR-nummer.

På næste side er vist et program der kan kontrollere CPR-numre, hvilket vi nu vil kontrollere ved en skrivebordstest. For at få afprøvet alle sætningerne, må vi altså sørge for at få CPR-numre med et beregnet kontrolciffer på 10,11 og mindre end eller lig med 9. De viste CPR-numre i datasætningerne linie 380, 390 og 400 opfylder disse krav, idet CPR-nummer

1.- Giver kontrolciffer = 10 og altså må forkastes.

2.- Giver kontrolciffer = 11 og altså må godkendes da det faktiske kontrolciffer jo er 0.

3.- Giver kontrolciffer = 7 der jo passer med det faktiske kontrolciffer, hvorfor CPR-nummeret må godkendes.

```
0010 // MODULUS 11 KONTROL AF CPR.-NUMRE
0020 //
0030 DIM A#(10)
0040 DIM F#(10)
0050 //
0060 VÆGT#:=4
0070 FOR POSITION#:=1 TO 9 DO
0080  F#(POSITION#):=VÆGT#
0090  VÆGT#:=VÆGT#-1
0100  IF VÆGT#)1 THEN GOTO 0120
0110  VÆGT#:=7
0120 NEXT POSITION#
0130 //
0140 FOR POSITION#:=1 TO 10 DO
0150  READ A#(POSITION#)
0160  IF A#(POSITION#)=-1 THEN GOTO 0400
0170  PRINT A#(POSITION#);
0180 NEXT POSITION#
0190 //
0200 SUM#:=0
0210 FOR POSITION#:=1 TO 9 DO
0220  SUM#:=SUM#+A#(POSITION#)*F#(POSITION#)
0230 NEXT POSITION#
0240 //
0250 KONTROL#:=11-SUM# MOD 11
0260 IF KONTROL#)9 THEN GOTO 0340
0270 IF KONTROL#<>A#(10) THEN GOTO 0310
0280 //
0290 PRINT " KAN VÆRE CPR.-NUMMER."
0300 GOTO 0140
0310 PRINT " KAN IKKE VÆRE CPR.-NUMMER"
0320 GOTO 0140
0330 //
0340 IF KONTROL#=10 THEN GOTO 0310
0350 KONTROL#:=0
0360 GOTO 0270
0370 //
0380 DATA 0, 1, 0, 1, 8, 0, 3, 5, 2, 1
0390 DATA 0, 1, 0, 1, 8, 0, 2, 3, 1, 0
0400 DATA 0, 1, 0, 1, 8, 0, 2, 3, 3, 7, -1
0410 END
```

Fig.7.7 Program til kontrol af CPR-numre.

A #										F #										VÆGT #	POSITION #	KONTROL #	SUM #
1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10				
										4										4	1		
											3									3	2		
												2								2	3		
													7							1			
														6						7	4		
															5					6	5		
																4				5	6		
																	3			4	7		
																		2		3	8		
																				2	9		
																				1			
																				7	10		
0																					1		
	1																				2		
		0																			3		
			1																		4		
				8																	5		
					0																6		
						3															7		
							5														8		
								2													9		
									1												10		
																					11		0
																					1		0
																					2		3
																					3		3
																					4		10
																					5		58
																					6		58
																					7		70
																					8		85
																					9		89
																					10	10	
0																					1		
	1																				2		
		0																			3		
			1																		4		
				8																	5		
					0																6		
						2															7		
							3														8		
								1													9		
									0												10		
																					11		0
																					1		0
																					2		3
																					3		3
																					4		10
																					5		58
																					6		58
																					7		66
																					8		75
																					9		77
																					10	11	

A #										F #										VÆGT #	POSITION #	KONTROL #	SUM #
1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10				
0																					1		
	1																				2		
		0																			3		
			1																		4		
				8																	5		
					0																6		
						2															7		
							3														8		
								3													9		
									7												10		
																					11		0
																					1		0
																					2		3
																					3		3
																					4		10
																					5		58
																					6		58
																					7		66
																					8		75
																					9		81
																					10	7	
-1																					1		

Skærm

0 1 0 1 8 0 3 5 2 1 kan ikke være CPR.-nummer

0 1 0 1 8 0 2 3 1 0 kan være CPR.-nummer

0 1 0 1 8 0 2 3 3 7 kan være CPR.-nummer

PROGRAMMET MED FRAKOBLEDE TESTUDSKRIFTER

```
0010 // MODULUS 11 KONTROL AF CPR.-NUMRE
0020 //
0030 DIM A#(10)
0040 DIM F#(10)
0050 //
0055 //TEST--1--PRINT "VÆGT-TABEL: 1 2 3 4 5 6 7 8 9  F#(1-9)"
0056 //TEST--2--PRINT SPC$(12);
0060 VÆGT#:=4
0070 FOR POSITION#:=1 TO 9 DO
0080  F#(POSITION#):=VÆGT#
0085  //TEST--3--PRINT F#(POSITION#);
0090  VÆGT#:=VÆGT#-1
0100  IF VÆGT#)1 THEN GOTO 0120
0110  VÆGT#:=7
0120 NEXT POSITION#
0125 //TEST--4--PRINT
0130 //
0140 FOR POSITION#:=1 TO 10 DO
0150  READ A#(POSITION#)
0160  IF A#(POSITION#)=-1 THEN GOTO 0410
0170  PRINT A#(POSITION#);
0180 NEXT POSITION#
0190 //
0200 SUM#:=0
0205 //TEST--5--PRINT "SUMVARIABEL SUM#:";
0210 FOR POSITION#:=1 TO 9 DO
0220  SUM#:=SUM#+A#(POSITION#)*F#(POSITION#)
0225  //TEST--6--PRINT " ";SUM#;
0230 NEXT POSITION#
0240 //
0250 KONTROL#:=11-SUM# MOD 11
0260 IF KONTROL#)9 THEN GOTO 0340
0270 IF KONTROL#()A#(10) THEN GOTO 0310
0280 //
0285 //TEST--7--PRINT """285"" KONTROL#= ";KONTROL#
0290 PRINT " KAN VÆRE CPR.-NUMMER."
0300 GOTO 0140
0310 PRINT " KAN IKKE VÆRE CPR.-NUMMER"
0315 //TEST--8--PRINT """315"" KONTROL#= ";KONTROL#
0320 GOTO 0140
0330 //
0340 IF KONTROL#=10 THEN GOTO 0310
0345 //TEST--9--PRINT """355"" KONTROL#= ";KONTROL#
0350 KONTROL#:=0
0360 GOTO 0270
0370 //
0380 DATA 0, 1, 0, 1, 8, 0, 3, 5, 2, 1
0390 DATA 0, 1, 0, 1, 8, 0, 2, 3, 1, 0
0400 DATA 0, 1, 0, 1, 8, 0, 2, 3, 3, 7, -1
0410 END
```

```
0 1 0 1 8 0 3 5 2 1  KAN IKKE VÆRE CPR.-NUMMER
0 1 0 1 8 0 2 3 1 0  KAN VÆRE CPR.-NUMMER.
0 1 0 1 8 0 2 3 3 7  KAN VÆRE CPR.-NUMMER.
```

```
0010 // MODULUS 11 KONTROL AF CPR.-NUMRE
0020 //
0030 DIM A#(10)
0040 DIM F#(10)
0050 //
0055 PRINT "VÆGT-TABEL: 1 2 3 4 5 6 7 8 9  F#(1-9)"
0056 PRINT SPC$(12);
0060 VÆGT#:=4
0070 FOR POSITION#:=1 TO 9 DO
0080  F#(POSITION#):=VÆGT#
0085  PRINT F#(POSITION#);
0090  VÆGT#:=VÆGT#-1
0100  IF VÆGT#>1 THEN GOTO 0120
0110  VÆGT#:=7
0120 NEXT POSITION#
0125 PRINT
0130 //
0140 FOR POSITION#:=1 TO 10 DO
0150  READ A#(POSITION#)
0160  IF A#(POSITION#)=-1 THEN GOTO 0410
0170  PRINT A#(POSITION#);
0180 NEXT POSITION#
0190 //
0200 SUM#:=0
0205 PRINT "SUMVARIABEL SUM#:";
0210 FOR POSITION#:=1 TO 9 DO
0220  SUM#:=SUM#+A#(POSITION#)*F#(POSITION#)
0225  PRINT " ";SUM#;
0230 NEXT POSITION#
0240 //
0250 KONTROL#:=11-SUM# MOD 11
0260 IF KONTROL#>9 THEN GOTO 0340
0270 IF KONTROL#<>A#(10) THEN GOTO 0310
0280 //
0285 PRINT """285"" KONTROL#= ";KONTROL#
0290 PRINT " KAN VÆRE CPR.-NUMMER."
0300 GOTO 0140
0310 PRINT " KAN IKKE VÆRE CPR.-NUMMER"
0315 PRINT """315"" KONTROL#= ";KONTROL#
0320 GOTO 0140
0330 //
0340 IF KONTROL#=10 THEN GOTO 0310
0345 PRINT """355"" KONTROL#= ";KONTROL#
0350 KONTROL#:=0
0360 GOTO 0270
0370 //
0380 DATA 0, 1, 0, 1, 8, 0, 3, 5, 2, 1
0390 DATA 0, 1, 0, 1, 8, 0, 2, 3, 1, 0
0400 DATA 0, 1, 0, 1, 8, 0, 2, 3, 3, 7, -1
0410 END
```

```
VÆGT-TABEL: 1 2 3 4 5 6 7 8 9  F#(1-9)
            4 3 2 7 6 5 4 3 2
0 1 0 1 8 0 3 5 2 1 SUMVARIABEL SUM#: 0  3  3  10  58  58  70  85  89
 KAN IKKE VÆRE CPR.-NUMMER
"315" KONTROL#= 10
0 1 0 1 8 0 2 3 1 0 SUMVARIABEL SUM#: 0  3  3  10  58  58  66  75  77
"355" KONTROL#= 11
"285" KONTROL#= 0
 KAN VÆRE CPR.-NUMMER.
0 1 0 1 8 0 2 3 3 7 SUMVARIABEL SUM#: 0  3  3  10  58  58  66  75  81
"285" KONTROL#= 7
 KAN VÆRE CPR.-NUMMER.
```

Af den vedlagte skrivebordstest, ses det at programmet opførte sig som forventet.

7.2.2. INDSÆTTELSE AF SKRIVESÆTNINGER

En anden og mindre besværlig metode, er at indsætte skrivesætninger i programmet de steder, hvor en test kræver at få udskrift fra. Laver man store programmer kan det betale sig allerede ved konstruktion af programmet at indbygge test-faciliteter, der kan kobles til og fra efter behag. Apropos, er der i mange EDB-systemer faciliteter til på en nem måde at foretage disse til og frakoblinger af testudskrifter. Som et eksempel på anvendelsen af denne metode har vi i programmet til modulus-11 kontrol indført skrivesætninger til testbrug. Den første udskrift viser programmet med testudskrifter indsat men frakoblet (ved at sætte // foran), og resultatet af kørslen.

Den næste udskrift viser så programmet med testudskrifter koblet til (ved fjernelse af //), og resultatet af kørslen.

Ved sammenligning med skrivebordstesten ses overensstemmelse.

8. PROBLEMANALYSE.

Dette afsnit vil forsøge at belyse hvilke handlinger der er nødvendige til løsning af en EDB-opgave i videste forstand. Forestil dig at laboratoriechefen kommer hen til dig og siger noget i retning af: "Ja, Hansen vi ved jo at du har gået på EDB-kursus, og som sådan er den eneste her på stedet, der har mulighed for at hjælpe os. Som du jo ved købte vi for 1/2 år siden en datamat af mærket KVALIMAT-4X, idet firmaet vi købte den af lovede at fremstille nogle programmer der kunne indsamle analyseværdier og foretage statistiske beregninger på værdierne. Firmaet er underbemandet, og vil først kunne hjælpe os om ca. 3/4 år, og så har de endda ikke lovet det fast, selv om det kun ville tage ca. 1 uge for dem. Det er jo stramme tider, så konsulentbistand vil vi helst ikke nytte os af. Det jeg vil bede dig om, er at forsøge at løse vores problem så hurtigt som muligt og få lavet de der 2-3 programmer. Vi skal helst kunne bruge dem inden for en måned.

Hvad vil den stakkels Hansen nu gøre for at løse den stillede opgave? Han fornemmede måske endda "uvejr i luften" hvis ikke opgaven blev løst hurtigt!

Det er problemer af denne størrelsesorden dette afsnit vil forsøge at stille en løsningsmodel op for, så man, hvis man kommer i Hansens situation, har en rimelig chance for at løse mysteriet.

Nu må man så spørge sig selv hvad sådan en model skal gøre godt for? Det er da ikke utænkeligt at Hansen uden brug af "modellen", kunne løse opgaven på en i første omgang tilfredsstillende måde! Svaret på spørgsmålet er, at selv om det i første omgang skulle lykkes at fremstille noget brugbart, er der mange ting at tage hensyn til, som man ikke umiddelbart tænker over, og som på et senere tidspunkt kan få ubehagelige følger hvis der netop ikke er taget hensyn til disse "skjulte" ting. Der eksisterer i dag EDB-systemer det har taget år at udvikle, hvor man f.eks. har undervurderet antallet af parametre (variable) man ønskede at registrere, med en tidskrævende og dyr omprogrammering til følge.

Hvis vi vender tilbage til Hansens tilfælde, kan man skitsere problemet som vist i fig. 8.1.

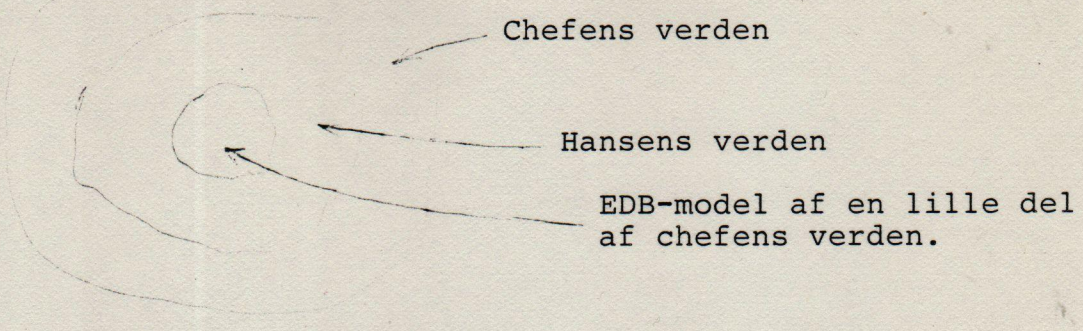


Fig. 8.1. Situationen for Hansen.

Her består chefens "verden" af hans forskelligartede meninger (farvede, ufarvede, forudindtagede, forkerte, rigtige, uovervejede, ... o.s.v.). Det samme gør sig selvfølgelig gældende for Hansen. EDB-modellen er resultatet af Hansens arbejde. EDB-modellen er udarbejdet ud fra Hansens opfattelse af den givne opgave, hvor opfattelsen er et resultat af samtaler med chefen og andre involverede personer samt af læsning af relevant litteratur. Det kræver ikke ret megen fantasi at forestille sig at der her kan opstå en masse misforståelser og bagatellisering af øjensynlig små problemer.

Nu er så det store spørgsmål hvad man dog kan gøre, for at komme den slags "uvæsen" til livs? Hertil kan følgende hjælpemidler tages i anvendelse:

1: Find ud af hvilke fakta der er knyttet til opgaven, og prøv at definere de problemer der skal løses, og opstil evt. alternative løsninger på problemerne. Forelæg fakta og løsningsmuligheder for chefen til accept og få ham til at vælge en eller flere af løsningerne til nærmere undersøgelse.

2: Undersøg de(t) udvalgte forslag nærmere med fastlæggelse af, hvad der kommer af ind- og uddata, og forsøg at vurdere hvor lang tid det vil tage at udarbejde de nødvendige programmer for de forskellige forslag. Forelæg igen løsningerne for chefen og få ham til at acceptere et af forslagene og vær sikker på at I stadig kan forstå hinanden og at den udvalgte løsning stadig kan udføre de ønskede handlinger/beregninger.

3: Udfør programmeringen og foretag en grundig afprøvning af de udviklede programmer, og afslut med at chefen afprøver programmerne med egne data, for at afgøre om de nu også udfører hvad han havde tænkt sig.

4: Undervejs er det vigtigt løbende at overveje/vurdere om det nu også kan betale sig at fortsætte udviklingen. Der kan jo opstå problemer som man ikke magter at løse, og måske derfor prøver at simplificere med uheldige virkninger til følge. Nogle vigtige årsager til at man kan blive nødt til at stoppe en programudvikling er følgende:

a: Projektet er for ambitiøst.
b: Formålet med projektet glemmes når ambitionerne sættes ned.
c: Utålmodighed og forhastede beslutninger.
d: Manglende evne til at stoppe projekter.

I de følgende afsnit vil ovenstående 4 punkter blive belyst nærmere, og der vil hvor det er formålstjenligt blive refereret til Hansens tilfælde.

8.1. LØSNINGSMULIGHEDER OG FAKTA.

Under dette punkt som vi vil kalde analysen, er det vigtigt at gøre sig fortrolig med brugerens situation. Hvem der er brugeren i Hansens situation kan kun afgøres ved at stille spørgsmål til i første omgang chefen. Formålet med analysen er at nå frem til nogle præcise EDB-krav og undgå alle misforståelser. Det er ofte sådan at EDB-manden (dig selv) kan hjælpe brugeren til at formulere bedre krav eller formulere dem sådan, at der også tages hensyn til senere udvidelse af systemet.

Analysen vil typisk bestå i en undersøgelse af hvordan man gør i øjeblikket, og hvad man forventer at kunne gøre med det

nye system. Hvad man gør i øjeblikket, vil vi kalde for brugerens nuværende situation, og forventninger til det nye system for problemformulering.

8.1.1. BRUGERENS NUVÆRENDE SITUATION.

Til beskrivelse af brugerens nuværende situation kan der f.eks. opstilles følgende punkter:

1: Formål med nuværende metode/system.
2: Forretningsgange for nuværende metode/system og de involverede personer, d.v.s. hvem udfører hvad, hvilke beregninger m.v. udføres, hvor ender resultaterne og hvem skal bruge dem, o.s.v..
3: Datavolumen, d.v.s. hvor meget fylder de datamængder der cirkulerer i systemet.
4: Indhold i alt ind- og uddata.

For at fremskaffe ovenstående oplysninger kan man bl. a. benytte sig af følgende metoder:

a: Foretage interview med de involverede personer (alle punkter).
b: Fremskaffe eksisterende blanketter og studere interne rapporter (fortrinsvis pkt. 4).
c: Forsøge at tælle datamængderne, ved f.eks. at følge en prøves vej igennem systemet og gang herefter op med antal prøver.

Vi vil endnu engang gøre opmærksom på, at man ofte får modstridende oplysninger om den samme ting gennem forskellige kilder. Det skyldes formentlig i de færreste tilfælde, at kilderne bevidst prøver at lyve, men er snarere forårsaget af misforståelser, idet et spørgsmål måske opfattes forkert, eller man giver udtryk for noget man mener at vide, hvor det faktisk forholder sig anderledes. Hvad skriftligt materiale angår, kan det se nydeligt og korrekt ud, men er måske i virkeligheden forældet. Det man kan lære heraf, er at det er vigtigt at kontrollere alle oplysninger gennem flere forskellige kilder.

8.1.2. PROBLEMFORMULERING.

Problemformuleringen der med udgangspunkt i bl.a. brugerens nuværende situation, skal danne grundlag for udarbejdelse af løsningsmuligheder, kan f.eks. omfatte følgende punkter:

1: Undersøgelse af problemer i det eksisterende system.
2: Formål med det nye system.
3: Ønsker om udvidelser på længere sigt.
4: Er der sammenhæng/hensyn til andre systemer.
5: Opstilling af løsningsmuligheder (meget kort).

Udarbejdelse af en problemformulering kan også ske ved interview, men en bedre teknik er egentlig gruppearbejde, hvor EDB-folk (dig selv) og brugere (chefen f.eks.) arbejder tæt sammen.

8.1.3. EKSEMPEL PÅ PROBLEMANALYSE.

Som eksempel på en problemanalyse vil vi her vende os mod det tidligere i dette afsnit angivne tilfælde med Hansen. Det er ikke meningen, at vi her vil foretage nogen gennemgribende analyse, idet en sådan hurtigt svulmer op i antal sider, men blot give nogle tænkte eksempler på analyseresultater.

Analysen starter med, at Hansen interviewer chefen, og under samtalen får udleveret blanketter og rapporter (eller henvisninger hertil) efterhånden som deres eksistens afsløres.

Nogle facts fra interviewet kan nu afsløres:

I øjeblikket forholder det sig sådan, at vejning af rotter samt nedskrivning af resultater og beregninger udføres manuelt idet de nedskrevne resultater leveres videre til statistiske beregninger i en anden afdeling. Det er denne proces chefen ønsker automatiseret, idet der er tale om mange rotter dagligt, og hver rotte skal vejes 3 gange. Det viser sig, at laboratoriet råder over en vægt med en analog udgang (d.v.s. der leveres en spænding U, der er proportional med vægtens visning M, $U=k \cdot M$). Hvad angår datamaten, er der ingen der ved hvordan man bruger den til vægtregistrering. Den helt konkrete opgave vil være noget i retning af:

Find ud af hvordan datamaten kan bruges til vægtregistrering, og samtidig afgøre om en rotte endnu er for lille til at indgå i statistikken. Herudover ønsker man en udskrift på linieskriveren af de forskellige registreringer samt resultatet af de statistiske beregninger. Der kan blive tale om at opbevare registreringerne i mindst en uge før de bliver slettet. På længere sigt ville det jo være smart om man kunne anvende statistikprogrammerne på andre forsøg samt koble andre apparater på datamaten.

Dette var resultatet af de indledende manøvrer. Det gælder nu om at få afklaret de forskellige problemer som stadig eksisterer (f.eks. hvordan kobler man datamaten op til vægtregistrering).

Hansen foretager yderligere undersøgelser, og finder ud af, at der i firmaet eksisterer en manual over datamaten, hvori der på engelsk vist nok står noget om registrering af spændingen v.h.a. et A/D-board, hvad det så end er for noget. Hansen finder desuden ud af, at der mangler en enhed til lagring af data over længere tid (f. eks. en floppy-disk). Hvad angår statistikker, er der i første omgang tale om middelværdi, spredning og varians, og senere om histogramtegning og signifikanstest, oplyses det fra chefens side.

Den stakkels Hansen er nu bragt i en tilstand der minder om en juleaften uden flæskesteg. Hvad skal han dog gøre? Ja, Hansens analyse er nu nået så vidt, at han har konstateret hvad der mangler, og hvad han ikke kan finde ud af, så der er kun en ting at gøre, nemlig at kontakte chefen og forelægge ham fakta, og løsnings-muligheder.

Fakta er at Hansen ikke kan klare problemet alene, og af løsningsmuligheder kan der opstilles følgende:

1: Hele problemet overlades til en konsulent, hvilket for bistanden alene vil koste 320 kr. i timen, hvortil kommer anskaffelser af f.eks. en floppy-disk til ca. 15.000 kr.
2: Hansen sendes på kursus hos firmaet der har leveret datamaten (1 uge) og prøver derpå at løse problemet. Desuden anskaffes en floppy-disk til ca. 15.000 kr.
3: Projektet opgives, og der fortsættes med manuel registrering.

Der eksisterer selvfølgelig andre løsninger, hvilket overlades til læseren at finde ud af. Af ovenstående 3 tilfælde vil der i nummer to kunne ydes hjælp fra laboratoriets statistikafdeling til forståelse af de statistiske beregninger. For at give chefen mulighed for at vurdere hvilke(n) af de 3 muligheder han vil have undersøgt nærmere, kan man forsøge at sætte tal på ting der berøres af løsningen, som vist i figur 8.2.

	Løsningsforslag		
Emne	Konsulentbistand	Kursus	Som før
Udviklingstid	30 timer	40+80 timer	0
Hjælp fra andre	4 timer (interview)	5 timer (statistik)	0
Arbejdsløn	ca. 10.000 kr.	ca. 6.500 kr.	0
Kurser	0	3.000 kr.	0
Materiel	18.000 kr.	18.000 kr.	0
Analyser/dag	100	100	50
Pålidelighed	5	3	4
Service-pris	300 kr./timen	50 kr/timen	0

Figur 8.2. Vurderingsskema.

De i figur 8.2. viste emner, er kun forslag til områder der berøres af gennemførelsen af en af de skitserede løsninger. Skemaet giver udtryk for førstehåndsvurderinger, men vil alligevel være en stor hjælp for i det her tilfælde chefen. Læg mærke til, at der er sat tal på en svært målelig størrelse, nemlig pålidelighed, hvor 5 er en ønsket egenskab og 1 en uønsket egenskab. Tallene udtaler sig om Hansens skøn over det udviklede systems pålidelighed, idet Hansen her har vurderet konsulentens system højere end både det manuelle og sit eget.

Det er muligt at chefen nu beder om at få nogle flere emner inkluderet i skemaet til støtte for en vurdering, hvilket man så må prøve at lave. Når skemaet er færdigt er det op til chefen at

at tage stiling til, hvad der videre skal ske.

8.2. SYSTEM FORSLAG.

Når brugeren d.v.s. chefen har udvalgt en eller flere af de fremlagte løsninger, er det så op til EDB-manden (dig selv) at udarbejde detaljerede løsninger af de udvalgte forslag. Hvis løsning 1 (konsulent) vælges, kan hele det videre forløb overlades til en konsulent. Vælges løsning 3 (som før) forbliver alt ved det gamle og sagen afsluttes her. Det sidste af de stillede forslag kræver, at Hansen i første omgang kommer på kursus, og lærer at udnytte den erhvervede datamat KVALIMAT-4X til spændingsmålinger. Dernæst skal Hansen, chefen og statistikafdelingen i samråd finde ud af, hvilke inddata og uddata der kan komme på tale og udarbejde formater for samme. Desuden skal datamængder og lagringsmåde fastlægges, idet dette er afgørende for, hvor stor baggrundskapacitet der skal være til rådighed. Til slut udarbejder Hansen et detaljeret systemforslag indeholdende ovenstående punkter samt oplæg til diverse programmer. Herefter forelægges de(t) detaljerede systemforslag for chefen, der så enten forkaster eller accepterer forslaget.

Når forslaget med tiden er blevet accepteret, er der så tilbage at foretage programmeringen og afprøve de udviklede programmer som skitseret i afsnit 7. Det hele afsluttes med, at brugeren (chefen) afprøver det udviklede system og forhåbentlig tilkendegiver, at han er tilfreds med resultatet.

Bemærk at der under hele udviklingsfasen har været en dialog mellem chefen og Hansen, netop til forebyggelse af misforståelser.

8.3. INTERVIEW-TEKNIK

For at få så meget at vide som muligt, og samtidig skabe en hyggelig atmosfære ved et interview, er der i litteraturen angivet mange gode råd om interview-teknik (O. Heise, kap. 16; Orilia og Stern, s. 23-27).

Nogle af de vigtigste skal kort nævnes her:

1: Hvis andre end opgavestilleren er involveret i opgaven, få da den nødvendige tilladelse dels hos op-

gavestiller og dels hos den interviewedes chef inden et interveiw.

2: Aftal tid og sted for mødet i forvejen.

3: Læs eventuelt baggrundsmateriale i forvejen og lav en liste af de emner der skal drøftes.

4: Præsenter dig ved interviewets start og forklar formålet. Varm op med lidt hyggesnak (hjembagt kage m.v.).

5: Lad være med at afbryde den interviewede.

6: Resumér hvad du har hørt for at den interviewede kan kontrollere din opfattelse.

7: Bliv ved emnet til du er sikker på at du har forstået det.

8: Undgå ledende spørgsmål d.v.s. spørgsmål der viser at du forventer et bestemt svar.

9: Når du kontrollerer andres oplysninger, så undgå at at vise det.

10: Undgå diskussioner og lad være med at lufte egne meninger.

11: Tag notater under mødet, og stop evt. den interviewede hvis det kniber med at notere væsentlige oplysninger ned.

12: Bed om skriftligt materiale, hvis det viser sig, at det findes. Hvis blanketter eller rapporter udleveres, så noter på dem, efterhånden som felterne drøftes.

13: Afslut selv mødet.

14: Skriv referat senest dagen efter, da man ellers glemmer meningen med de spredte notater. Lav samtidig en plan for et evt. supplerende interview.

De notater man tager skal for overskuelighedens skyld bestå af stikord til venstre på papiret, og korte forklarende sætninger til højre på papiret. Man vil normalt ved et 1-2 timers interview producere 3-4 A4 sider's notat.

9. FILBEHANDLING

De følgende afsnit beskriver hvad begreberne "fil" og "data" står for, samt de værktøjer der traditionelt bør findes i et datamatisk system til håndtering af "filer". Værktøjernes virkemåde og brug beskrives rimeligt detaljeret, i et forsøg på at skabe forståelse for hvad der egentlig sker når en "fil" manipuleres. Der vil gennemgående blive refereret til et bestemt datamatisk system, nemlig en comet-datamat med floppy-disk og printer tilsluttet under operativsystemet CP/M, dog uden at afsnittene derved mister deres generelle indhold, idet eksempler fra andre EDB-systemer angives.

For ikke at læseren skal miste orienteringen/overblikket over stoffet, vil afsnittene kun nogle steder uddybe det aktuelle emne, og generelt holde sig på det principielle niveau. Uddybende forklaringer må derfor henlægges til gennemgangen af stoffet.

fil datamængde
data: samling af tal
program
oversættelsesprogram fra fx COMAL-80



9.1. HVAD ER EN FIL.

Ordet "fil" vil, hvis man slår op i en teknisk ordbog fortælle, at det er et spåntagende stykke værktøj anvendt af smede og maskinarbejdere. Dette er imidlertid ikke den rigtige betydning når man snakker i EDB-vendinger; her betyder en "fil" (eng.: file = arkiv), en samling af "data" organiseret til brug ved en given opgave (f.eks. en samling af analyseresultater til brug for senere beregninger). D.v.s. at det sammensatte ord "filbehandling" betyder noget i retning af:"Manipulering af en samling data". Det skal lige bemærkes at "data" på ingen måde er begrænset til kun at omfatte resultater fra en programkørsel, men ligesåvel omfatter selve programmet, idet programmet jo selv blot er inddata til et andet program, nemlig oversætter-programmet (f.eks. BASIC eller COMAL-80) o.s.v.. Sagt på en anden måde er data: "en formaliseret repræsentation af kendsgerninger eller forestillinger på en sådan form, at den kan kommunikeres (sendes/modtages) eller omformes ved en eller anden proces".

En sådan repræsentation kan tage sigte på såvel menneskelig tolkning (f.eks. en trykt tekst) som maskinel tolkning v.h.a. apparatet (f.eks. hulkort eller elektriske signaler) eller begge dele (f.eks. optisk læsbar skrift).

9.2. HVOR GEMMES EN FIL.

En fil i EDB-mæssig forstand vil som regel altid være gemt på maskinlæsbar form på f.eks. en magnetisk plade (på disken), på et magnetbånd (spole eller kasette-bånd), en hulstrimmel, en stabel hulkort o.s.v.. Som noget ret nyt er der på markedet kommet nogle små data-opsamlingsapparater til at bære i en lomme, hvor data bliver lagret i de elektroniske kredse (RAM, DRAM m.v.), således at f.eks. en skovarbejder nemt og smertefrit en hel dag kan gå rundt og måle træer, indtaste resultaterne og, når han kommer hjem, sætter han bare apparatet til datamaten der så "støvsuger" dette for data, og lagrer disse på et af de føromtalte medier.

Herudover er det værd at bemærke, at filer som regel skal gemmes under et navn, hvilket har to umiddelbart indlysende fordele,

nemlig 1) at man ikke behøver at gå rundt og huske på hvilken plade i pladestablen, hvilken pladeside, hvilket spor, hvilken start-sektor og endelig længden på den fil man gemte for 4 mdr. siden, idet der på pladen i en tabel under det pågældende navn vil være en beskrivelse af filens beliggenhed m.v. og, 2) at man ved filens navngivning kan anvende et navn som senere hen kan give et fingerpeg om hvad filen indeholder. Hvis en fil f.eks. indeholder måleværdier for titreringer kunne man kalde filen "TITRER 1.RES" hvor "TITRER 1" f.eks. giver oplysning om at det er titreringsapparatur nr. 1 og "RES" at det er resultatet.

Et andet navn kunne være "TITRER 1. VEJ" der f.eks. kunne være en fil der indeholder en vejledning i brugen af apparaturet. Et sidste eksempel kunne være "SKAT. CML", der f.eks. står for et program skrevet i Comal ("CML") til beregning af skat. Det er altså vigtigt at man anvender et så oplysende navn som muligt til senere identifikation.

9.3. YDRE ENHEDER SOM FILER.

Hvis et EDB-system indeholder mere end et baggrundslager (fysisk eller logisk), må man nødvendigvis give alle baggrundslagrene en entydig adresse i form af et navn eller et nummer, som brugeren kan anvende ved sine fil-manipuleringer. Af bekvemmeligheds-årsager vælger man tit at betragte en ydre enhed som en fil, omend en speciel fil. Hvis et system f.eks. indeholdt to floppydiske og en linieskriver ville man f.eks. kalde de to floppydiske for A henholdsvis B og linieskriveren for LP. Skulle man således referere til en fil på floppydisk B med navnet "PH. VAL" skulle man angive det samlede navn til "B:PH.VAL", idet det angivne kolon (:) er nødvendigt for at skille navnet "PH.VAL" for angivelsen af floppy-disken "B". Ønsker man at få en fil skrevet ud på linieskriveren refereres denne med "LP:". Af andre navne på ydre enheder kunne man f.eks. tænke sig flg.:

DS: Dataskærm
CRD: Kortlæser (Card-Reader)
PTR: Hulstrimmel-læser (Paper Tape Reader)
O.s.v.

9.4. OPDELING AF FILER I FIL-TYPER.

Af brugsmæssige årsager deler man filer op i forskellige typer af filer. For blot at få en fornemmelse af, at det kan være praktisk at organisere en fil på en bestemt måde, nemlig efter hvad og hvordan den skal anvendes, nøjes vi med her at nævne to typer og forklare lidt om deres anvendelse. Lad os kalde dem for type 1 og type 2. Type 1 er en fil hvis indhold f.eks. kunne være en tekst til en rapport eller et indtastet program, der er karakteristisk derved, at de skrevne linier har uensartet længde og skal læses i rækkefølge efter hinanden. En sådan fil kalder man en "sekventiel" fil , (se fig. 9.1) idet den skrives og læses sekventielt (linie efter linie). En ting man må huske på, er at man med denne type filer ikke umiddelbart kan få skrevet en bestemt linie, men må læse filen igennem fra start til den ønskede line nås (tænk her på et magnetbånd).

Inventarliste
2 stk. bunsenbrændere
28 stk. bægerglas (2000 ml)
3 stk. Mettler-vægte

Fig. 9.1. Eksempel på sekventiel fil.

Type 2 filen er en fil hvis indhold f.eks. kunne være et register over farlige stoffer, hvor der om hvert stof var registreret dets kemiske navn, giftgruppe, samtlige salgsnavne og behandling ved forgiftning. Karakteristisk for denne fil-type er, at alle poster er lige lange, og at man ved at angive nummeret på stoffet med det samme, uden at skulle løbe filen igennem, får fat i oplysningerne om stoffet. En sådan fil kalder man for en "direkte struktureret" fil se (fig. 9.2.)

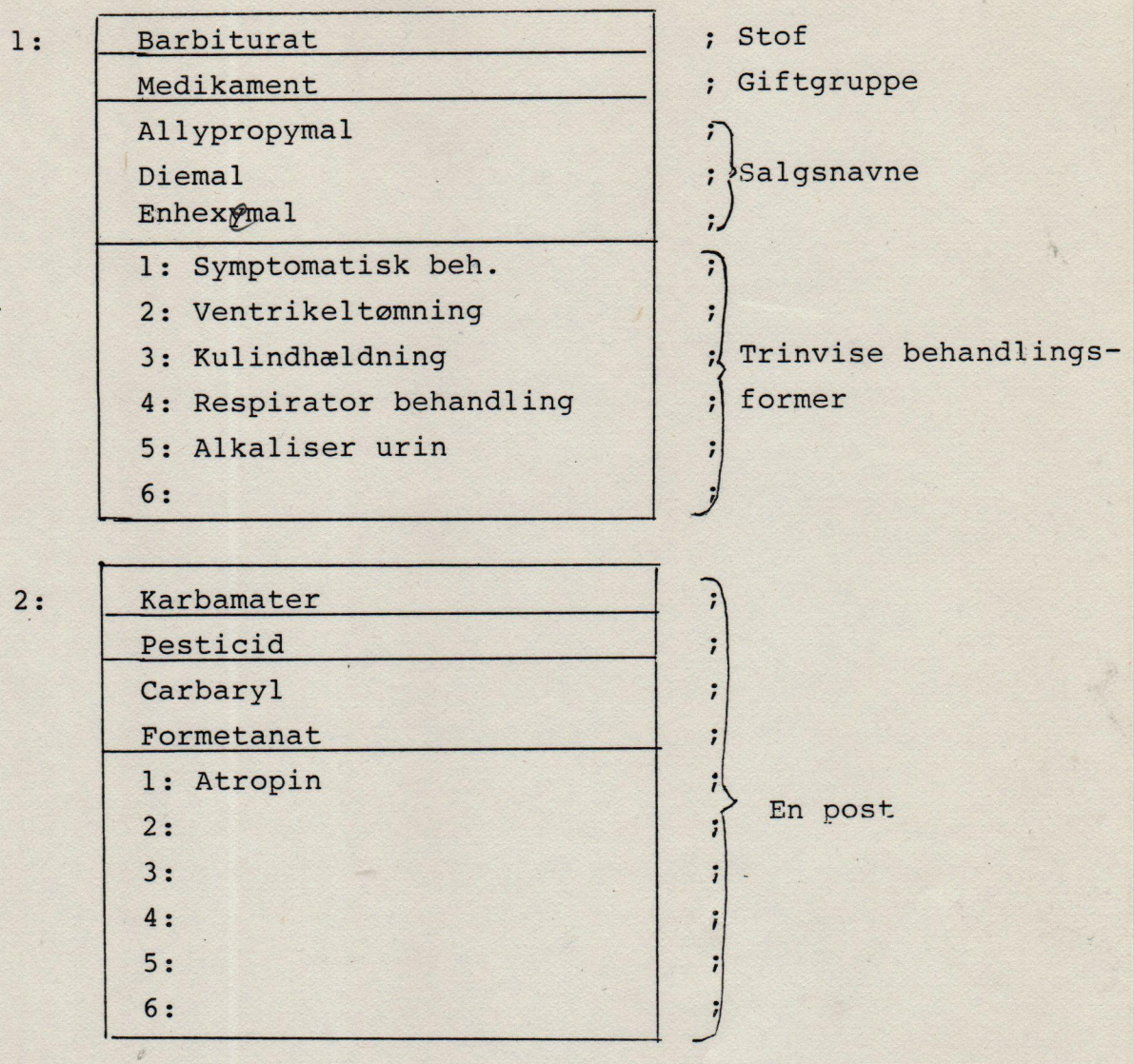


Fig. 9.2. Eksempel på direkte struktureret fil.

Denne type filer kan kun gemmes på et medium med direkte tilgang som f.eks. en diskette.

9.5. FIL-MANIPULERING

I dette afsnit vil vi fortælle noget om hvad man kan forvente at finde af filmanipulerings-faciliteter på et EDB-anlæg, der indgår i et udviklingsmiljø (som f.eks. et forsøgslaboratorium), samt beskrive faciliteternes virkemåde i korte træk.

Som noget fundamentalt for et EDB-system må man forvente at man kan :

1: Oprette filer.
2: Skrive filer.
3: Læse filer.
4: Slette filer.
5: Ændre i filer.
6: Kopiere filer mellem ydre enheder.
7: Beskytte sine filer mod læsning/skrivning.
8: Aflæse status for sine filer.
9: Aflæse filkataloget.
10: Oversætte programmer.
11: Kæde flere programmer sammen til et.
12: Køre programmer.

Formål og virkemåde af pkt. 1-12 vil blive beskrevet i de følgende afsnit, idet vi for at kunne referere til typiske filmanipulerings-ordres udseende/format, vil inddrage en Comet-datamat med floppy-disk og linieskriver tilsluttet, styret af operativsystemet CP/M. Nogle af de nævnte punkter 1-12 vil i visse sammenhænge være usynlige for brugeren; f.eks. vil det at oversætte og køre et program i forbindelse med et microprocessor-styret måleapparat være noget der for oversættelsens vedkommende, er sket en gang for alle, og hvad angår kørslen sker dette automatisk når appararatet tændes/aktiveres.

9.5.1 OPRETTELSE AF FILER.

Det at oprette en fil betyder, at man gør EDB-systemet opmærksom på at man under et eller andet navn har tænkt sig at gemme noget data på et af dets baggrundslagre. Dette betyder at EDB-systemet inden en sådan oprettelse kan godtages, må foretage en undersøgelse af følgende punkter:

1: Om navnet på filen er i orden.
2: Om der i forvejen findes en fil med samme navn.
3: Om der overhovedet er mere plads på baggrundslageret.

9.5.1.1 FILNAVNE.

Forskellige EDB-systemer har deres egne vedtagelser angående et filnavns tilladte udseende (Syntax). Der vil som regel være et bestemt format af filnavnet; f.eks. skal dette på cometen under CP/M-operativsystemet have formen

PPPPPPPP.SSS

hvor "PPPPPPPP" repræsenterer primærdelen og "SSS" sekundærdelen af navnet. Herudover vil filnavnets længde have en grænse, ligeledes forskellig fra system til system. I ovenstående eksempel skal primærdelens længde være mindre end eller lig med 8 tegn og sekundærdelens ikke længere end 3. Som en sidste ting gælder der som regel at ikke alle tegn kan indgå i et filnavn; f.eks. vil et navn som

&F16=:.<<<

ikke blive accepteret under CPM, idet tegnene <>.,;:=?*() ikke må indgå i et filnavn, hvis dette skal være entydigt (ved reference til flere filer med f.eks. samme sekundærdel kan der benyttes en *).

9.5.1.2 NAVNEKONFLIKT VED OPRETTELSE.

Hvis et filnavns udseende er blevet godkendt, vil næste opgave for edb-systemet være, at finde ud af om der i forvejen eksisterer et navn med nøjagtig samme udseende. Gør der ikke det, kan vi gå videre til sidste undersøgelse. I modsat fald vil der afhængigt af EDB-systemet ske èn af følgende ting:

a: Der bliver gjort opmærksom på at navnet eksisterer i forvejen og at man bliver nødt til at finde på et andet (f. eks. HP-1000-RTE4B).

b: Navnet bliver oprettet på ny, idet navnet blot får tilføjet et nyt versionsnummer. Ser en gammel version f.eks. således ud: X.Y ;3, vil den nye herefter komme til at hedde X.Y;4 altså en anden fil. (F.eks. PDP-11/44-RSX 11 M).

c: Den gamle fil slettes uden videre og den ny oprettes under samme navn. (F.eks. ABC 80).

d: Der bliver gjort opmærksom på at navnet eksisterer i forvejen, hvorefter man får valget imellem at slette det gamle eller finde på et nyt (f.eks. Comet-CP/M under comal-80).

9.5.1.3 PLADSMANGEL PÅ BAGGRUNDSLAGER

Når vi er nået så langt i oprettelsen af en fil, at navnet på alle måder er blevet godkendt, skal EDB-systemet som den sidste opgave undersøge om der i filkataloget er plads til yderligere et navn med angivelse af filens fremtidige placering på baggrundslageret. Det man skal bemærke er at et sådant katalog kun har en endelig afgrænset størrelse, og derfor kun kan indeholde et bestemt antal filbeskrivelser. Hvis der ikke er plads i kataloget, vil systemet udskrive en passende fejlmeddelse på terminalens skærm.

I modsat fald, hvor der altså er plads i filkataloget, vil skrivning af filen herefter påbegyndes. Tilbage er så bare at håbe at den fil man er ved at skrive, ikke er længere, end at den kan være på den resterende del af baggrundslageret. Dette behandles i næste afsnit.

9.5.2. SKRIVNING AF FILER.

Når en fil skal skrives på baggrundslageret kendes længden af filen ikke før den er skrevet. Det er jo en kedelig ting set med EDB-systemets øjne, da en påbegyndt skrivning måske viser sig ikke at kunne fuldføres (koster tid). Det er imidlertid en ting man må leve med, hvorfor det eneste fornuftige der er at gøre for edb-systemet hvis den situation opstår, er at slette det nyligt oprettede fil fra kataloget, og gøre brugeren opmærksom på at der ikke var plads til filen. Dette skal dog ikke hindre en i at forsøge at skrive en anden og kortere fil.

Når en fil er skrevet, noteres dens aktuelle længde i filkataloget under det pågældende navn.

En fils længde kan noteres på mange måder. Nogle af de mest anvendte enheder er -SEKTORER-, BLOKKE-, -KBYTE- o.s.v..

Et nærliggende spørgsmål er nu, hvordan systemet ved hvor en fil ender (f.eks. på et magnetbånd). At dette med lethed kan afgøres ligger i, at man internationalt har defineret et specielt tegn der kaldes EOF - END-OF -FILE-, der altid automatisk skrives sidst i en fil og således angiver, at her slutter filen.

9.5.2.1 HVORDAN SKRIVES EN FIL.

Når man skal skrive en fil, må man nødvendigvis starte et program op, der kan foretage den ønskede skrivning. Et sådant program vil næsten altid være at finde som ét ud af en række standardprogrammer der leveres med systemet ved købet/salget.

Skal man f.eks. under CP/M-operativsystemet på floppy-disken (disketten) oprette/skrive en fil med f.eks. forklarende tekst til et kemisk forsøg, sker dette ved at give orden

ED FORSØG.TXT

hvor ED er programnavnet og FORSØG.TXT er navnet (tilfældigt valgt) på filen. ED er en forkortelse for det engelske ord "edit", der betyder "at redigere". Efter opstart af ED-programmet (i "almindelig" tale kaldet editoren), kan man nu give ordre til editoren om f.eks. indsættelse af tekst (Insert), sletning af linier (Kill), visning af tekstlinier (Type), søgning efter en bestemt forekomst af tegn (Find),.. o.s.v..



Fig. 9.3 editor i brug

Der er til en editor som regel knyttet et væld af ordrer af mere eller mindre avanceret udformning. Brugen af en editor er ikke noget man lærer fra det ene minut til det andet, (se fig. 9.3), men er noget der skal opøves gennem længere tids anvendelse.

Editorer kan have mange forskellige navne, afhængig af hvem der har konstrueret den, hvad de kan og hvilket system de anvendes i. Af andre editorer kan nævnes

a: EDIT/1000 (HP-1000 anlæg)
b: EDI
c: EDT (PDP-11/44 anlæg)
d: KED

hvor den sidste betyder Keypad EDitor (Am: nøgle/tast-styret Editor, d.v.s. at ordre til editoren gives ved at trykke på en enkelt tast). For en god ordens skyld skal nævnes, at de fleste editorer arbejder på det man kalder ASCII-filer, altså filer hvis indhold er læsbar tekst kodet i ASCII-kode. F.eks. kan en editor ikke arbejde på oversatte programmer da disse er lagret i maskinkode.

Det der lige er blevet beskrevet, er oprettelse/skrivning af filer via terminalens tastatur. Dette er naturligvis ikke den eneste måde at skrive filer på, idet de fleste programmeringssprog via specielle ordrer, giver mulighed for at foretage en sådan oprettelse/skrivning, fra et program man selv har skrevet. I Comal-80 f.eks., oprettes en fil med navnet H2SO4.SYR med ordren

```
OPEN#3,"H2SO4.SYR", WRITE
```

hvorved man senere i programmet kan refere til denne fil ved tallet 3 på følgende måde

```
PRINT FILE 3: "analyse 1"; N
```

hvorved teksten "Analyse 1" og værdien af variablen N skrives i filen.

Ordene OPEN og PRINT vil foranledige at dels et oprettelsesprogram og dels et skriveprogram vil blive aktiveret.

9.5.3. LÆSNING AF FILER

Når en fil skal læses fra et baggrundslager, vil der ligesom under oprettelsen ske en kontrol af filnavnets lovlighed, og herefter en søgning i filkataloget efter det pågældende navn. Hvis navnet ikke eksisterer vil EDB-systemet på terminalens skærm udskrive en passende fejlmeddelse, og herefter afvente nye instrukser. Eksisterer filen derimod, vil filbeskrivelsen (pladeside, spor, sektor, m.v.) fra kataloget blive læst over i primærlageret til brug for operativsystemet. Det man intuitivt sætter i forbindelse med læsning af en fil , er en udskrift af filen i klartekst på terminalens skærm eller på linieskriveren. Dette er dog kun den halve sandhed, idet læsning af en fil ligesåvel kan betyde læsning af f.eks. analyseværdier til brug for et program man selv har skrevet, og hvor man derfor ikke umiddelbart får filens indhold til at "se".

I forbindelse med læsning af filer, vil vi her benytte lejligheden til at gøre opmærksom på forskellen mellem det man kalder

"formatterede" og "uformatterede" filer, idet nye brugere af et EDB-system erfaringsmæssigt står uforstående over for de to begreber, hvilket i bedste fald resulterer i en skærmfuld mærkværdige tegn og i værste fald at terminalen "hænges op" d.v.s. at man herefter intet kan foretage sig via tastaturet. For at forstå forskellen vil vi tage udgangspunkt i tallet 17 som vi vil skrive dels læse både formatteret og uformatteret.

Når 17 skrives formatteret gemmes hvert ciffer for sig som tegn i ASCII-kode (se fig. 9.2a).

Formatteret:	00110001	00110111	9.2.a
ASCII-værdi:	49	55	
Tegn :	"1"	"7"	

U formatteret:	00000000	00010001	9.2 b
ASCII-værdi:	0	17	
Tegn :	NUL	DC1	

Når 17 skrives uformatteret gemmes det som tallet selv altså som værdien 17 (se fig. 9.2 b). Det vil kort sagt sige, at en udskrift af den formatterede fil vil give de to tegn "1" og "7", hvorimod udskrift af den uformatterede fil vil være de to ikke synlige tegn NUL og DC1, der begge fungerer som specialtegn/styretegn i systemet. Alt i alt betyder det at man ved læsning af en uformatteret fil på forhånd skal vide hvordan data er lagret / skal tolkes.

9.5.3.1 HVORDAN LÆSES EN FIL

Ved læsning af en fil må man ligesom ved skrivning af en fil starte et program, der kan foretage den ønskede læsning. Under operativsystemet CP/M på cometen, kan en sådan løsning foretages enten med programmet ED, programmet TYPE eller programmet PIP, hvor ED er den i afsnit 9.5.2.1 omtalte editor, TYPE et specielt læseprogram (Eng.: Type = skrive på maskine) til ASCII-filer, hvis fordel er at det er nemt at bruge og huske navnet på, og PIP (Eng.: Peripheral Interchange Program = udvekslingsprogram for ydre enheder) er et avanceret filmanipulerings-program hvis brug kræver en del øvelse. Hvis vi f.eks. gerne vil have udskrevet (altså læst) en fil med navnet MINFIL.TXT på skærmen vil dette med de 3 forskellige programmer se således ud (understreget tekst skrives af brugeren):

Læseprogrammer i CP/M systemet

```
1:  A>ED minfil.TXT
    :*#A
    :*B20T

2:  A>TYPE MINFIL.TXT

3:  A>PIP LST:=MINFIL.TXT
```

Af læseprogrammer i andre systemer kan nævnes:

a: FMGR (File Manager) i HP - 1000 system
b: PIP i PDP-11/44 system
c: LIST i ABC80 system
d: LIST under COMAL-80 i cometen

Som det tydeligvis fremgår af dette afsnit vil der mange gange i et system være op til flere muligheder for at få indholdet af en fil læst.

Hvis man fra et program man selv har skrevet, ønsker at læse en fil fra et baggrundslager, vil der i de fleste programmeringssprog være mulighed herfor. For endnu en gang at referere til Comal-80, vil en læse-sekvens typisk skulle udformes på følgende måde:

```
OPEN #2,"MINFIL.TXT", READ
        :
READ FILE 2: A$,V
```

hvor OPEN-ordren vil foranledige et opslag i filkataloget samt sørge for at man senere i programmet kan referere til filen ved tallet 2, og READ ordren vil starte et læseprogram op, der vil læse og lægge data over i A$ og V.

9.5.4. SLETNING AF FILER.

Sletning af filer er en simpel operation for operativsystemet, idet det her blot gælder om at slå op i filkataloget og slette det pågældende navn, hvorefter filen logisk set ikke eksisterer mere. Noget andet er, at filens indhold først bliver slettet rigtigt, når denne overskrives af en anden fil. Det at slette en fil er en nødvendig men farlig ting at foretage. Det nødvendige går på, at et baggrundslager jo kun har en endelig begrænset størrelse, hvor-

for dette på et eller andet tidspunkt vil være fyldt op, eller man kunne forestille sig at man havde en masse "kladde-filer" liggende, som ingen skulle bruge til noget mere. Det farlige ligger i, at man måske trods stor påpasselighed kommer til at slette den forkerte fil der, når uheldet er ude, selvfølgelig kun eksisterer ét sted, nemlig der hvor man lige har slettet den, hvilket i værste fald kan betyde flere dages arbejde spildt (f.eks. ved manuel indtastning af måleværdier). For ikke at gøre nye EDB-brugere unødigt nervøse, må vi vel allerede her afsløre, at man i de fleste systemer har mulighed for at beskytte sine filer mod sletning, hvilket vil blive behandlet i et senere afsnit.

9.5.4.1. HVORDAN SLETTES EN FIL.

Som den interesserede læser måske allerede har gættet, foregår også sletning af filer ved at starte et sletteprogram (standard), med angivelse af filnavnet. Under CP/M vil sletning af en fil med f.eks. navnet "KLADDE.TXT" foregå v.h.a. programmet ERA (eng.: Erase=udviske) på følgende måde:

```
A>ERA KLADDE.TXT
```

Hvis filen eller måske hele disketten er beskyttet mod sletning vil der intet ske, bortset fra, at systemet vil gøre opmærksom på, at filen er beskyttet mod sletning.

Af slettefaciliteter i andre systemer kan nævnes følgende:

a:	PU	(eng.: Purge = rense) i HP-1000 system
b:	PIP/DE	(eng.: Delete = slette) i PDP11/44 system
c:	DELETE	i ABC80 Disc operating system
d:	DELETE	i COMAL-80 under CP/M

Ligesåvel som man fra et program kan oprette, skrive og læse en fil, kan man som regel også slette filer fra et program. Dette vil i Comal-80 have følgende typiske udseende, idet vi forestiller os at filens navn er "MINFIL.TMP" (eng.: Temporary = midlertidig) og at den er lagret på et baggrundslager DK1:

DK0 ~ A (spalte)
DK1 ~ B

linjen. DELETE "DK1:MINFIL.TMP" slette filnavn fra programmet

Grunden til at sekundærdelen af filnavnet her er kaldt.TMP er, at vi vil benytte lejligheden til at gøre opmærksom på, at betegnelsen "temporær fil" er en særdeles anvendt terminologi for en midlertid fil, altså en fil der sædvanligvis kun har en kort levetid.

9.5.5. ÆNDRING AF FILERS INDHOLD.

Ved ændring af filers indhold, afhænger måden hvorpå dette kan gøres af filens type. Hvis filen er "formatteret" (se afsnit 9.5.3.) kan man ganske enkelt foretage de ønskede ændringer v.h.a. en editor, idet man dog skal være opmærksom på, at en fil der samtidig er "Direkte struktureret" (se afsnit 9.4.) ikke altid slipper "helskindet" gennem en editor, idet editoren i de fleste tilfælde vil lave filen om til en "sekventiel" fil. Arbejder man med en direkte struktureret fil eller en "uformatteret" fil, må man for at ændre i disse, selv lave programmer der kan foretage en sådan ændring. Ligesom ved skrivning og sletning af filen, forudsættes en ændring i en eksisterende fil, at filen ikke er skrive/slette-beskyttet eller at man er i stand til at angive en eventuel beskyttelseskode. Når man ændrer en fils indhold v.h.a. en editor vil dette ske på følgende måde:

1: Filen læses fra baggrundslageret over i et arbejdsområde i primærlageret.
2: De ønskede ændringer på filen foretages på kopien i arbejdslagret.
3a: Den manipulerede fil i arbejdslageret kopieres derpå over på baggrundslageret under samme navn, eller (EXIT)
3b: Hvis man fortryder ændringerne kan man "hoppe ud" af editoren uden at få kopien skrevet over på baggrundslageret.

9.5.5.1. HVORDAN ÆNDRES EN FILS INDHOLD.

Ved ændring af en sekventiel formatteret fil startes editoren og de ønskede ændringer foretages v.h.a. editorens ordre-

samling. Under CP/M-operativsystemet ser dette ud som følger idet understreget tekst er det man selv indtaster:

```
A>ED RAPPORT.AP1
:*#A            ; filen -> primærlager
:*              Forskellige editerings
                ordrer
 :
 :
:*E Exit        ; ny fil --> baggrundslager
A>
```

I ovenstående eksempel kunne det være en rapport over et apparat nr. 1 man ønskede at ændre i.

Af editorer i andre systemer kan der henvises til afsnit 9.5.2.1.

Hvis man arbejder på direkte strukturerede filer må man gøre sig helt klart hvilket indhold der svarer til et bestemt postnummer (se afsnit 9.4.), således at man skriver evt. ændringer det rigtige sted i filen.

9.5.6. KOPIERING AF FILER

Det at man ønsker at lave en kopi af en fil, kan have forskellige årsager. En af de vigtigste er, at man ønsker at have en sikkerhedskopi liggende, for at værne sig imod forskellige fejlsituationer som f.eks. kan være:

1: Mekaniske fejl der resulterer i en fysisk destruktion af magnetpladerne/båndene.
2: Elektroniske fejl der resulterer i utilsigtet skrivning på pladerne/båndene evt. med en ødelæggelse af fil-kataloget til følge.
3: Sjusk og fejltagelser ved manuelle operationer, der sletter de forkerte filer.
4: Katastrofer (brand m.v.) og utilsigtede ydre hændelser (f.eks. strømsvigt).

Fejlmulighederne er mange og varierer stærkt (pkt. 1 og 2) fra installation til installation. Så det er altså en god ide med faste mellemrum at tage en sikkerhedskopi (eng.: Backup) af filer man ikke ønsker at miste. I forbindelse med pkt. 4 skal for en god ordens skyld nævnes, at sikkerhedskopier selvfølgelig ikke skal opbevares i samme lokalitet som originalerne. I nogle EDB-systemer er indlagt automatisk backup, der f.eks. foretages hver aften efter arbejdstids ophør, og hvor kopien den følgende morgen anbringes et sikkert sted.

En anden grund til at lave kopier af filer er, at man ønsker at overdrage f.eks. nogle programmer til andre personer. Det at kopiere filer mellem forskellige installationer er tit problematisk, da forskellige systemer ofte anvender forskellige måder at lagre data på. Med andre ord hersker der for små systemers vedkommende kaotiske tilstande hvad angår standardisering af formater.

Noget anderledes ser det ud i større systemer, idet der her ofte anvendes standarder som i sin tid blev formuleret af IBM (International Buisness Machines A/S). Hvis der ikke er forenelighed (eng.: Compatibility) mellem to systemer (se fig. 9.4.) må man f.eks. via standard-grænseflader (eng.: Interface) sende data over telefonnettet (se fig. 9.5.).

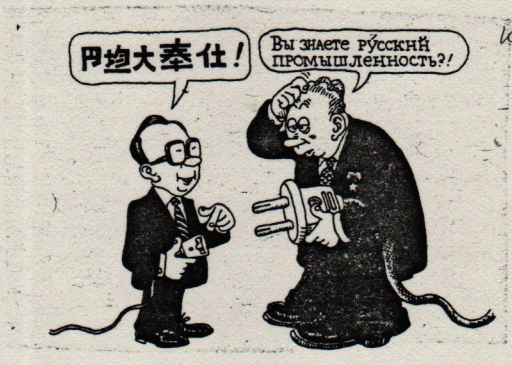


Fig. 9.4. Uforenelighed.

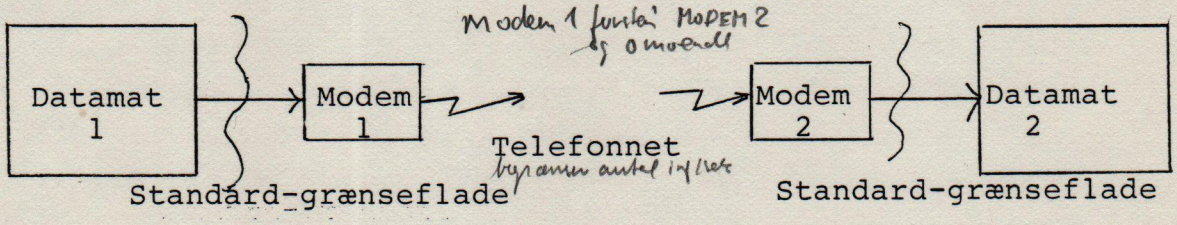


Fig. 9.5. Datatransmission over telefonnettet.

En tredie grund til at foretage kopiering, er fordi man ønsker at blande flere filer sammen til en. Denne situation er aktuel hvis man f.eks. har skrevet en rapport eller lavet et program på flere forskellige filer og nu ønsker at blande disse til en enkelt fil i en eller anden rækkefølge. Denne specielle form for kopiering kaldes at konkatenere (at sammenkæde) eller at merge (eng.: merge = smelte sammen). Hvis vi f.eks. skal konkatenere følgende 3 tekster i rækkefølgen tekst a-c-b.

CP/M system PIP

a) "Denne handling
b) at konkatenere."
c) betyder

vil resultatet blive:

"Denne handling betyder at konkatenere". Når man skal kopiere må man til forskel fra tidligere (skrivning/læsning m.v.) angive to navne nemlig et navn der fortæller hvad det er for en fil man ønsker at kopiere og et navn der fortæller hvor man har tænkt sig at gemme kopien af filen. Skal man konkatenere må man selvfølgelig angive navnene på samtlige filer man ønsker at sætte sammen, således at filerne nævnes i den rækkefølge de skal sammensættes.

Det skal for god ordens skyld endnu engang nævnes, at ydre enheder kan opfattes som en speciel filtype hvortil der kan kopieres (hvis det er en uddata enhed).

9.5.6.1. HVORDAN KOPIERES FILER.

Kopiering foretages ved opstart af et kopieringsprogram, der i ethvert velrenommeret EDB-system bør foreligge som standardprogram. Navne på sådanne kopieringsprogrammer kan være højst forskelligartede fra system til system, ja endog inden for samme system i tilfælde af tilstedeværelsen af flere kopieringsprogrammer.

Formålet med alle disse programmer er det samme, nemlig kopiering, men forskellen ligger i, hvilken slags kopiering der skal foretages, og hvilken type fil man ønsker at kopiere. Af forskel-

lige kopieringsmåder findes bl.a. kopiering af en hel pladestabel til en anden pladestabel (eller diskette til diskette) spor for spor, hvor der altså ikke foretages anden fejlcheck, end at der efter kopieringen står det samme på begge medier. Denne kopieringsmåde er hurtig, da det jo ikke er nødvendigt at lave opslag i filkataloger for at læse og skrive i disse. Den samme kopieringsform kan anvendes for kopiering mellem pladelager og magnetbånd, og magnetbånd imellem. En anden kopieringsmåde er den man som almindelig bruger af et EDB-system vil anvende mest, nemlig kopiering af filer én af gangen, hvor der foretages opslag i filkataloger m.v..

Hvad filtyper angår, vil det tit være sådan, at ét kopieringsprogram kan kopiere sekventielle filer (formatterede som uformatterede) og et andet kan kopiere direkte strukturerede filer.

Når man skal til at kopiere, og man er i tvivl om hvad man skal gøre, er det en god ide at spørge en anden som ved noget mere, da man ellers kan komme til at volde skade på eksisterende filer (ikke nødvendigvis uoprettelig skade).

En kopiering under CP/M-operativ-systemet vil typisk kunne tage sig ud som følger:

A>PIP B:=GLASREGN.CSB

hvor "B:" er navnet på en ydre enhed, nemlig Floppydisk B, og "GLASREGN.CSB" navnet på en fil på disketten A, der kunne være et program til føring af glasregnskab.

PIP er et tidligere nævnt program (se afsnit 9.5.3.1.) der i ovenstående tilfælde vil lave en kopi af filen GLASREGN.CSB på disketten B under samme navn. Vi ser her at formatet for kopieringen er

PIP <uddata fil>=<inddata fil>

der kan læses som:"uddatafilen sættes lig med inddatafilen". Under CP/M-operativsystemet kan man endvidere ved angivelse af et specialtegn gøre programmet PIP opmærksom på, at man ønsker at kopiere alle filer, der f.eks. som sekundærdel af filnavnet har teksten "CSB". Dette kan gøres ved anvendelse af specialtegnet "*" (stjerne) på følgende måde:

A> PIP A:=B:*.CSB

hvorved alle filer på B disketten med "CSB" som sekundærdel i navnet vil blive kopieret over på A disketten (hvis der selvfølgelig er plads!).

Af kopieringsprogrammer i andre systemer kan nævnes:

1) COPY (UNIVAC 1100, under EXEC 8 operativ systemet)
2) ST (HP 1000, under RTE-4B operativsystem)
3) DSC
4) PIP
5) BRU (PDP 11/47 under PSX 11M operativsystemet)
6) PAP
7) FLX
8) ERRCOPY (ABC 80, under ABC 80 DOS)

Af ovenstående programmer er DSC f.eks. et program der kopierer spor for spor fra en disc til en anden.

9.5.7. BESKYTTELSE AF FILER.

I EDB-systemer hvor mere end én bruger har adgang til det samme baggrundslager, kan det være praktisk (på det nærmeste en menneskeret) at kunne beskytte sine filer mod sletning, redigering og undertiden også mod læsning fra uvedkommende personers side. I mange systemer findes denne beskyttelses-facilitet, men desværre ikke i alle.

Beskyttelsen kan foregå på mange måder, og har som oftest kun virkning over for personer der overhovedet ikke kender det aktuelle system, forstået på den måde, at personer med et indgående kendskab til systemet kan være i stand til at aflæse forskellige sikkerhedskoder. Beskyttelsens effektivitet afhænger selvfølgelig i høj grad af hvad det er for et system og hvilke sikkerhedsstrategier der anvendes (f.eks. kryptografering).

Nogle simple metoder til beskyttelse er f.eks., at man opdeler det fælles baggrundslager i mindre "logiske" bidder, som så kun én person (eller gruppe) har adgang til, og hvor adgangstegnet er én eller anden af personen (gruppen) fastlagt kode. Denne sikkerhedskode vil så være lagret et eller andet sted på disken. En anden metode er, at alle har adgang til det samme lagerområde, hvor man så til gengæld ved oprettelsen af filerne angiver en eller anden sikkerhedskode pr. fil, som vil være gemt i filkataloget, hvor

hvor man så kun kan slette, redigere, læse hvis man i filnavnet angiver beskyttelseskoden. Kombinationen af disse to metoder kan også ses.

Udover at kunne beskytte enkelte filer og områder på en disk, kan man på nogle medier beskytte disse mod overskrivning. På disketter kan man påklæbe/aftage (systemafhængigt hvad man skal) en klæbemærkat, der dækker /afdækker et hak i pappet omkring pladen. På større pladelagre sidder der som regel en knap hvormed man ved indtrykning kan beskytte hele pladelageret mod skrivning. På store 7-9 spors magnetbånd sidder der bagpå en skrivering af blødt plastic, der ved aftagning sikrer at man kun kan læse fra magnetbåndet. Til sidst vil vi nævne kasettebåndet der findes i et utal af versioner, hvor nogle af dem (de dyre af dem selvfølgelig) kan skrivebeskyttes v.h.a. en lille forskydelig plasticlås, og andre ved påklæbning/aftagning af en mærkat.

9.5.7.1. HVORDAN BESKYTTES FILER.

Beskyttelse af filer kan foregå på forskellige måder, hvoraf vi kun vil nævne to.

Den ene foregår ved opstart af et program, hvor man så specificerer hvilken fil der skal beskyttes, og graden af beskyttelse (slette, redigere, læse). Den anden metode fungerer ved at man ved filens navngivning angiver en bestemt kode. Under CP/M-operativsystemet kan en fil få status som "kun læses" (eng.: Read Only) ved opstart af programmet "STAT" på følgende måde:

```
A> STAT B: GLASREGN.CSB $R/O   Read Only
```

hvor "STAT" er et status-program med mange anvendelsesmuligheder, her bl.a. filbeskyttelse. "$"-tegnet fortæller STAT-programmet at der nu kommer en angivelse af den pågældende fils nye status, her til Read-Only. Filbeskyttelse i andre systemer kan se ud som følger:

```
1: PIP MINFIL.TXT  PR:126983 (PPP11-/44)
2: ST,5,FETU:-1308           (HP-1000)
3: RUN PROTECT               (ABC-80)
```

hvor filbeskyttelsen for de to første er angivet ved understregning, og for den sidste opgives når programmet "PROTECT" beder herom.

9.5.8. AFLÆSNING AF FILSTATUS.

Aflæsning af filstatus betyder at man v.h.a. et program aflæser nogle for en fil typiske parametre. Det kan f.eks. dreje sig om filens type (sekventiel/direkte), dens størrelse (udtrykt i f.eks. KByte), filens indhold (formatteret/uformatteret), beskyttelsesstatus altså om den er slette-, redigerings eller læsebeskyttet, og om filen kan forventes intakt. Sidstnævnte punkt kan blive aktuelt hvis man f.eks. foretager en kopiering og der faktisk ikke er plads til den kopierede fil. I sådan et tilfælde vil operativsystemet "øremærke" filen til "ikke intakt". Den samme situation kan opstå hvis man i sit program foretager skrivning af filer, og programmet uventet stopper p.g.a. en fejl, således at filen ikke lukkes på rette vis.

I nogle EDB-systemer kan man få udskrevet nogle for den almindelige bruger måske uinteressante parametre, nemlig hvor filen rent faktisk befinder sig på f.eks. pladelageret, altså oplysning om pladenr. , pladeside, spor-nr. og nummeret på den sektor hvor filen begynder.

9.5.8.1. HVORDAN AFLÆSES FILSTATUS.

Aflæsning af filstatus og graden af information man kan få ud af en sådan aflæsning er systemafhængigt. Det er f.eks. ikke i alle systemer at det er muligt at få informationen om filtype og filindhold, uagtet at disse oplysninger nødvendigvis må foreligge under en eller anden form til brug for operativsystemet.

Under CP/M-operativsystemet foreligger der programmet "STAT" der varetager formidlingen af al statusinformation. Med dette program kan man bl.a. aflæse en fils størrelse dels udtrykt som antal 128 byte poster (RECS) og dels som antal kbyte der er reserveret til filen (BYTS=RECS*128/1024). Denne information fremkommer på følgende måde:

```
A> STAT B:GLASREGN.CSB
```

hvor "B" som sædvanlig betyder diskette B og "GLASREGN.CSB" er den aktuelle fils navn. Udskriften fra ovenstående brug af "STAT" kunne typisk se ud som følger:

```
RECS  BYTS  EX  D:FILENAME.TYP
  85   11K      B:GLASREGN.CSB

BYTES REMAINING ON B: 33K
```

hvoraf man kan aflæse at filen GLASREGN.CSB på (D)rive B fylder 85 poster (eng.: Records) af 128 byte der omregnet til Kbyte= 11 Kbyte. Som standardudskrift gives altid oplysning om hvor meget plads der er tilbage på det pågældende medie, her 33 Kbyte tilbage på drive B.

Af statusformidlende programmer i andre systemer kan nævnes følgende:

1: PIP } (PDP-11/44 under RSX-11M)
manuel 2: DSP } display
3: DL (HP-1000 under RTE-4B)
4: LIB (ABC-80)

9.5.9. AFLÆSNING AF FILKATALOGET.

Som tidligere nævnt indeholder filkataloget navnene på samtlige filer samt disses adresser på lagringsmediet m.v.. Når man snakker om filkatalog er det hermed underforstået, at der er tale om et medie med direkte tilgang med mindre andet er specificeret, idet nogle operativsystemer f.eks. ved håndtering af magnetbåndsfiler i starten af båndet opretter et filkatalog der så angiver hvor mange filer eller hvor mange blokke ude af båndet en given fil starter.

Programmer der kan læse i filkataloget og udskrive dets indhold, kommer til deres fulde ret når man skal søge efter en bestemt fil, som man måske oprettede to måneder tidligere og hvor man kun kan huske sekundærdelen af filnavnet. Til brug i sådanne filfælde indeholder mange EDB-systemer faciliteter til søgning på dele af filnavne. Hvis man f.eks. i det givne eksempel kun kan huske at sekundærdelen er "HGL", kan man under CP/M systemet v.h.a. et program "DIR" (eng.: Directory= Adressebog) søge på følgende måde:

A> DIR *.HGL

hvorved alle filnavne med "HGL" som sekundærdel vil blive udskrevet. Man kan selvfølgelig også få en udskrift af samtlige en diskettes indhold af filer ved blot at angive "DIR" eller "DIR *.*". Som en yderligere facilitet kan man i et filnavn erstatte enkelte

tegn med et "?" der så angiver at alle tegn kan indsættes på denne plads. Hvis der f.eks. eksisterer tre filnavne

```
1: FORMALIN.VAL
2: FORMALDE.HYD
3: FORTRINS.RET
```

og programmet "DIR" startes på følgende måde:

```
A> DIR FOR??L??.*
```

vil man få udskrevet filnavnene 1 og 2, da nr. 3 falder udenfor på tegn nr. 6 der skal være et L.

Af katalogudskrivnings-programmer i andre systemer kan nævnes følgende:

1:	CAT	(i COMAL-80 under CP/M)
2:	LIB	(ABC-80)
3:	PIP/LI	(PDP-11/44)
4:	DIR	
5:	DL	(HP-1000)

9.5.10. OVERSÆTTELSE AF PROGRAMMER.

Som noget helt fundamentalt for anvendelsen af et EDB-system, skal man kunne skrive og oversætte programmer. Et program skrives i et eller andet programmeringssprog f.eks. COMAL-80 og vil udgøre en datamængde (altså en fil) der fungerer som inddata til et andet program nemlig oversætteren, der transformerer, i det her tilfælde COMAL-80 programmet til et mere primitivt sprog assembler, der igen fungerer som inddata til en anden oversætter o.s.v..

Når et program oversættes fra dets symbolske form (selve den af brugeren skrevne programtekst) vil der dannes en ny fil som beskrevet ovenfor, der udgør noget man kalder et relokerbart (eng.: Relocate=at flytte) program, altså et flytbart program. Det flytbare går på at der inden for sådan et program ikke er referencer (f.eks. goto sætninger) ud over programmets egen kode (se fig. 9.6.).

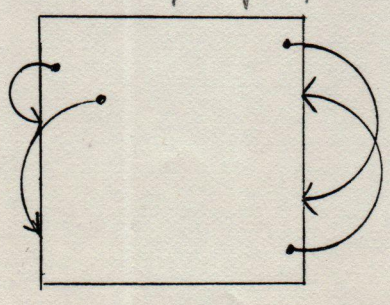


Fig. 9.6. Relokerbart program.

Dette betyder, at man kan indføje programmet som en del af et større program uden at tænke på om der sker hop til ikke definerede adresser.

I nogle EDB-systemer omtales et relokerbart program som objekt-kode. I COMAL-80 vil sekundærdelen af filnavnet være "CSB" hvor der er tale om relokerbare programmer, og "CML" hvor der er tale om et symbolsk program.

9.5.10.1. HVORDAN OVERSÆTTES PROGRAMMER.

Et program oversættes ved at starte et program "oversætteren" og angive hvilken fil der skal være inddata hertil. Indtil videre har den normale procedure været, at man først oversætter de forskellige programstumper man skal bruge, og herefter kæder dem sammen til et program. Der er altså her tale om handlinger som programmøren ser som enkeltstående aktiviteter. Denne opsplitning af aktiviteterne kan være camoufleret på mindre datamatiske systemer som f.eks. Cometen med tilhørende programmer. Her skal man for overhovedet at kunne skrive et program i Comal-80 starte programmet COMAL-80, der for hver gang man har skrevet en linie, laver syntax-analyse af denne (en del af oversættelsesproceduren).

Først på det tidspunkt hvor programmet er færdigskrevet og det startes med "RUN" ordren, vil der ske en egentlig oversættelse og sammenkædning af programstumper, hvorefter programmet udføres en ordre af gangen. I det her skitserede tilfælde virker COMAL-80 programmet altså både som Editor, oversætter og sammenkædningsprogram.

Hvis vi kigger på et andet EDB-system PDP 11/44 der kører under operativsystemet RSX-11M, vil en oversættelse komme til at se ud som følger, idet programmeringssproget denne gang er fortran:

OBJ

```
>FOR REGNSKAB.OBJ=REGNSKAB.FTN
```

Her angiver "FOR" at det er Fortranoversætteren der skal startes, og at det er filen "REGNSKAB.FTN" der er inddata hertil. Filen "REGNSKAB.OBJ" er navnet på den fil der efter oversættelsen indeholder det relokerbare program, som jo netop er resultatet af oversættelsen.

Af oversættere i andre systemer kan nævnes følgende:

1: COMPL, <inddatafil>,<uddatafil> i HP-1000 system.
"COMPL" står her for det engelske ord compiler = oversætter, og COMPL-programmet kan oversætte flere forskellige programmeringssprog.
2: PAS <uddatafil>=<inddatafil> i PDP-11/44 system.
"PAS" står her for en oversætter af PASCAL-programmer.

Der eksisterer selvfølgelig et utal af forskellige oversættere, med lige så mange forskellige navne, og af svingende kvalitet.
Det vigtigste er jo også blot at forstå at der eksisterer oversættere, og at man ikke skal være bange for at spørge en anden person om brugen eller eksistensen af oversættere.

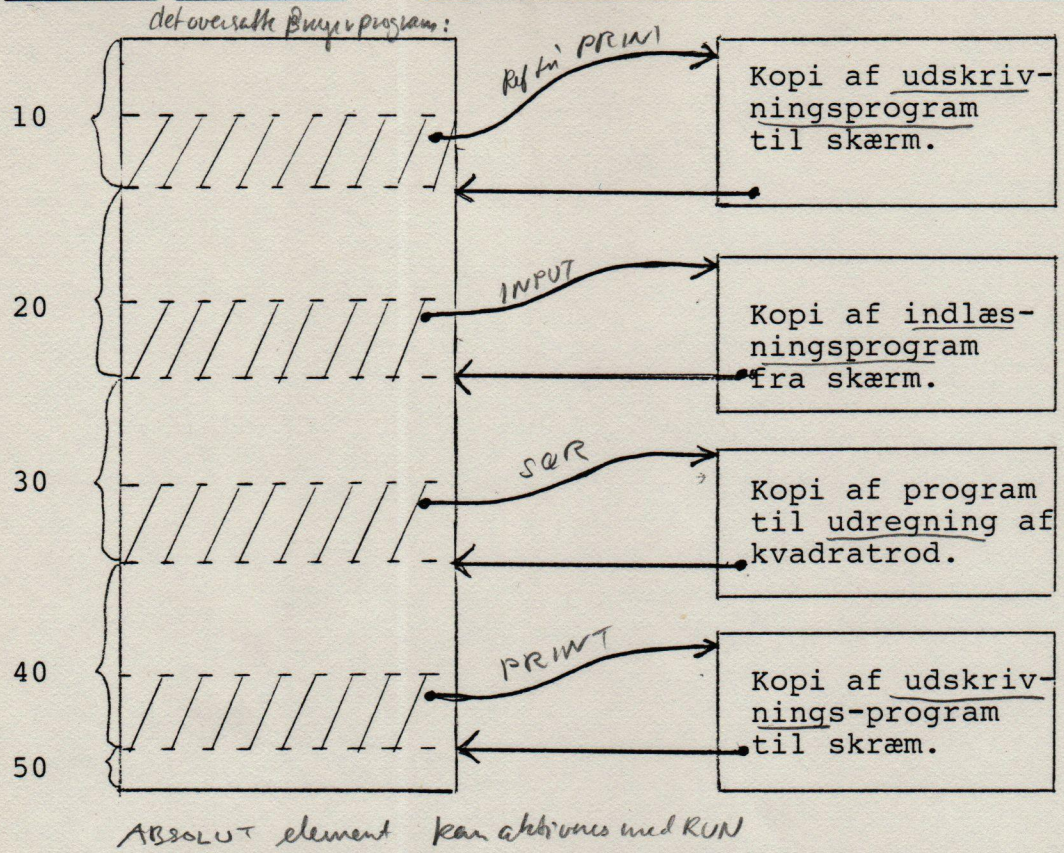
9.5.11. SAMMENKÆDNING AF FLERE PROGRAMMER.

Sammenkædning af programmer er en nødvendig ting at foretage, for at et brugerskrevet program kan blive fuldstændig "køreklart". Det forholder sig nemlig således, at alle ydre enheder bliver styret af hver deres for brugeren usynlige programdel. For hver gang der i et program er referencer til en ydre enhed (PRINT,WRITE,READ, INPUT o.s.v.) skal der aktiveres en ikke brugerskrevet programdel. De omtalte programdele er programmer, der er udviklet af f.eks. leverandøren af EDB-systemet, og er placeret på f.eks. en systemdisk/diskette. Lad os antage at vi har oversat følgende program:

```
10 PRINT "INDTAST A og B";
20 INPUT A,B
30 C:= SQR(A↑2+B↑2)
40 PRINT "hypotenusen=";C
50 END
```

Det køreklare program vil herefter f.eks. se således ud:

LINK



I dette eksempel skal venstre del af skitsen (brugsdel) opfattes som det oversatte brugerprogram, hvori der på de relevante steder er indføjet referencer til indlagte kopier af hjælpeprogrammerne. Pilene midt i skitsen illustrerer de indlagte referencer. Læg mærke til at der både er en reference fra brugerprogram til hjælpeprogram og omvendt, idet programforløbet jo efter udførslen af hjælpeprogrammet skal fortsætte i hovedprogrammet. Helt til højre i skitsen er så angivet de til afviklingen af dette program nødvendige hjælpeprogrammer. Læg her mærke til, at der ud over hjælpeprogrammer til styring af de ydre enheder, er indføjet et hjælpeprogram til udregning af kvadratroden. Dette er taget med for at vise, at hjælpeprogrammer også kan være andet end styreprogrammer til ydre enheder (det samme er f.eks. også tilfældet med diagnoseprogrammer). Der kan til mange EDB-systemer købes færdige hjælpeprogrammer ud over, hvad der som standard følger med ved købet af systemet. Hvis man er beriget med tilstrækkelig viden om det system man arbejder med, kan man selvfølgelig selv fremstille sådanne hjælpeprogrammer. I mange EDB-manualer (vejledningen) kaldes hjælpeprogrammer for biblioteksprogrammer.

Nu er det jo sådan, at det viste "køreklare" program ikke fremkommer ved blot at oversætte brugerprogrammet. Der skal endnu

en programkørsel til før dette opnås, nemlig en kørsel med et program som vi her til kalde en "programbygger", der foretager indsættelse af hjælpeprogrammerne, og danner de viste krydsreferencer i den tidligere viste skitse. Det vil med andre ord sige, at man som almindelig bruger ikke nødes til at tænke på, hvordan et program skal gøres køreklart, idet man blot skal starte "programbyggeren". Det lyder jo nemt, men kan, til dels afhængig af "programbyggerens" kvalitet, være lidt af et studium at finde ud af, hvordan en sådan skal håndteres rigtig. Endvidere er det ikke al dokumentation der er lige velskrevet, og den foreligger tit og ofte på engelsk.

En sidste ting i forbindelse med programbygning vi vil gøre opmærksom på, er at det jo på forhånd er svært at vide hvor meget et køreklart program fylder, hvorfor man kan komme ud i problemer med lagerplads i datamatens indre lager, da det jo er her programmet skal være for at kunne køres. I sådanne tilfælde må man, hvis muligheden foreligger, benytte sig af en metode man kalder segmentering, hvilket det i dette kompendium vil føre for vidt at beskrive. Dog skal det nævnes at brugen af ordren "CHAIN" i Comal-80 minder lidt om en segmentering.

Skal være overcatte programmer o: reddet ved SAVE

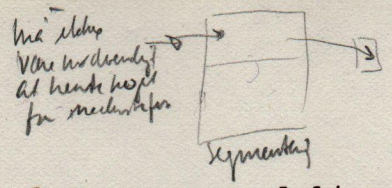
Til slut skal det for god ordens skyld nævnes, at et køreklart program mange steder kaldes for et "task", en "absolut" fil m.v..

.TSK

9.5.11.1. HVORDAN SAMMENKÆDES PROGRAMMER.

Programmer sammenkædes ved at starte et program omtalt som "programbyggeren" i forrige afsnit. Som det også fremgik her, kan det være en kompliceret sag at anvende et sådant program. Dette skyldes mange ting, hvoraf vi her skal nævne to nemlig uoverskuelig dokumentation og vældet af ordrer til en "programbygger". Men som med så meget andet lærer man af at dumme sig, og da man ikke kan ødelægge noget, er det bare med at gå igang med "bygningen".

I COMAL-80 under CP/M-operativsystemet skal man som bruger overhovedet ikke tænke på bygning af færdige programmer, idet dette foretages automatisk når man starter programmet med "RUN" ordren. Den eneste situation hvor man er nødt til at foretage en form



= OVERLAY

for programopdeling (segmentering) er, hvis programmet bliver for stort og ikke på én gang kan være i datamaten. Denne opdeling foretages med ordren "CHAIN", hvis anvendelse omtales i afsnit 7.

Af "programbyggere" i andre systemer, hvor den i forrige afsnit omtalte deling i oversættelse og bygning forefindes, kan nævnes følgende:

1: LOADR i HP-1000 systemet, hvortil opgives alle programmets bestanddele, samt specialordrer.
2: TKB i PDP-11/44 systemet (TKB=Task-Builder=Job-bygger), hvortil der ligeledes opgives alle programmets bestanddele, samt specialordren.
3: MAP i UNIVAC/EXEC 8 systemet der fungerer som pkt. 1 og 2.

9.5.12. KØRSEL AF PROGRAMMER.

Kørsel af et program betyder, at man v.h.a. en eller anden ordre til operativsystemet opgiver navnet på det program man ønsker igangsat. I enkeltbruger-systemer (altså hvor der kun er en terminal tilsluttet) vil en sådan ordre blive effektueret med det samme, og programmet vil starte et ganske kort øjeblik efter. I større systemer med flere brugere, og mange ydre enheder er man, for at fordele centralenhedens tid på en for brugerne rimelig måde, nødt til at lave nogle ordninger med kødannelser, således at alle igangsatte programmer f.eks. får tildelt 5 sec. af gangen, idet det endnu er de færreste EDB-systemer der har mere end én centralenhed. Når de 5 sec. således er gået bliver programmet sat bagest i køen af ventende programmer. Der eksisterer selvfølgelig mange forskellige strategier for hvor længe et givet program skal køre og hvor længe det herefter skal vente. I nogle systemer kan man ved angivelse af forskellige parametre, sætte en prioritet på et program. Denne prioritet kan f.eks. afhænge af hvor stort programmet er, og hvor mange sider papir man forventer udskrevet på en linieskriver. Ikke alle sådanne tidstildelings-strategier er lige effektive, hvorfor man kan komme ud for at skulle vente længe på afslutningen af sit program. Det skal lige bemærkes at bare nogle se-

kunders ventetid kan forekomme en som en uendelighed af tid, når man sidder foran en skærmterminal.

Alle disse strategier m.v. er ikke noget den almindelige bruger af EDB-systemet skal bekymre sig om (andet end ventetiden), idet opstart af et program i de fleste systemer foretages med en "RUN"-kommando, hvorefter der evt. kan/skal angives nogle kørselsparametre. F.eks. kunne kørslen af et program "PRIMTAL" (i et ikke nærmere specificeret system) startes ved at skrive:

```
:RUN,PRIMTAL,5,20
```

hvor 5 kunne være antal rekvirerede sekunder centralenheds-tid og 20 kunne være en angivelse af at man forventede 20 siders udskrift på linieskriveren.

I COMAL-80 under CP/M systemet, vil en kørsel af et program beliggende på en diskette bestå af 2 kommandoer nemlig:

```
*LOAD PRIMTAL
* RUN
```

hvor LOAD læser programmet over i datamatens primærlager og RUN starter programafviklingen.

9.6. FEJLUDSKRIFT VED FILBEHANDLING

I alle EDB-systemer vil der være en eller anden form for fejludskrift i tilfælde af forkerte eller fejlbehæftede filmanipulationer. Disse fejlmeddelelser kan være alt fra et tal til en virkelig god og meningsfyldt tekst-udskrift. Uanset hvad der udskrives, vil der altid til et leveret system følge en manual over de mulige fejl man kan lave, hvilket betyder at man ved opslag i denne manual kan få en uddybning af den måske i første omgang kryptiske fejlmeddelelse. Igen gælder det her, at kun ved gang på gang at dumme sig og herefter foretage opslag i manualen og/eller spørge bedrevidende, lærer man efterhånden at kunne tyde fejludskrifterne, og forstå hvorfor de kom.

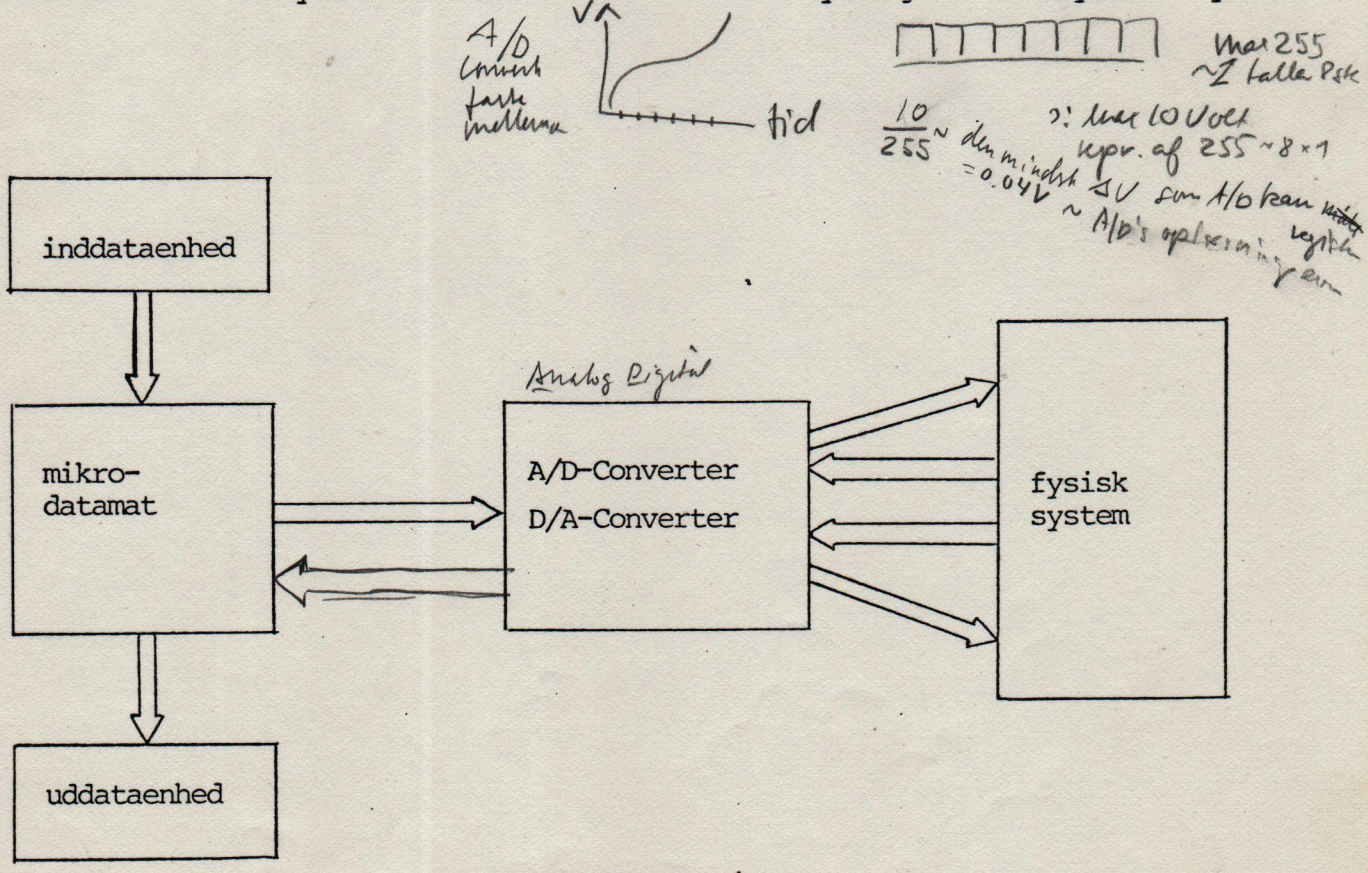


10. PRAKTISK ANVENDELSE

I afsnit 3.5. blev der givet et eksempel på, hvordan man kan anvende et datamatisk system til styring af en potentiometrisk titrering, og samtidig blev det nævnt, at styringen normalt kan ske fra en titratorenhed indeholdende en microprocessor. Den kvikke læser vil straks spørge om, hvorfor man skal bruge en datamat til styring, når der efterhånden findes så mange analyseapparater på markedet, der indeholder en microprocessor, så de kan programmeres til at udføre alverdens ting.

Sammenligner man et programmerbart analyseudstyr og en datamat, vil analyseudstyret kun kunne anvendes inden for et af fabrikanten sat begrænset område, mens datamaten gør det muligt, at analyseudstyret kan anvendes på brugerens betingelser og dermed kan udnyttes optimalt. Desuden kan den samme datamat anvendes i forbindelse med forskelligt analyse udstyr, hvor man ellers skulle købe mange analyseapparater med hver sin indbyggede microprocessor. Datamaten der bedst anvendes i laboratoriet er en mikrodatamat, som i anskaffelsespris (incl. hjælpeudstyr) vil koste det samme som 1-2 programmerbare analyseapparater.

I figur 10.1. er skitseret, hvad der er nødvendigt at anskaffe, hvis man vil benytte en mikrodatamat til styring af et fysisk system.



Figur 10.1.

Der findes mange mikrodatamater på markedet men det er i denne forbindelse vigtigt at vide, om mikrodatamaten indeholder en multiprocessor. Tilstedeværelsen af en multiprocessor gør, at mikrodatamaten på en nem måde kan bearbejde resultater fra flere fysiske systemer på samme tid, hvilket besværliggøres uden multiprocessor. Til de fleste mikrodatamater anvendes en skærmterminal til ind- og udlæsning, men det vil også være praktisk at anskaffe en printer.

Datamaten forbindes til det fysiske system via to convertere (oversættere). A/D-converteren omsætter analoge signaler (signaler fra det fysiske system) til digitale signaler og D/A-converteren omsætter signaler den modsatte vej. Converterne skal ofte forbindes med flere målepunkter i det fysiske system, og da disse fås med forskelligt antal ind- og udgange, må man gøre sig klart, hvor mange ind- og udgange, man mindst har behov for. Hvad angår A/D-converteren, skal man være opmærksom på følgende 4 parametre:

1 : Opløsningsevne
2 : Forstærkning
3 : Indgangsimpedans
4 : Konverteringsfrekvens

En A/D-converter omsætter som sagt et analogt signal til et digitalt, hvor det digitale signal repræsenteres af et vist antal bit, der er bestemmende for konverterens opløsningsevne. Traditionelt kan man få A/D-convertere med 8 eller 12 bit opløsning. Med 8 bit får man en opløsning på $1/2^8 = 0{,}4\%$ og med 12 bit en opløsning på $1/2^{12} = 0{,}025\%$. Forstærkningen i en A/D-converter er enten fast (fra 1 til ca. 100 gange) eller kan varieres enten ved udskiftning af en IC-kreds med modstande, eller programmæssig (programmerbar forstærkning). Indgangene på A/D-converteren har en modstand (impedans), hvormed de vil belaste målepunktet i den fysiske opstilling. Disse impedanser afhænger af fabrikat og af forstærkning, og vil typisk ligge i området fra 10-100 kohm.

Det er altså vigtigt at man ikke f.eks. kobler A/D-konverteren direkte på et målepunkt med en høj impedans, da målepunktet derved vil blive belastet af A/D-konverteren. Er målepunktets impedans f.eks. 10 KΩ og A/D-konverteren ligeså, vil A/D-konverteren kun

registrere halvdelen af den faktiske spænding (se fig. 10.2.).

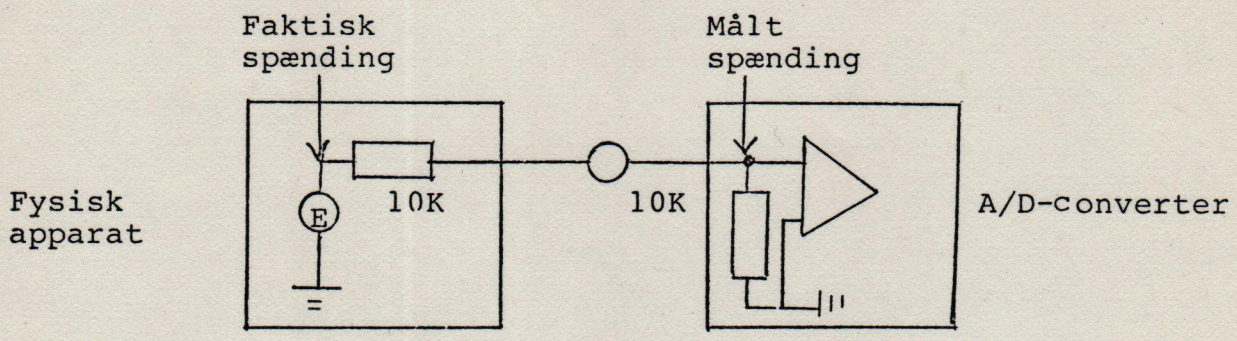


Fig. 10.2. Den målte spænding vil være den faktiske spænding, spændingsdelt over lige store modstande, altså halvdelen.

Hvad angår konverteringsfrekvensen (antal konverteringer/tidsenhed), er denne afhængig af teknologi og antal bit opløsning. Hvis der f.eks. er tale om en 12-bit opløsning, og datamaten har en 8 bit indgang, skal der for hver konvertering sendes 2 byte i stedet for én, hvilket halverer konverteringsfrekvensen. Hvis vi arbejder med en 8 bit konvertering er konverteringstiden af størrelsesorden 100 microsekunder, og hvis der f.eks. er 8 indgange ialt, vil det betyde at hver kanal bliver konverteret for hver $8 \cdot 100 = 0.8$ m sec. Der er her altså tale om en konverteringsfrekvens på 1/0.8 m sec ≈ ca. 1200 Hz.

Hvis det forholder sig således, at det fysiske system ikke "mærker", at det bliver styret af en datamat, snakker man om tidstro databehandling (kørsel).

Det er altså vigtigt for at få troværdige resultater, at konverteringsfrekvensen for A/D-konverteren er stor i sammenligning med ændringerne i det fysiske system.

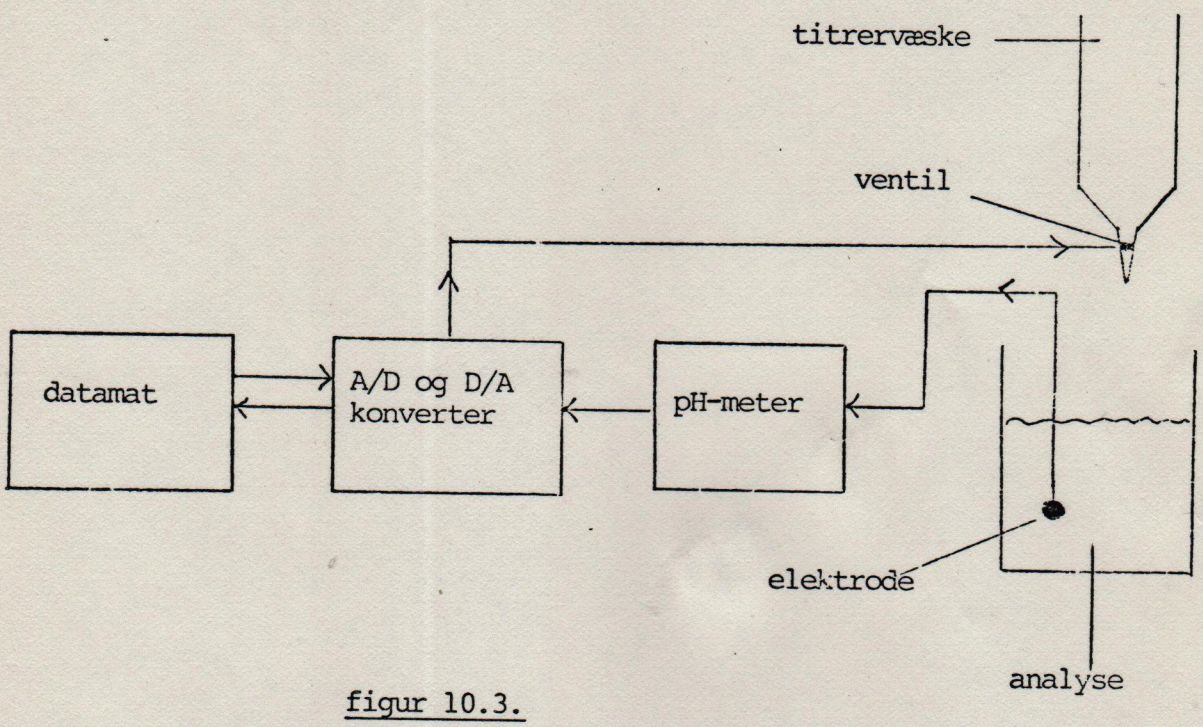
Hvis konverteringsfrekvensen er for lav, kan der opstå hazard, d.v.s. at en ændring i det fysiske system måske ikke bliver registreret, eller først bliver registreret, når det er for sent.

De følgende afsnit beskriver eksempler på fysiske systemer, der kan styres af en mikrodatamat, med angivelse af målepunkter samt en

skitse af det tilhørende pseudoprogram, der skitserer programforløbet til styringen.

10.1. POTENTIOMETRISK TITRERING

For nemheds skyld startes med den tidligere viste opstilling til potentiometrisk titrering. En forudsætning, for at benytte en mikrodatamat til at styre denne opstilling må være, at man løbende har en række ens analyser (f.eks. sure) der skal titreres med samme væske og til samme ækvivalenspunkt, d.v.s. samme pH-værdi.



figur 10.3.

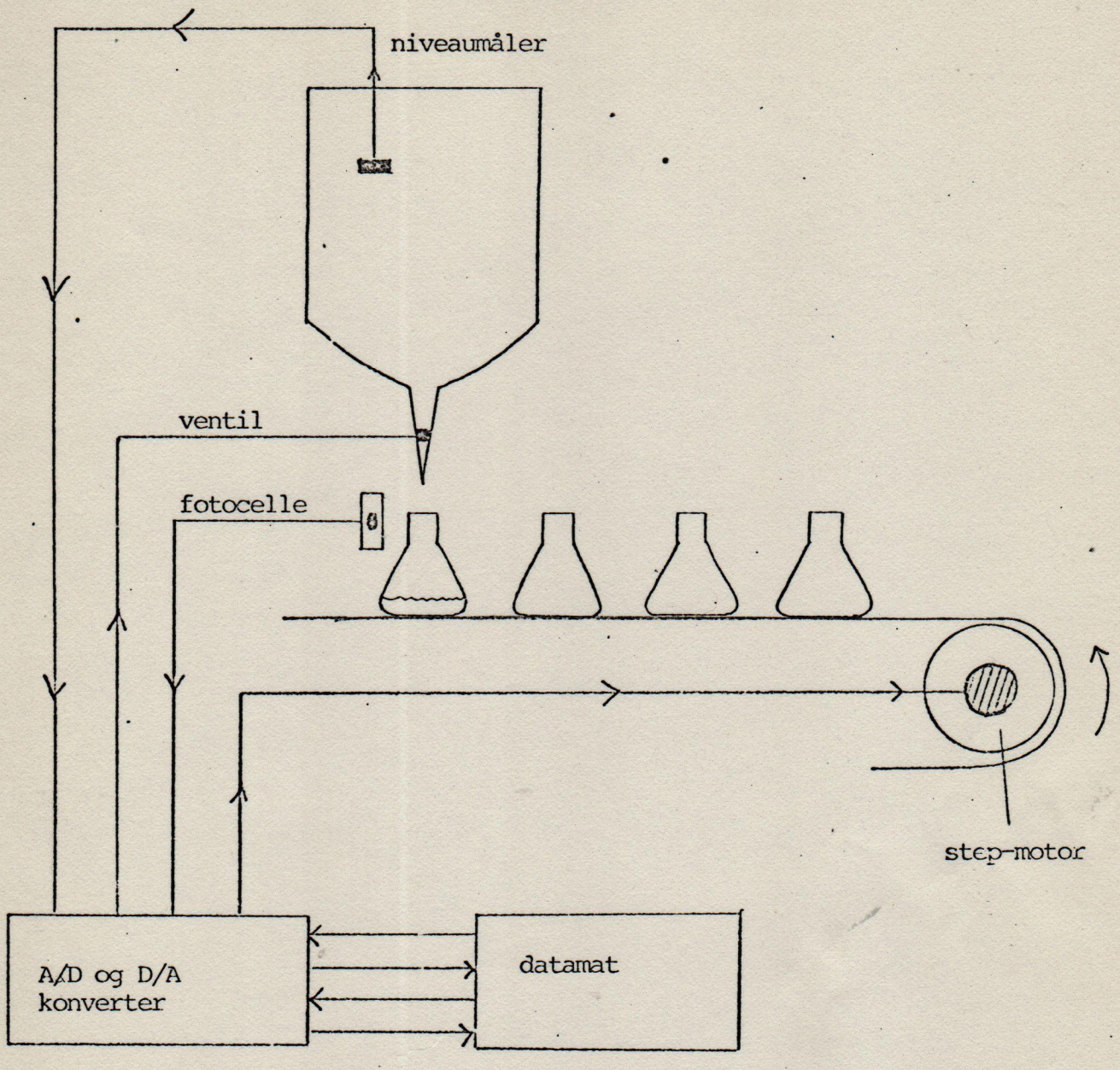
I denne opstilling er der kun et målepunkt: Kombielektroden, som har forbindelse til A/D-konverteren via pH-meteret. Beholderen med titrervæsken åbnes eller lukkes v.h.a. en ventil (f.eks. en magnetventil) i forbindelse med D/A-konverteren. Hvis ventilens åbningstid t sek. svarer til en tilsætning på k ml og "pH" betegner den målte pH-værdi og "grænseværdien" er pH ved ækvivalenspunktet, kan styringsprogrammet f.eks. se sådan ud:

```
1 Indlæs grænseværdi, t, k, titrervæskens koncentration
2 Sæt ML = 0
3 Gentag
4     Åbn ventil i t sek.
5     Sæt ML = ML + k
6 Indtil pH ≥ grænseværdi
7 Beregn koncentrationen
8 Udskriv koncentrationen
9 Stop
```

I eksemplet er grænseværdien og ventilens åbningstid variable, så programmet kan anvendes til forskellige titreringer, og skal derfor indlæses hver gang. Variablen ML tæller ml titrervæske, som skal benyttes til koncentrationsberegningen. Punkterne 3- 6 gennemløbes indtil grænseværdien er nået, hvorefter koncentrationen beregnes og udskrives i punkt 7 og 8.

10.2. DOSERING AF VÆSKER.

På laboratoriet hænder det ofte, at man skal afmåle et bestemt volumen af samme væske i et stort antal kolber. For at spare tid og undgå fejldoseringer kan den proces mekaniseres og styres af en mikrodatamat som skitseret i figur 10.4.



figur 10.4.

Kolberne er anbragt på et bånd, der flyttes frem af en step-motor. Afstanden mellem kolberne skal være den samme og skal svare til den afstand, båndet bevæger sig, når motoren er tændt i f.eks. t_1 sek. Væsken doseres fra en stor beholder ved, at ventilen åbnes i t_2 sek., hvilket svarer til et bestemt volumen. Opstillingen har to målepunkter, hvor det ene er en fotocelle, der benyttes til at registrere om der er en kolbe under doseringsbeholderen eller om kolben f.eks. er væltet. Det andet målepunkt måler væskeniveauet i doseringsbeholderen f.eks. med et flydelegeme (se også afsnit 10.3.), med det formål at systemet stopper, når beholderen er tom.

Pilenes retning angiver, at datamaten styrer ventilen og stepmotoren, samtidig med at den modtager besked fra de to målepunkter.

Hvis det forudsættes, at der skal være en kolbe under doseringsbeholderen ved systemets start, kan forløbet af programudsnittet være:

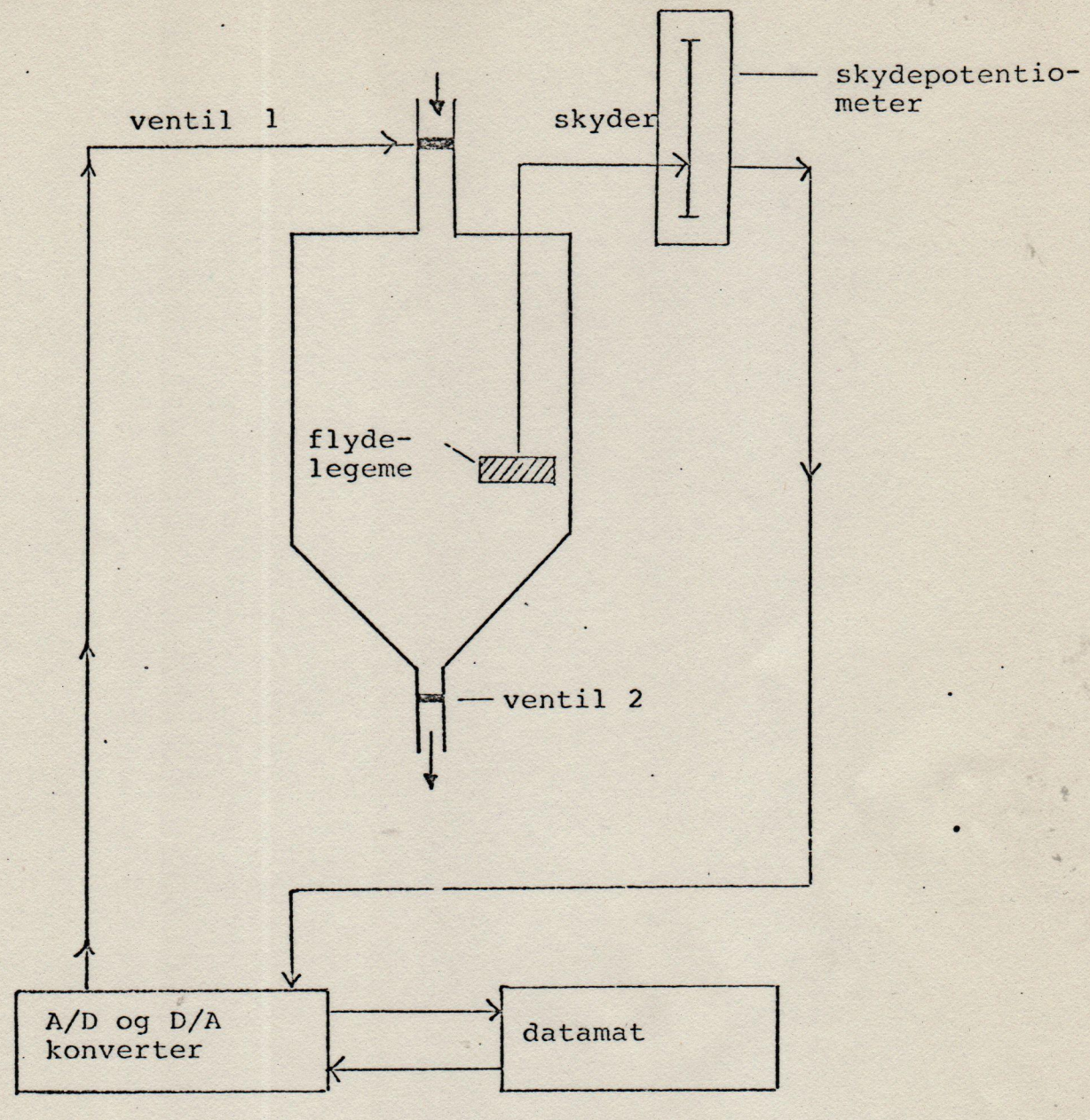
```
 1 Hvis doseringsbeholderen er tom eller kolben er væltet så
 2    Udskriv fejl
 3 ellers
 4    gentag
 5     Åbn ventil i t2 sek.
 6     Tænd motor i t1 sek.
 7    Indtil beholderen er tom eller kolben er væltet
 8    Udskriv fejl
 9 End
10 Stop
```

Indholdet af punkterne 1 og 2 er anbragt i starten af programmet for det tilfælde, at man har glemt at fylde væske i beholderen eller har glemt, at der skal være en kolbe under beholderen ved start. Hvis et af punkterne er opfyldt udskrives, at der er en fejl, og systemet sættes ikke igang. Hvis derimod punkt 1 ikke er opfyldt fortsættes til punkt 4 til 8, hvor doseringen foretages, og den næste kolbe kører hen under beholderen.

Programmet stopper kun, hvor beholderen er tom eller kolben er væltet. Det sidstnævnte er også opfyldt, når der ikke er flere kolber tilbage.

10.3 NIVEAUMÅLING.

Som nævnt i afsnit 10.2 er det muligt at måle væskeniveauet i en beholder v.h.a. et flydelegeme og få resultaterne omsat til digitale signaler. I fig. 10.5 er skitseret en opstilling til niveaumåling samt hvordan beholderen kan fyldes op.



Figur 10.5.

Ventil 1 foroven benyttes til at fylde beholderen op fra et reservoir og gennem ventil 2 forneden udtages et varierende flow. Flydelegemet er forbundet til skyderen på et skydepotentiometer v.h.a. en stiv stang. D.v.s. at når flydelegemet bevæger sig op og ned, bevæges skyderen på tilsvarende måde. Når skyderen er i øverste stilling

er beholderen "fuld" (der skal også være plads til flyderen), og når skyderen er i laveste stilling, har væskeniveauet den laveste tilladelige værdi, her kaldet minimum. Spændingen over den indskudte del af potentiometeret måles og omsættes af A/D-Converteren.

Hvis det er forudsat, at der er væske i beholderen ved start, og den højeste værdi for væskeniveauet kaldes maximum, kan det tilhørende programafsnit være udformet som følgende:

```
1 Gentag
2     gentag
3         åben ventil 1
4     indtil niveau ≥ maximum
5     gentag
6         luk ventil 1
7     indtil niveau < = minimum
8 forevigt
```

Påfyldningen sker ved gennemløb af punkterne 2 til 4 indtil maximumsværdien nås, hvorefter ventilen lukker. Løkken 5 til 7 gennemløbes indtil minimumsværdien nås, hvorefter der startes forfra ved punkt 1.

1 start
2 hvad min højde oplysning om
REPEAT
3 aftes niveau
4 hvis nivea < min så åbn ventil
5 hvis --- > max så luk ventil
6 UNTIL

11. STIKORDSREGISTER

A

Side:

A/D Converter.....10.2, 7.47
Addition (+)..... 7.17
Adresseoverføring..... 7.52
Aktuelle parametre..... 7.48
Algoritme..... 7.1
Analog-Digital-converter.....10.2,7.47
AND..... 7.37, 7.38
Anvendelse af EDB i laboratoriet..... 3.5
Anvendelse af EDB..... 3.2
Aritmetisk-Logisk enhed 4.7, 4.8, 4.9
Aritmetiske operatorer..... 7.17
Aritmetiske udtryk..... 7.15
ASCII-kode..... 9.10, 9.11, App.F-3
Assembler..... 4.5, 4.6

B

Baggrundslager, pladsmangel..... 9.8
Baggrundslager..... 7.24, 9.3
BCD-kode..... App. F-1
Beskyttelse af fil..... 9.19, 9.20
Betingede sætninger..... 7.31
Biblioteksprogram..... 9.26
Binære talsystem..... 4.2, App. E-1
Bit..... 4.2
Bitmønster..... 4.2
Blokdiagram..... 6.1-6.9
Bus..... 4.7, 4.9
Båndstation..... 5.2, 5.5

C

CASE WHEN OTHERWISE..... 7.39
Centralenhed..... 3.2, 4.7, 4.8
CLOSE FILE..... 7.25
Comal-80..... 7.1
Converter..... 3.5, 10.2
Cursor..... 7.4

Side:

D
D/A-Converter.....10.2
Data..... 3.1, 3.2, 9.2
Datamatens opbygning..... 4.1
Datamatisk system..... 3.2
Datamedier..... 5.1
Datatransmission..... 9.16
Decimalsystemet..... App. E-1
Digital-Analog-converter..... 10.2
DIM..... 7.9, 7.11, 7.12
Direkte struktureret fil..... 9.4, 9.14, 9.18
Diskette..... 5.3, 5.5
DIV..... 7.17, 7.18, 7.19
Division (/)..... 7.17
DRAM..... 9.2
Dyadiske operatorer..... 7.17, 7.37

E
EBCDIC-kode..... App. F-2
EDB..... 3.1
Editor..... 9.9, 9.11, 9.14
ENDCASE..... 7.39
ENDIF..... 7.34
ENDPROC..... 7.48
EXEC..... 7.50

F
Fejludskrift..... 9.29
Fil, beskyttelse..... 9.19, 9.20
Fil, definition..... 9.2
Fil, direkte struktureret..... 9.4, 9.14, 9.18
Fil, formatteret..... 9.11, 9.14
Fil, kopiering..... 9.15, 9.17
Fil, lukning..... 7.25
Fil, læsning..... 7.25, 9.10, 9.11
Fil, opbevaring..... 9.2, 9.3
Fil, oprettelse..... 9.6
Fil, sekventiel..... 9.4, 9.14, 9.18

Side:
Fil, skrivning..... 7 30, 9.8, 9.9
Fil, sletning..... 9.12, 9.13
Fil, uformatteret..... 9.11, 9.14
Fil, ændring..... 9.14
Fil, åbning..... 7.25
FILE..... 7 25
Filkatalog..... 9.8, 9.22
Filnavne..... 9.7, 9.10
Filstatus..... 9.21
Filtyper..... 9.4
Floppy-disk..... 5.3, 5.5
Formatteret fil..... 9.11, 9.14
Formelle parametre..... 7.48
Forstærkning..... 10.2
FOR TO STEP NEXT..... 7.41
Funktioner..... 7.47

H
Heltalsdivision..... 7.17
Heltals variable..... 7.7
Hjælpeprogram..... 9.25, 9.26
Holleriths kode..... 5.1, App. F-1
Hulkort..... 5.1, 5.5
Hulstrimmel..... 5.2, 5.5

I
IF THEN..... 7.33
IF THEN ELSE..... 7.34
IF THEN ELIF ELSE..... 7.35
Inddata enhed..... 3.2, 4.3, 4.7, 4.9, 5.1
Indgangsimpedans..... 10.2
Indiceret variabel..... 7.10, 7.12
Indre lager..... 3.2, 4.7, 4.8, 4.9
INPUT..... 7.21, 7.23
INPUT FILE..... 7.25
Interviewteknik..... 8.8

K
Kasettebånd..... 5.3
Kodeformer..... App. F

Side:

Kontkatenere..... 9.17
Konkatenering..... 7.21
Konstanter..... 7.13
Konverteringsfrekvens.....10.2, 10.3
Kopiering af filer..... 9.15, 9.17

L
Laboratorieanvendelse af EDB.....10.4, 10.5, 10.7
Linieskriver..... 5.4, 5.5
LIST .....App. A
Logiske udtryk..... 7.37
LOOP ENDLOOP..... 7.46
Lukning af filer..... 7.25
Læsesætninger..... 7.21
Læsning af filer..... 7.25, 9.10, 9.1

M
Magnetbånd..... 5.2, 5.5
Magnetplade..... 5.3, 5.5
Microkode..... 4.5, 4.6
MOD..... 7.17, 7.18, 7.1
Monadiske operatorer..... 7.17, 7.37
Multiplikation (*)..... 7.17
Multiprocessor.....10.2

N
NOT..... 7.37, 7.38
Numeriske variable..... 7.7

O
Oktale talsystem.....App. F-3
OPEN FILE..... 7.25
Operativsystem..... 4.5, 4.6
Operatorer..... 7.17, 7.20
Opgaveløsninger.....App. C
Opgaver.....App. B
Opløsningsevne.....10.2
Oprettelse af fil..... 9.6
Opstart af COMET-3000 datamat..... .....App. A
OR..... 7.37, 7.38

Side:

Ordliste.....App. D
Ottetalsystemet.....App. F-3
Oversættelse af programmer..... 9.23, 9.24

P
Parametre..... 7.48
Pladelager..... 5.3
Pladelagerstation..... 5.3, 5.5
Potensopløftning (↑)..... 7.17, 7.19
PRINT..... 7.28, 7.29
PRINT FILE..... 7.30
Printer..... 5.4, 5.5
Prioritet..... 7.17
Problemanalyse..... 8.1, 8.4, 8.5
PROC..... 7.48
Procedurer..... 7.47
Processor..... 3.2, 4.7, 4.9
Programafprøvning..... 7.55
Programbygger..... 9.27, 9.28
Programkørsel..... 9.28
Programmer..... 3.2
Programmering..... 7.1
Programtæller..... 4.7, 4.8

R
RAM..... 9.2
READ..... 7.25
READ-DATA..... 7.21, 7.22
Reelle variable..... 7.7
Registre..... 4.7, 4.8, 4.9
Regneenhed..... 4.3, 4.4
Relationsoperatorer..... 7.38
REPEAT UNTIL..... 7.44
Repeterende sætninger..... 7.41

S
Sammenkædning af programmer..... 7.21, 9.17,9.28
SAVE.....App. A

Side:

Sekventiel fil..... 9.9, 9.14, 9.18
SELECT OUTPUT..... 7.30
Semantik..... 7.2
Skrivebordstest..... 7.55, 7.56,7.58
Skrivesætninger..... 7.28
Skrivning af filer..... 7.30, 9.8, 9.9
Skærmterminal..... 5.4, 5.5
Sletning af fil..... 9.12, 9.13
Stopdata..... 6.6
Stopværdi..... 7.41
Styreenhed..... 4.7, 4.8, 4.9
Styrevariabel..... 7.41
Styring v.h.a. EDB.....10.1
Subtraktion (-)..... 7.17
Syntaks..... 7 2
Syntaksanalyse..... 7.4
Sætninger..... 7.3
Sætningstyper..... 7.15

T

Talkonstanter..... 7.13, 7.14
Talsætvariable..... 7.8, 7.12
Tekstkonstanter..... 7.13, 7.15
Tekstoperatorer..... 7.20
Tekstvariabel..... 7.11, 7.12
Terminal..... 5.3
Testdata 7.55
Tekstudskrift..... 7.60, 7.62
Tildelingssætninger..... 7.15, 7.20
Titalsystemet..... 4.2, App. E-1
Totalsystemet..... App. E-1
Trinvis programforbedring..... 6.10-6.12

U

Uddata enhed..... 3.2, 4.3, 4.7, 4.9,5.
Uformatteret fil..... 9.11, 9.14
Underprogram..... 7.47

V Side:
Variabel, heltallig..... 7.7
Variabel, numerisk..... 7.7
Variabel, reel..... 7.7
Variabel, tekst..... 7.11, 7.12
Variabeltyper..... 7.6
Variable, indicerede..... 7.10, 7.12
Variable, talsæt..... 7.8, 7.12
Variable..... 7.5
Værdioverføring..... 7.52

W
WHILE ENDWHILE..... 7.43

Y
Ydre enheder..... 5.1

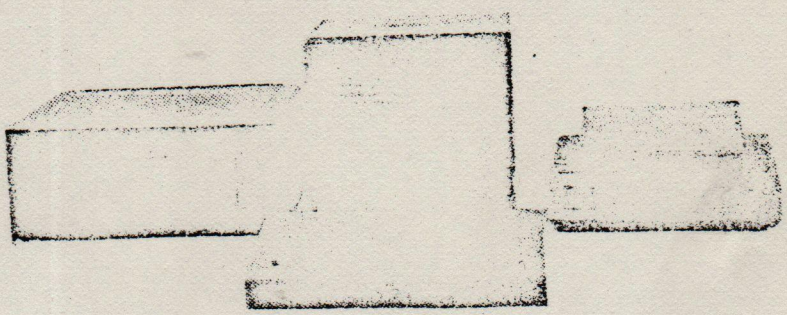
Æ
Ændring af fil..... 9.14

Å
Åbning af filer..... .... 7.25

APPENDIX A

OPSTART AF COMET 3000

OPSTART
AF
COMET 3000
MICRODATAMAT



INSTRUKTIONSBOG
FOR
BEGYNDERE

POUL KASTBERG

Forord

Dette er hverken en lærebog eller en manual, men en kortfattet, meget udførlig instruktion til den, som første gang skal starte systemet op. Det er tanken, at man skal benytte instruktionsbogen, mens man sidder ved udstyret. Man vil næppe få meget ud af at studere bogen uden at have udstyret.

Man kan klare sig med denne beskrivelse, hvis man blot skal køre et program. Man kan måske have nytte af beskrivelsen, hvis man skal skrive eller ændre et program, men beskrivelsen er langt fra komplet.

Ønsker man en komplet beskrivelse, kan man anvende de manualer, som ICL udsender.

Ollerup, den 9/12 1980

Torsten Alf Jensen

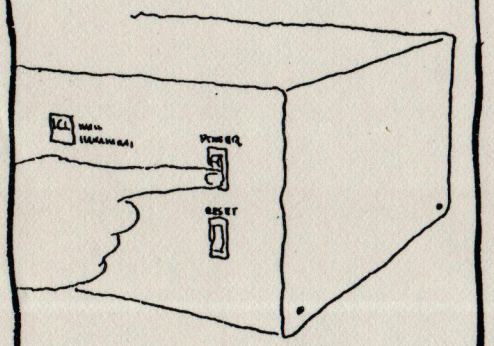
Statens Erhvervspædagogiske Læreruddannelse
CAI-Laboratoriet
Svendborgvej 10
5762 V. Skerninge

Indhold

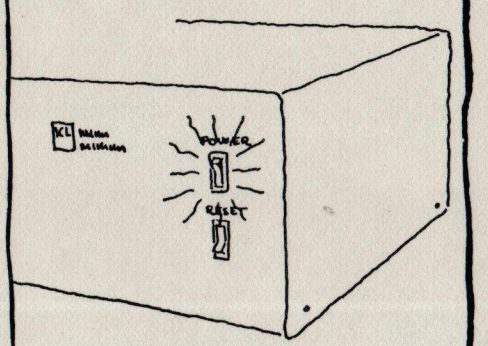
	Side
Sådan forbindes udstyret	4
Sådan tændes udstyret	8
Sådan virker tastaturet	10
Sådan startes COMAL-80 op	15
Sådan kører man et færdigt program, som ligger på en diskette	18
COMAL-kommandoer i alfabetisk orden	20
Formatering af disketter	31

SÅDAN TÆNDES UDSTYRET

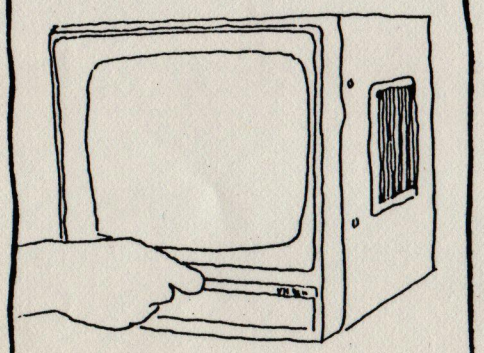
Tænd på forsiden af datamaten



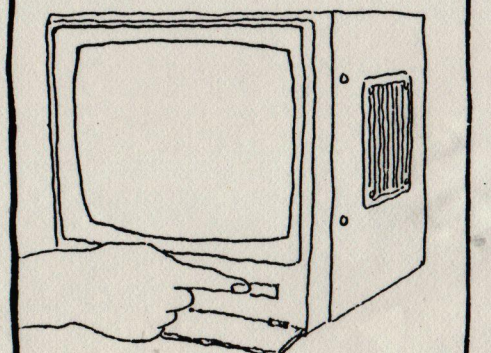
Herved tændes lyset i knappen



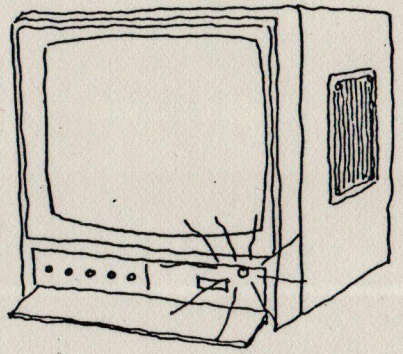
Luk op for betjening af skærmen



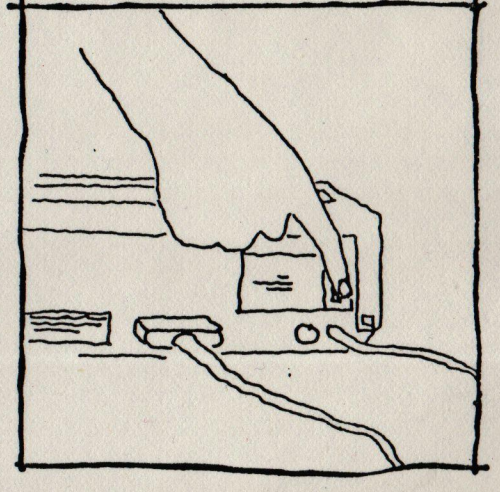
Tænd for skærmen



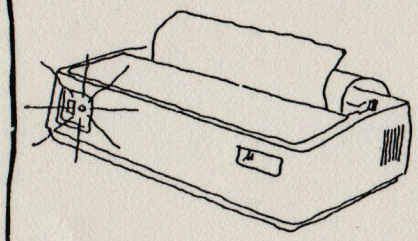
Herved tændes dette lys



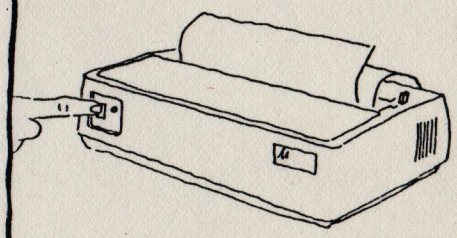
Tænd på bagsiden af printeren



Herved tændes dette lys

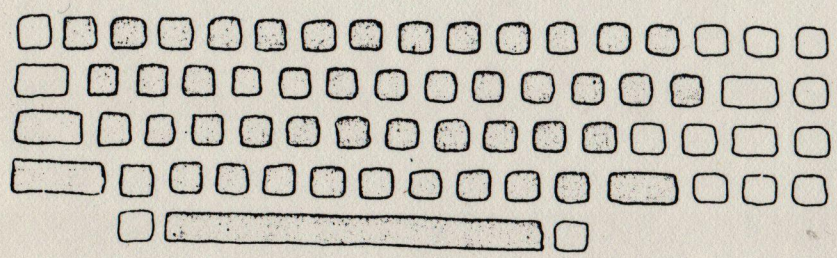


Tænd på forsiden af printeren

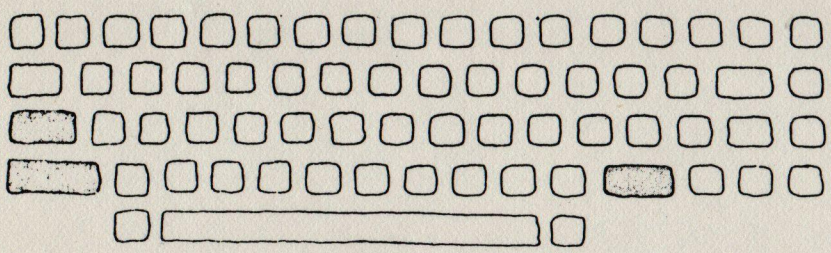


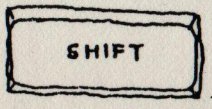
SÅDA VIRKER TASTATURET

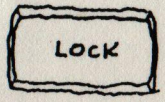
Tastaturet er opbygget som et almindeligt skrivemaskinetastatur suppleret med nogle specialtaster.



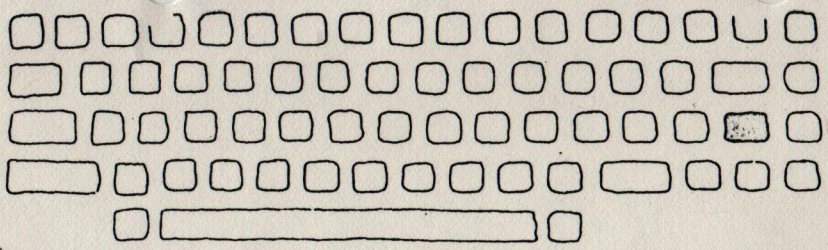
De taster, som er fremhævet på figuren ovenfor, svarer helt til tasterne på en skrivemaskine.



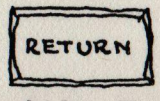




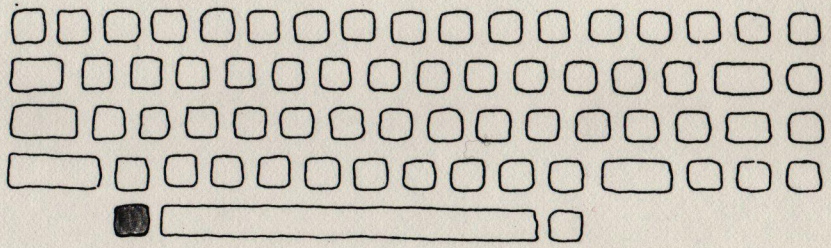
benyttes som på en skrivemaskine, når man vil have store bogstaver eller de specialtyper, som sidder oven over tallene på tastaturet. Shift virker så længe, den holdes nede. Lock virker fra der er trykket på den, til der trykkes igen.



47
CENT
ENTER

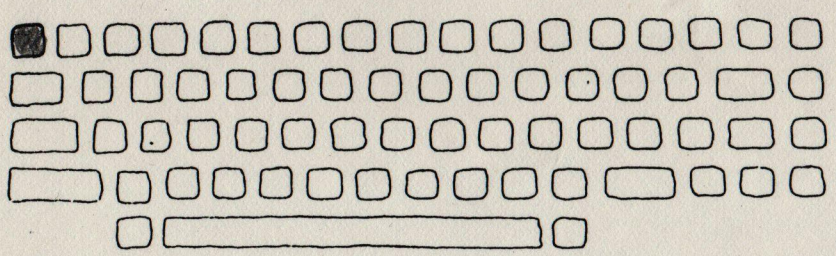


benyttes som på en skrivemaskine til linieskift. Tryk på RETURN bevirker imidlertid også, at det, som er skrevet indtil der trykkes return, behandles af datamaten.



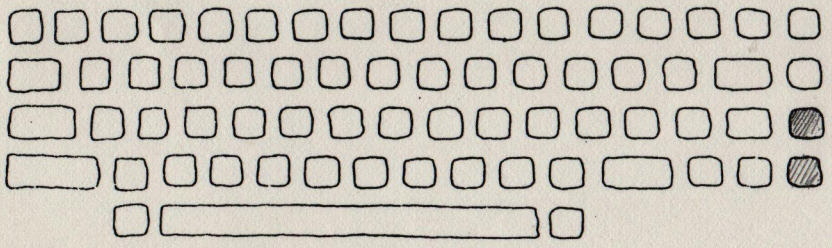


benyttes, når man ønsker store bogstaver. Alphalock har i modsætning til shift ingen virkning på andet end bogstaver (altså bl.a. ikke på tallene).

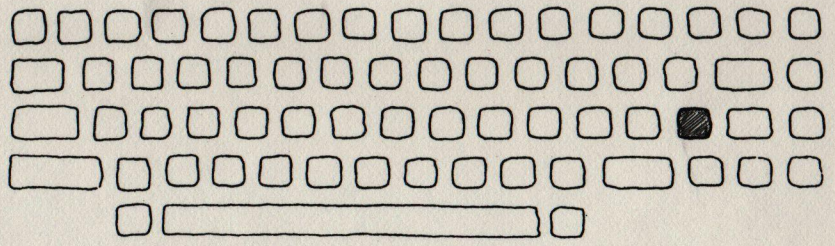


ESC = "escape" benyttes til at hoppe ud af en eller anden situation.

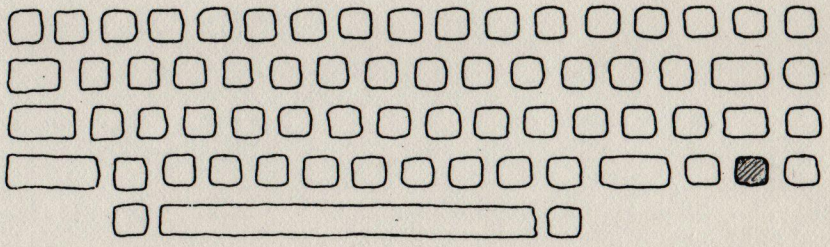
Under kørsel af et program vil man således normalt kunne standse kørslen ved at trykke på escape.



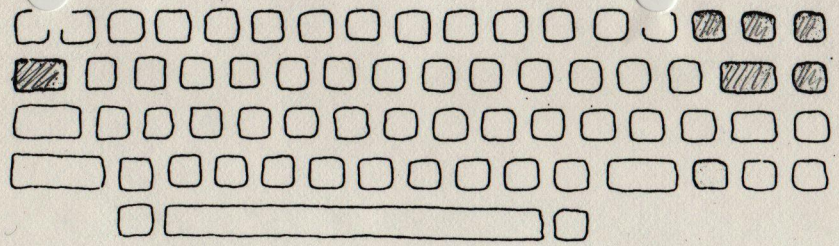
Benyttes til at flytte markøren frem og tilbage i den linie, som markøren står i. Markøren kan dog normalt ikke flyttes længere ud på linien, end der er skrevet.



DEL = "delete" benyttes til at fjerne tegn fra skærmen. Et tryk på DEL fjerner det tegn, som markøren står ved , og rykker det som står til højre for markøren på linien en plads til venstre.



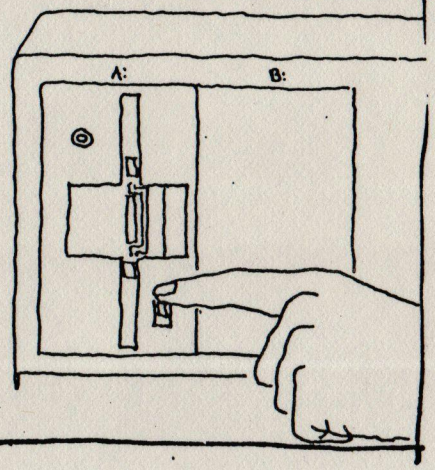
INS = "insert" benyttes, når der skal indsættes tegn i en allerede skrevet linie. Et tryk på INS flytter den del af linien, som står til højre for markøren en plads til højre. D.v.s. at linien "åbnes" en plads for hver gang der trykkes på INS.



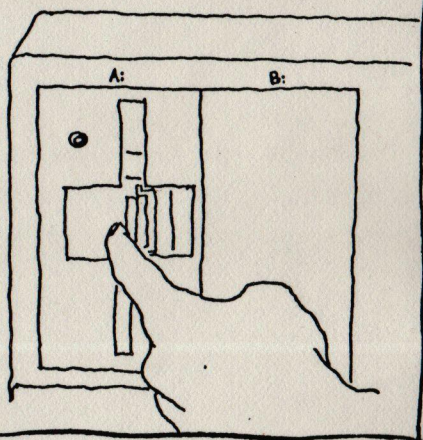
De tegn, som er fremhævet på figuren ovenover, benyttes normalt ikke, når man kører COMAL.

SÅDAN STARTES COMAL-80 OP

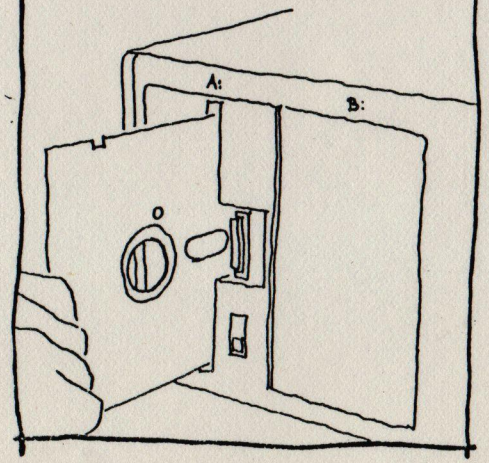
Lås op for diskettedrevet ved at trykke låseknappen ned



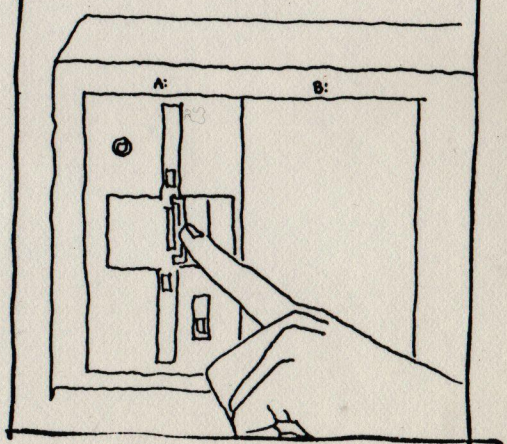
Luk op for spalten



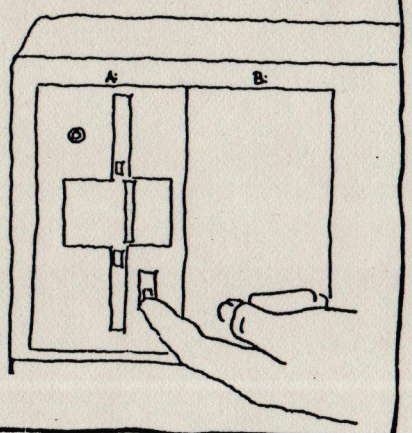
Skyd systemdisketten ind sådan:



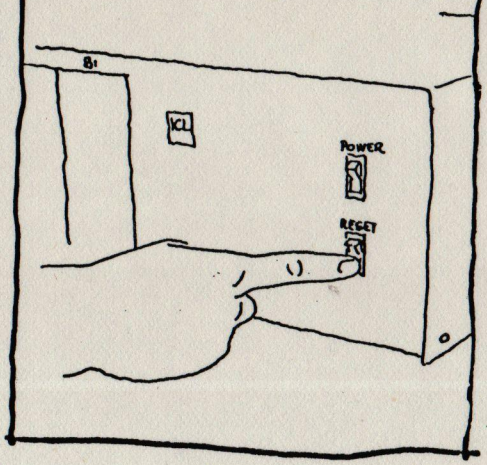
Luk spalten igen



Skub sikkerhedslåsen op



Tryk på reset-knappen



Skriv nu:
COMAL-80 og tryk return

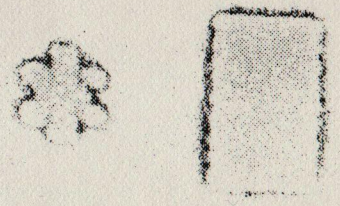
Nu kommer dette billede på skærmen

Nu kommer dette billede på skærmen

Tryk 1 på tastaturet

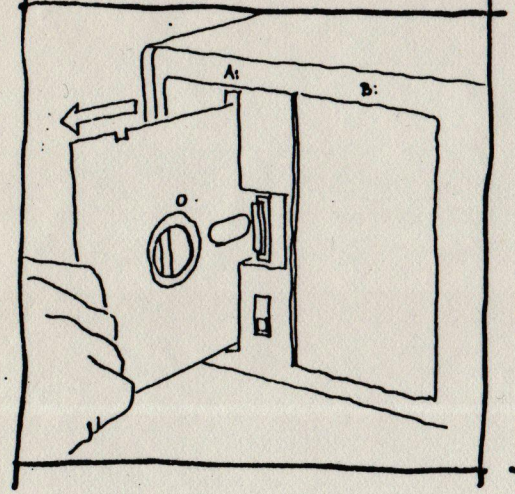
Skriv J - så får du udførlige fejlmeddelelser ved evt. fejl

Når der er stjerne, er COMAL 80 klar

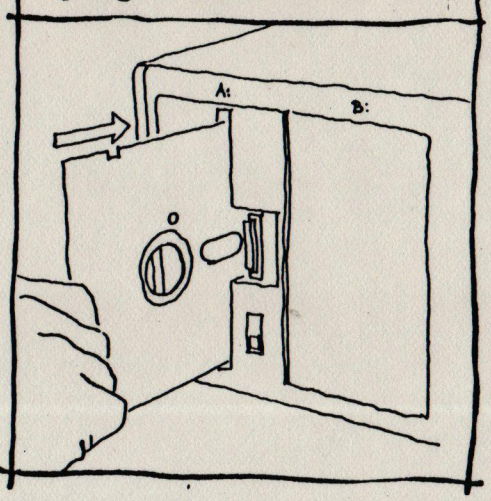


SÅDAN KØRER MAN ET FÆRDIGT PROGRAM, SOM LIGGER PÅ EN DISKETTE

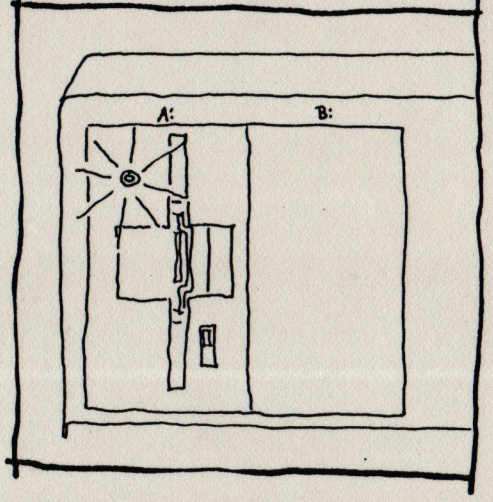
Tag systemdisketten ud



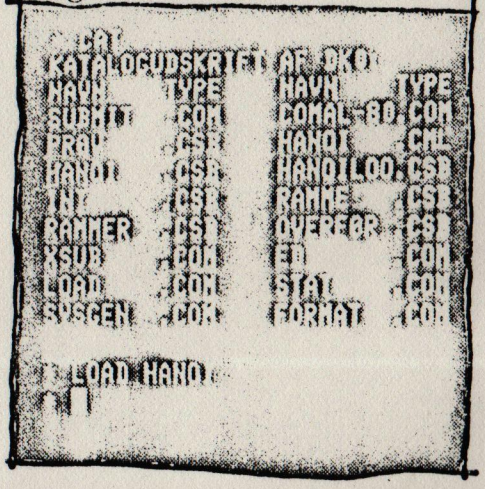
Indsæt disketten med programmet



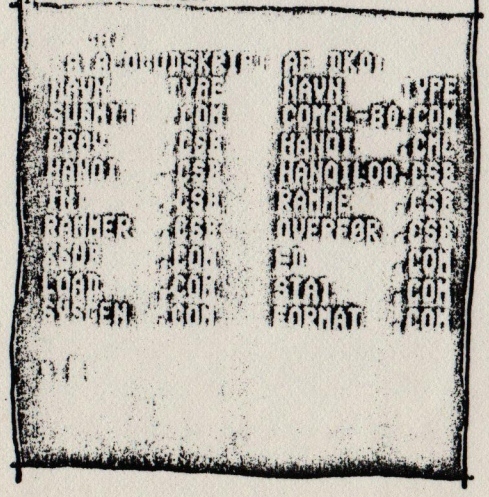
Når lampen lyser læses der fra disketten



Når der kommer * er programmet læst ind i arbejdslageret



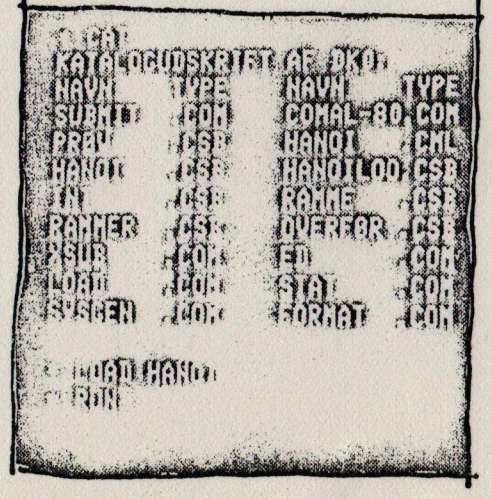
Skriv CAT - så får du en oversigt over de filer som ligger på disketten



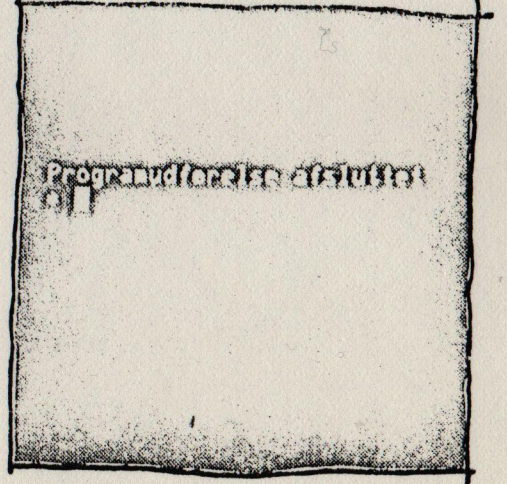
Skriv load program-navn



Skriv RUN - så kører programmet



Når programmet er færdigt fås denne udskrift:



COMAL-KOMMANDOER
I ALFABETISK ORDEN

Nogle COMAL-kommandoer kan kun anvendes som kommandoer. Andre kan både anvendes som kommandoer og som sætninger i et COMAL-program. I nedenstående alfabetiske liste er ikke medtaget ord, som kun sjældent benyttes som kommandoer.

AUTO

Auto benyttes til automatisk nummering af linierne, når man indlæser programmer.

AUTO — alene bevirker, at der startes med linie 10 og springes 10 linier ad gangen.

AUTO 100 — bevirker, at der startes i linie 100 og springes 10 ad gangen.

AUTO 100,5 — bevirker, at der startes i linie 100 og springes 5 linier ad gangen.

40,10 40 50 60 osv....

CAT

CAT bevirker, at der udskrives en oversigt over de filer, som ligger på disketten.

CLEAR

CLEAR bevirker, at alt, hvad der står på skærmen, slettes, og markøren sættes i venstre øverste hjørne. (CLEAR har ingen indvirkning på det, der står i arbejdslageret).

CLOSE

CLOSE — alene bevirker, at alle filer lukkes.

CLOSE 1 — bevirker, at filen, som har fået tildelt nr. 1, lukkes, mens evt. andre åbne filer forbliver åbne.

CON

CON — benyttes, når man ønsker at køre eller fortsætte kørslen af et program uden at slette de værdier, som de variable allerede har.

CON — alene bevirker, at programmet, som ligger i arbejdslageret udføres herfra.

CON 30 — bevirker, at udførelsen af programmet, som ligger i arbejdslageret, starter i linie 30.

DEL

Benyttes til at slette linier i det program, som ligger i arbejdslageret.

DEL 60 sletter linie 60.

DEL 60,120 sletter linierne fra 60 til 120 begge inklusive.

DEL ,60 sletter alle linier til og med 60.

DEL 60, sletter alle linier fra og med 60.

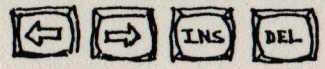
DELETE

DELETE benyttes til at slette filer på disketten.

DELETE SKAT.CSB sletter programfilen med navnet SKAT.CSB . Der skal angives, hvilken type fil, der skal slettes. CSB angiver, at filen er en binær programfil.

EDIT

EDIT benyttes til at ændre linier i det program, som ligger i arbejdslageret, uden at man behøver at skrive linierne om.

EDIT 60 bevirker, at linie 60 skrives på skærmen. Man kan herefter rette i linien ved hjælp af tasterne: 

samt ved at skrive "oven i". Rettelsen afsluttes, når der trykkes return.

ENTER

Benyttes til at hente en ASCII programfil fra disketten til arbejdslageret. En ASCII fil er lagt ud på disketten med kommandoen LIST .. Hvis en programfil hentes ind i arbejdslageret med ENTER slettes indholdet af arbejdslageret ikke. Man kan således lave en kombination af et program, som ligger i arbejdslageret og et program, som ligger på disketten. ENTER SPIL henter den fil fra disketten, som har navnet SPIL.

FORMAT

Kommandoen FORMAT benyttes kun i forbindelse med kasettestationer. COMAL-kommandoen FORMAT kan altså ikke benyttes til at formatere disketter. Disketter skal formatteres med CP/M programmet FORMAT. (Se side 31).

INIT

INIT benyttes til at initiere en diskette, d.v.s. starte en diskette, som allerede har været brugt til at gemme program- eller datafiler, og som lige er sat ind i diskettedrevet. Hvis man undlader at skrive INIT , kan man godt læse fra men ikke skrive på disketten. Hvis disketten er helt ny, kan den ikke startes med INIT . Den skal først formateres, og det kan ikke gøres fra COMAL. (Se side 31).

LET

LET benyttes til at tildele variable værdier.

LET A=7 bevirker, at A får værdien 7. Kommandoen LET har altså helt samme virkning som COMAL-sætningen LET . Kommandoen benyttes oftest, når man har afbrudt kørslen af et program og ønsker at "håndændre" enkelte variable, inden man kører videre med CON . Kommandoen LET A=7 kan kun udføres, hvis A allerede er oprettet under kørslen af et program. Ved både kommandoen og sætningen LET kan ordet LET udelades, og man kan nøjes med A=7

LOAD

Benyttes til at hente en binær programfil fra disketten til arbejdslageret. En binær fil er lagt ud på disketten med kommandoen SAVE . Hvis en programfil hentes ind i arbejdslageret med LOAD slettes det hidtidige indhold af arbejdslageret.

LOAD SPIL henter den fil fra disketten, som har navnet SPIL.

SAVE SPIL gemmer SPIL på disk. und GARF,CSB

LIST

LIST kan benyttes på flere måder.

LIST — alene bevirker, at det program, som ligger i arbejdslageret skrives ud på skærmen. Hvis programmet fylder mere end en side, kan man blade en side frem

ved at trykke på mellemrumstangenten. Tryk på return bevirker, at udskriften flytter én linie frem.

LIST 10,70 — bevirker, at linierne fra linie 10 til linie 70 udskrives på skærmen.

LIST LP: — bevirker, at det program, som ligger i arbejdslageret, udskrives på printeren.

LIST 10,70 LP: — bevirker, at linierne fra 10 til 70 udskrives på printeren.

LIST SPIL — hvor SPIL er et vilkårligt valgt navn, skriver indholdet af arbejdslageret ud på disketten som en ASCII programfil med navnet SPIL. Inden LIST anvendes på denne måde, skal disketten være initieret (se INIT).

LIST LP: skriv program ud

MAT

MAT benyttes til at tildele alle elementer i en indiceret variabel en bestemt værdi.

MAT A=7 bevirker, at alle elementer i A får værdien 7.

Kommandoen MAT har altså helt samme virkning som COMAL-sætningen MAT . Kommandoen benyttes oftest, når man har afbrudt kørslen af et program og ønsker at "håndændre" enkelte variable, inden man hører videre med CON. Kommandoen MAT A=7

kan kun udføres, hvis A allerede er oprettet ved kørsel af et program.

NEW

NEW bevirker, at indholdet af arbejdslageret slettes.

OPEN

OPEN benyttes til at åbne datafiler med henblik på skrivning eller læsning. Kommandoen OPEN svarer helt til COMAL-sætningen OPEN .

OPEN FILE 1,POINT,RANDOM,100 åbner en random datafil med navnet POINT og tildeler denne fil nummer 1. 100 angiver bloklængden.

OPEN FILE 2,RESUL,WRITE åbner en sekventiel datafil med navnet RESUL til skrivning. Filen tildeles nummeret 2.

OPEN FILE 2,RESUL,READ åbner den sekventielle datafil med navnet RESUL til læsning. Filen tildeles nummeret 2.

Der kan ikke åbnes flere filer med samme nummer. Der kan højst være 8 datafiler åbne på en gang. Når en sekventiel fil er lukket, kan den ikke igen åbnes for skrivning.

"Håndåbning" af filer benyttes normalt kun, når et program er standset på grund af fejl, og man ønsker at køre videre med CON med bestemte filer åbne.

PRINT

PRINT kan benyttes på flere måder. Kommandoen PRINT svarer helt til COMAL-sætningen PRINT .

PRINT ANTAL hvor ANTAL er et variabelnavn, bevirker at værdien af ANTAL skrives på skærmen. PRINT anvendes ofte på denne måde, når et program er standset på grund af en fejl, og man ønsker at se, hvilke værdier de variable har.

PRINT FILE 1:ANTAL hvor ANTAL er navnet på en variabel, bevirker at værdien af ANTAL skrives i den sekventielle fil, som ved åbning har fået tildelt nummer 1.

PRINT FILE 2,27:ANTAL hvor ANTAL er navnet på en variabel, bevirker at værdien af ANTAL skrives i post nr. 27 af den random-datafil, som ved åbning har fået tildelt nummer 2.

Når der skrives i filer ved anvendelse af PRINT, bliver data udskrevet i ASCII form. Sådanne datafiler kan kun læses med INPUT FILE. (Mens binære datafiler skrives med WRITE og læses med READ).

RENUM

Renumber

hvis man feter ere spritte en linje ind mellem fet 20 og 30

RENUM benyttes til at ændre numrene på linierne i programmet i arbejdslageret.

RENUM alene bevirker, at linierne i programmet tildeles numrene 10, 20, 30, 40 .....

RENUM 10:50,200,5 bevirker, at linierne fra 10 til 50 begge inklusive tildeles numrene 200, 205, 210 .. 200 angiver altså første nye linienummer, mens 5 angiver springet i nye linienumre. Der må ikke i forvejen ligge linier i programmet i det interval, der flyttes til. Linier, som ligger uden for intervallet fra 10 til 50 berøres ikke.

RENUM anvendt på denne måde kan benyttes til at flytte afsnit af programmet.

RUN

RUN benyttes, når man ønsker at udføre det program, som ligger i arbejdslageret. RUN medfører, at alle variable slettes, og alle datafiler lukkes inden kørslen af programmet starter.

RUN alene bevirker, at kørslen starter i programmets første linie (d.v.s. den linie, som har det laveste linienummer).

RUN 50 bevirker, at kørslen starter i linie 50.

SAVE

SAVE benyttes, når programfiler skal gemmes på disketten.

SAVE SPIL hvor SPIL er et valgt navn, laver en kopi på disketten af det program, som ligger i arbejdslageret. Programfilen bliver en binær fil med

navnet SPIL . Binære programfiler kan hentes ind i arbejdslageret med kommandoen LOAD . Inden SAVE benyttes, skal disketten være initieret. (Se INIT).

SELECT

SELECT OUTPUT benyttes til at angive på hvilket medium udskrift skal finde sted fra PRINT-sætninger i programmet. Kommandoen SELECT OUTPUT svarer helt til COMAL-sætningen SELECT OUTPUT.

I udgangssituationen vil skærmen være valgt som medium. (Kommandoen RUN bringer systemet til udgangssituationen).

SELECT OUTPUT "LP:" bevirker, at output fra print-sætninger kommer på printeren.

SELECT OUTPUT "DS:" bevirker, at output fra print-sætninger kommer på skærmen.

SELECT OUTPUT "LBNR" bevirker, at output fra print-sætninger kommer på datafilen med navnet LBNR.

SELECT OUTPUT som kommando bruges oftest, når et program er standset ved fejl, og man ønsker at ændre outputmedium inden man fortsætter kørslen med CON.

SIZE

SIZE benyttes til at skaffe oplysninger om, hvor meget af arbejdslageret der er brugt til program, hvor meget der er brugt til variable og hvor meget der er til disposition.

FORMATERING AF DISKETTER

Når en helt ny diskette tages i brug, eller når man ønsker at slette alt, hvad der står på en brugt diskette, skal disketten formateres. Dette kan ikke gøres ved hjælp af COMAL, men derimod ved hjælp af CP/M-programmet FORMAT. Følgende fremgangsmåde kan anvendes:

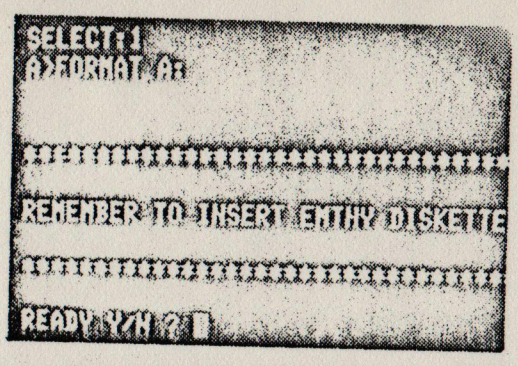
1. Kontroller at det, der står i arbejdslageret, ikke er noget, der skal bruges.
2. Sæt systemdisketten i diskettedrevet. (For at være sikret imod fejltagelser bør systemdisketten være skrivesikret - d.v.s. hakket i øverste kant af disketten skal være overklæbet med tape).
3. Tryk RESET, herved kommer dette billede på skærmen:



4. Tryk 1, herved kommer dette billede på skærmen:

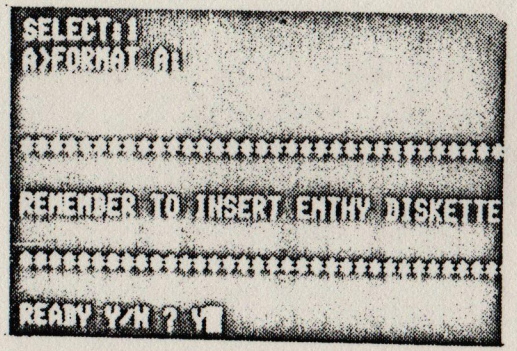


5. Skriv FORMAT A: herved kommer dette billede på skærmen:

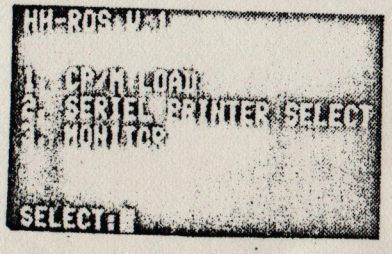


6. Udtag systemdisketten og indsæt den diskette, som skal formateres.

7. Skriv Y



8. Efter nogen tid kommer denne udskrift:



Disketten er nu formateret. Disketten udtages nu og COMAL-systemet læses ind igen på sædvanlig måde.

APPENDIX B

OPGAVER

Det første ciffer i opgavernes nummer refererer til det tilhørende teoriafsnit i kompendiet.

4-1

Antag at skuffe 1,2 og 3 i datamatmodellen fig. 4.2.a indeholder følgende:

Skuffe	Symbol	Værdi
1	A	9
2	B	5
3	C	4

hvor symbol = variabel

Alle andre skuffer indeholder 0.

Angiv med ovenstående som udgangspunkt ordrefølgen for følgende udtryk:

a) A=A-17

b) B=-B+3 · C

6-1

Lav et pseudoprogram der kan finde og udskrive maximum- og minimumsværdier i en talrække. F.eks. skal talrækken 119, -4, 22, 41, 16, 5 som resultat af programkørslen give maximum = 119, og minimum = -4.

6-2

Lav et pseudoprogram der kan finde og udskrive gennemsnittet af en række tal. F.eks. skal talrækken 23, 4, 9, 15, 1 efter endt behandling i programmet give resultatet 10.04, der jo er gennemsnittet af tallene.

6-3

Find de 10 mindste tal i nedenstående talfølge:

24623 25108 24243 24741 24192 25177 24749 25049 24231 24510 24198 24657 24496
24727 24858 24498 24271 24197 24387 24553 24288 25151 24413 24539 24802 24428
24432 24331 24873 25020 24985 24638 24657 24230 24756 25121 24240 25087 25034
24988 24318 24340 24690 24760 24778 24190 24611 24712 24960 24569 24253 24220
25165 24603 24732 24999 24691 24620 24632 24960 24454 24801 24449 25182 24986
24332 24340 24220 24351 24687 24606 24751 24298 24368 25070 24261 24665 24245
24283 24304 24861 24334 24226 24523 24957 24629 24924 24438 24680 24345 25154
24743 24375 24693 25159 24571 24710 25020 25050 24994 24681 24959 24306 24709
24221 24916 24560 24671 24448 25021 24570 25144 24350 24403 25071 24406 24467
24386 24916 25098 24843 24978 24679 24934 24324 24756 24645 24292 24978 25007
24858 24877 25179 24738 24646 24468 24685 24911 24803 24484 24715 25008 24338
24832 24850 24795 24473 24299 24294 24893

Løsning:

HVordan gjorde du?

Med din sorteringsmetode i erindring, skal du lave et pseudoprogram der sorterer en talrække på samme måde som du selv lige sorterede! D.v.s. at f.eks. talrækken 7, 11, 9, 2, 3, 7, 2 efter endt behandling i programmet skal se således ud:

2, 2, 3, 7, 7, 9, 11

6-4

Lav et pseudoprogram der i kartoteket "AZ" indsætter oplysningerne:

1: Medarbejdernr.
2. Navn
3: Adresse
4: Postnr.

om en ny medarbejder. Der skal tages behørig hensyn til, om der i kartoteket i forvejen findes en person med den nye medarbejders medarbejdernr. Kartotekskortene er ikke sorteret på nogen måde.

Til brug for pseudoprogrammeringen kan man f.eks. benytte at følgende pseudoordrer (Comal-80 orienterede):

1: Åben kartotek
2: Luk kartotek
3: Læs kartotekskort nr.: N (N=1..., heltal).
4: Skriv kartotekskort nr.: N (N.1..., heltal).
5: Hvis ⟨betingelse opfyldt⟩ så
    ordrer
   ellers
    ordrer
6: Indlæs fra skærm
7: Udskriv på skærm
8: Gentag
    ordre
   indtil betingelse opfyldt
9: STOP

Man er selvfølgelig velkommen til at benytte egne pseudoordre.

6-5

Lav et blokdiagram der finder og udskriver det største af 3 vilkårlige tal.

6-6

En virksomhed fremstiller stålarmaturer i serier. Før en serie sendes på markedet, foretages en stikprøvevis kontrol af armaturerne. Tegn et blokdiagram, der viser følgende sæt regler for produktionskontrol:

Der udtages en stikprøve på 25 stk. Hvis antallet af defekte armaturer ikke overstiger 1, godkendes serien. Hvis antallet af defekte armaturer overstiger 3, kasseres serien. Er antallet af defekte armaturer større end 1, men ikke større end 3, udføres en ny stikprøve på 25 stk. Overstiger antallet af defekte armaturer fra begge prøver tilsammen 5, kasseres serien, ellers godkendes den.

6-7

Udarbejd et blokdiagram, der kan anvendes til beregning af et fedtstofs forsæbningstal.

KOH-forbruget undersøges dels efter 1/2 times kogning og dels efter 1 times kogning. Følgende indlæsninger foretages derefter: N for KOH (N), g afvejet fedtstof til 1/2 times kogning (G1), ml KOH forbrugt efter 1/2 times kogning (M1), g afvejet fedtstof til 1 times kogning (G2), ml KOH forbrugt efter 1 times kogning (M2).

Efter indlæsningen ønskes forsæbningstallene efter henholdsvis 1/2 times kogning (F1) og 1 times kogning (F2) beregnet efter følgende formel:

$$\text{Forsæbningstal} = \frac{\text{ml KOH} \cdot N_{KOH} \cdot 56{,}11}{\text{g afvejet fedtstof}}$$

Herefter ønskes udskrift foretaget efter følgende regler:

$\lvert F1-F2 \rvert < 2$	Tag gennemsnit af F1 og F2 og udskriv resultatet.
$\lvert F1-F2 \rvert \geq 2$ og $F1>F2$	Udskriv: Fejl i analyse, udfør ny analyse.
$\lvert F1-F2 \rvert \geq 2$ og $F1<F2$	Udskriv: udfør ny analyse med forlænget kogetid.

Når 1 analyse er beregnet, går man videre til næste analyse. Stopdata er 0,0,0,0,0.

7-1

Lav et program, der beregner og udskriver kvadrattallet på tallene fra 0-14, og afprøv programmet.

7-2

Kemikaliefabrikken Løvo producerer bl.a. $FeCl_3,6H_2O$, som laboranterne løbende kontrollerer ved en renhedsbestemmelse, d.v.s. bestemmer indholdet af ren $FeCl_3$. Da der tit sker regnefejl, ønsker laboratoriechefen at få udarbejdet et EDB-program til beregning af $FeCl_3$-indholdet i analyserne.

En analyse udføres således:

Ca. 0,6 g $FeCl_3,6H_2O$ afvejes og fortyndes til 20 ml med vand, hvorefter der titreres med ca. 0,1 N $Na_2S_2O_3$. Der udføres to titreringer.

Beregningen sker efter:

$$\text{w/w\% } FeCl_3 = \frac{\text{ml } (Na_2S_2O_3) \cdot \text{N } (Na_2S_2O_3) \cdot 162{,}21 \cdot 100}{1000 \cdot \text{gram } FeCl_3,\ 6H_2O}$$

Lav et program til beregning af w/w% $FeCl_3$, idet forskellen på dobbeltbestemmelsens ml-forbrug ikke må overskride 0,05 ml. Hvis grænsen overskrides, skal der ikke foretages nogen beregning, men blot gives en udskrift om at analysen skal laves om.

Hvis grænsen er overholdt, beregnes w/w% $FeCl_3$, og resultatet udskrives. Som kontrol af indtastningen udskrives normalitet og ml forbrugt $Na_2S_2O_3$ og gram $FeCl_3,\ 6H_2O$.

7-3

Lav et program, der simulerer en potentiometrisk titrering på grundlag af analysens pH-værdi. Titreringen skal stoppe, når pH $\geq$ en fastsat grænseværdi. Titrervæsken tilsættes med et bestemt antal ml af gangen, hvilket svarer til en bestemt ændring af pH. Programmet skal udskrive ml forbrugt titrervæske.

APPENDIX C

OPGAVELØSNINGER

Løsning til opgave 4-1.

a) Udtryk: A=A-17

Skuffe	Indhold
1	A
2	B
3	C

Følgende ordresæt kan benyttes:

```
MVI,17    ; læg 17 i skuffe I
MIS,4     ; læg indholdet af I i skuffe 4
LDX,1     ; læg indholdet af skuffe 1 i skuffe x.
LDY,4     ; læg indholdet af skuffe 4 i skuffe Y.
SUB       ; træk skuffe Y fra X i R
MRS,1     ; læg indholdet af R i skuffe 1 (=A).
```

b) Udtryk: B=-B+3·C.

Følgende ordresæt kan benyttes:

```
LDX, 5    ; læg indhold af skuffe 5 i skuffe X
LDY,2     ; læg indhold af skuffe 2 i skuffe Y
SUB       ; træk skuffe Y fra X i R.
MRS,5     ; læg indhold af R i skuffe nr. 5
MVI,3     ; læg 3 i skuffe I
MIS,0     ; læg indholdet af I i skuffe 0
LDX,0     ; læg indholdet af skuffe 0 i skuffe X
LDY,3     ; læg indholdet af skuffe 3 i skuffe Y.
MUL  ;    ; gang skuffe X med Y i R
MRS,4     ; læg indholdet af R i skuffe 4
LDX,4     ; læg indholdet af skuffe 4 i skuffe X
LDY,5     ; læg indholdet af skuffe 5 i skuffe Y
ADD       ; læg skuffe X og Y sammen i R
MRS,2     ; læg indholdet af R i skuffe 2
```

Løsning til opg. 6-1

1. fase

```
Start
        Læs alle tal i talrækken og find Max. og Min.
        Udskriv Max. og Min.

STOP
```

2. fase

```
Start
        Læs første tal T i talrækken
        Max = T; Min = T

        Gentag
              Læs næste tal T i talrækken
              Hvis T større end Max så er Max = T
              Hvis T mindre end Min så er Min = T
        Indtil der ikke er flere tal i talrækken
        Udskriv Max. og Min.
STOP
```

Løsning til op. 6-2

1. fase

```
Start
        Læs alle tal i talrækken og find gennemsnittet G
        Udskriv G
STOP
```

2. fase

```
Start
        Gentag
              Læs et tal T
              Læg T til summen af de foregående tal.
              Læg 1 til antal af tal
        Indtil der ikke er flere tal i talrækken
        Beregn gennemsnittet G
        Udskriv gennemsnittet G
STOP
```

3. fase

```
Start
        Sum = 0
        Antal = 0
        Gentag
              Læs et tal T
              Sum = Sum + T
              Antal = Antal + 1
        Indtil ikke flere tal i talrækken
        G = Sum/antal
        Udskriv G
STOP
```

Løsning til opg. 6-3

1. fase

```
Start
        Læs alle tal i talrækken
        Sortér talrækken efter stigende værdier
        Udskriv den sorterede talrække
STOP
```

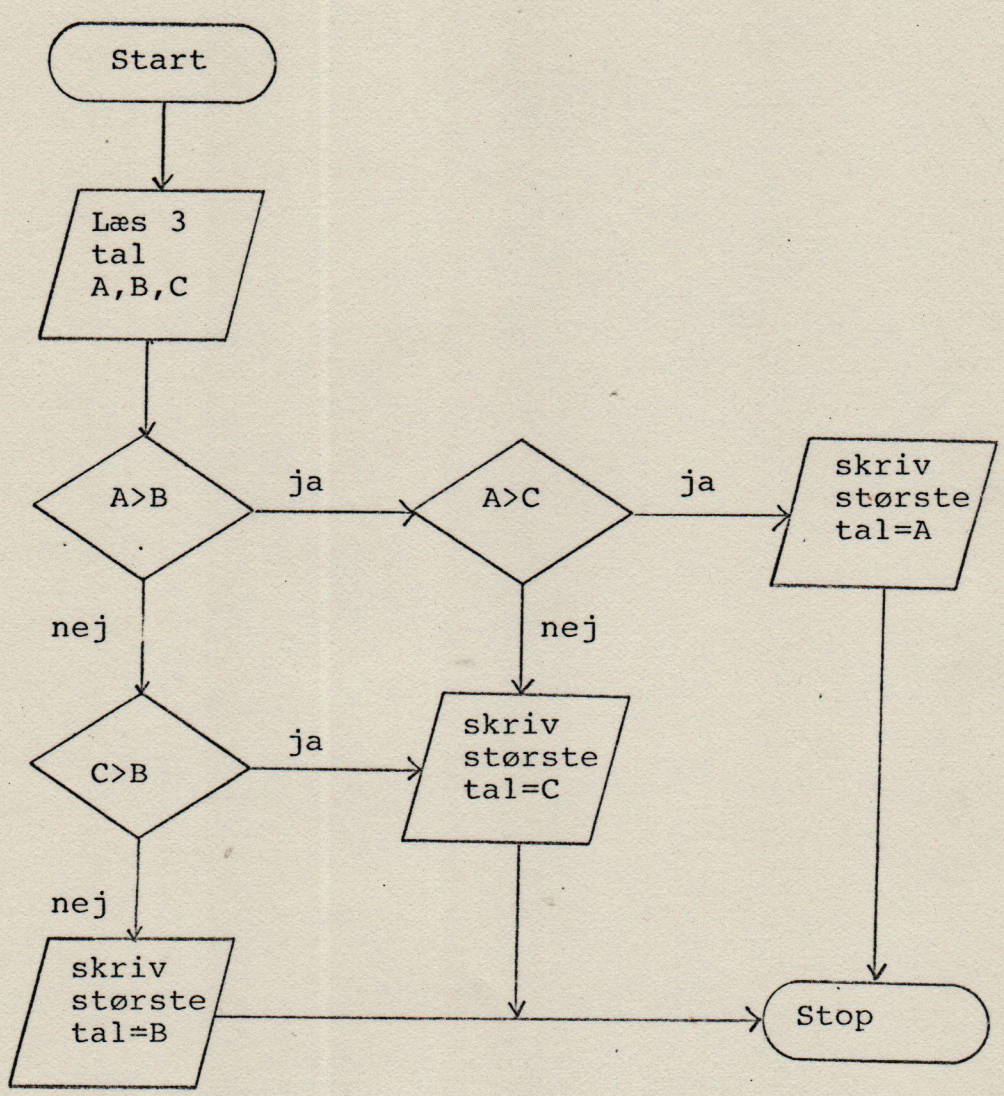
2. fase

```
Start
        Læs alle tal i talrækken T (1..N)
        Gentag
              Find mindste tal MIN i talrækken
              Udskriv MIN
              Slet MIN fra talrækken
        Indtil der ikke er flere tal i talrækken
STOP
```

3. fase

```
Start
        Læs alle tal i talrækken T(1), T(2),...T(antal tal)
        Gentag
              Talnr. = 1; Position = 1
              Min = T(1)
              Gentag
                  Talnr. = Talnr. + 1
                  Hvis T(Talnr) <Min så er Min = T(talnr)
                  og Position = Talnr
              Indtil hele talrækken er gennemløbet
              Udskriv MIN
              Slet MIN fra talrækken (= tal nummer "Position")
        Indtil der ikke er flere tal i talrækken
STOP
```

Løsning til opg. 6-5.



Løsning til opg. 6-6.

A: Hvad skal beregnes?

Om en serie armaturer skal godkendes eller kasseres ud fra en stikprøve-forskrift.

B: Hvad er inddata?

Antal defekte armaturer i stikprøver på 25 stk.

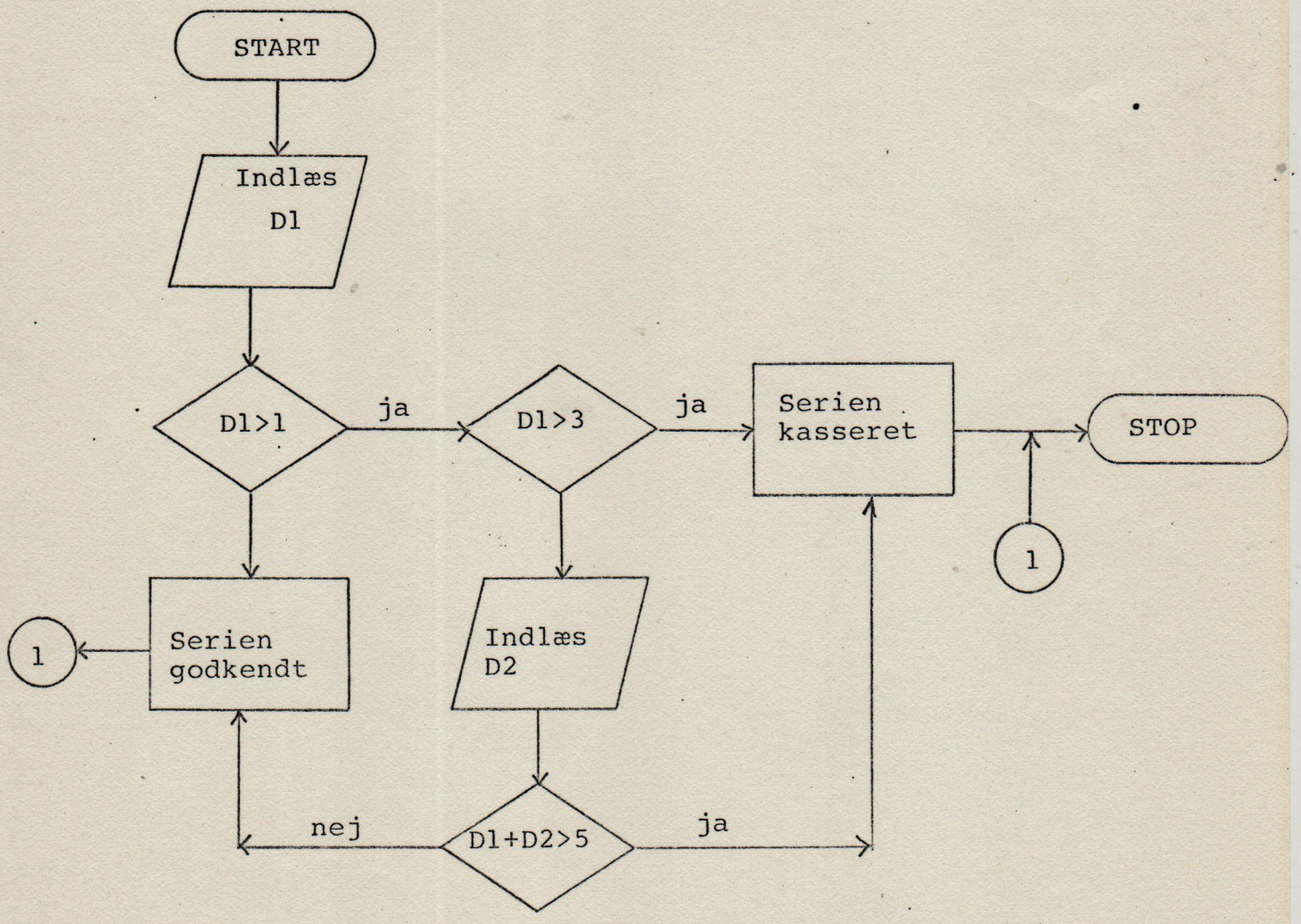
C: Hvad skal udlæses?

Der forlanges ingen udskrifter!

D: Forkortelser

I første stikprøve er antal defekte armaturer = D1

I anden stikprøve er antal defekte armaturer = D2



Løsning til opg. 6-7.

1: Hvad ønskes beregnet?

Forsæbningstallet efter 1/2 og efter 1/1 times kogning (F1 og F2)

Anvendt formel:

$$\text{forsæbningstal} = \frac{\text{ml KOH} * N_{KOH} * 56.11}{\text{g afvejet fedtstof}}$$

hvor 56.11 er molekylvægten for KOH

2: Hvad indlæses?

$N = N_{KOH}$

G1 = g afvejet fedtstof til 1/2 t. kogning

M1 = ml KOH forbrug efter 1/2 t. kogning

G2 = g afvejet fedtstof til 1/1 t. kogning

M2 = ml KOH forbrug efter 1/1 t. kogning

3: Hvad ønskes udskrevet?

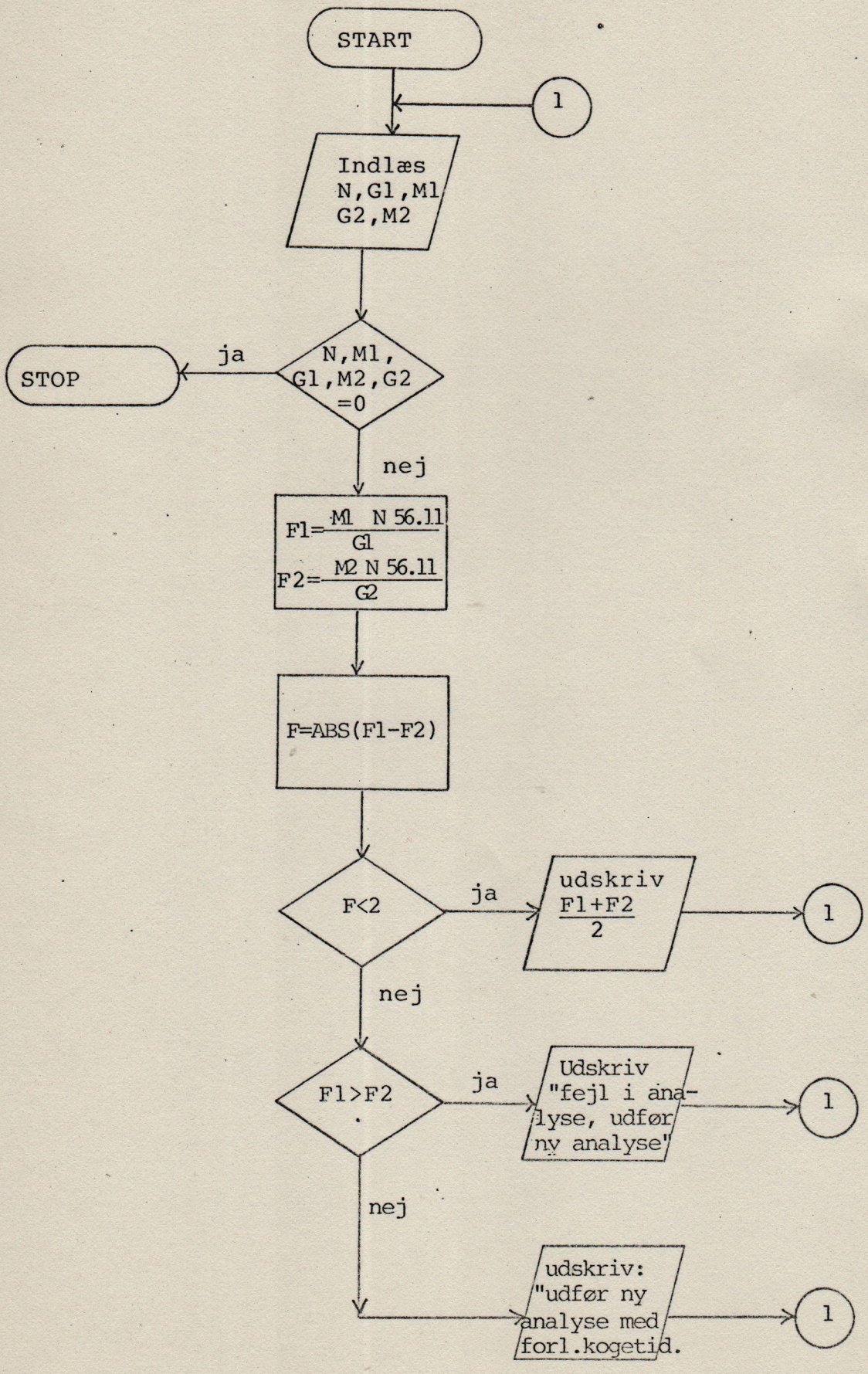
Hvis	Udskrift
$\lvert F1-F2\rvert < 2$	(F1+F2) / 2
$\lvert F1-F2\rvert \geq 2$ og F1>F2	"Fejl i analyse, udfør ny analyse
$\lvert F1-F2\rvert > 2$ og F2>F1	"Udfør ny analyse med forlænget kogetid

4: Som stopdata anvendes 0,0,0,0,0

5: Med de angivne forkortelser fås følgende formel for forsæbningstallet F

$$F = \frac{M * N * 56.11}{G}$$

BLOKDIAGRAM til opg. 6-7.



Løsning til opg. 7-2.

ANALYSE

A) Hvad skal beregnes?

w/w% $FeCl_3$

B) Hvad skal indlæses?

Forbrugt ml $Na_2S_2O_3$ fra de 2 titreringer
normalitet for $Na_2S_2O_3$
gram afvejet analyse

C) Hvad skal udlæses?

w/w% $FeCl_3$
normalitet
ml forbrugt $Na_2S_2O_3$
gram afvejet analyse

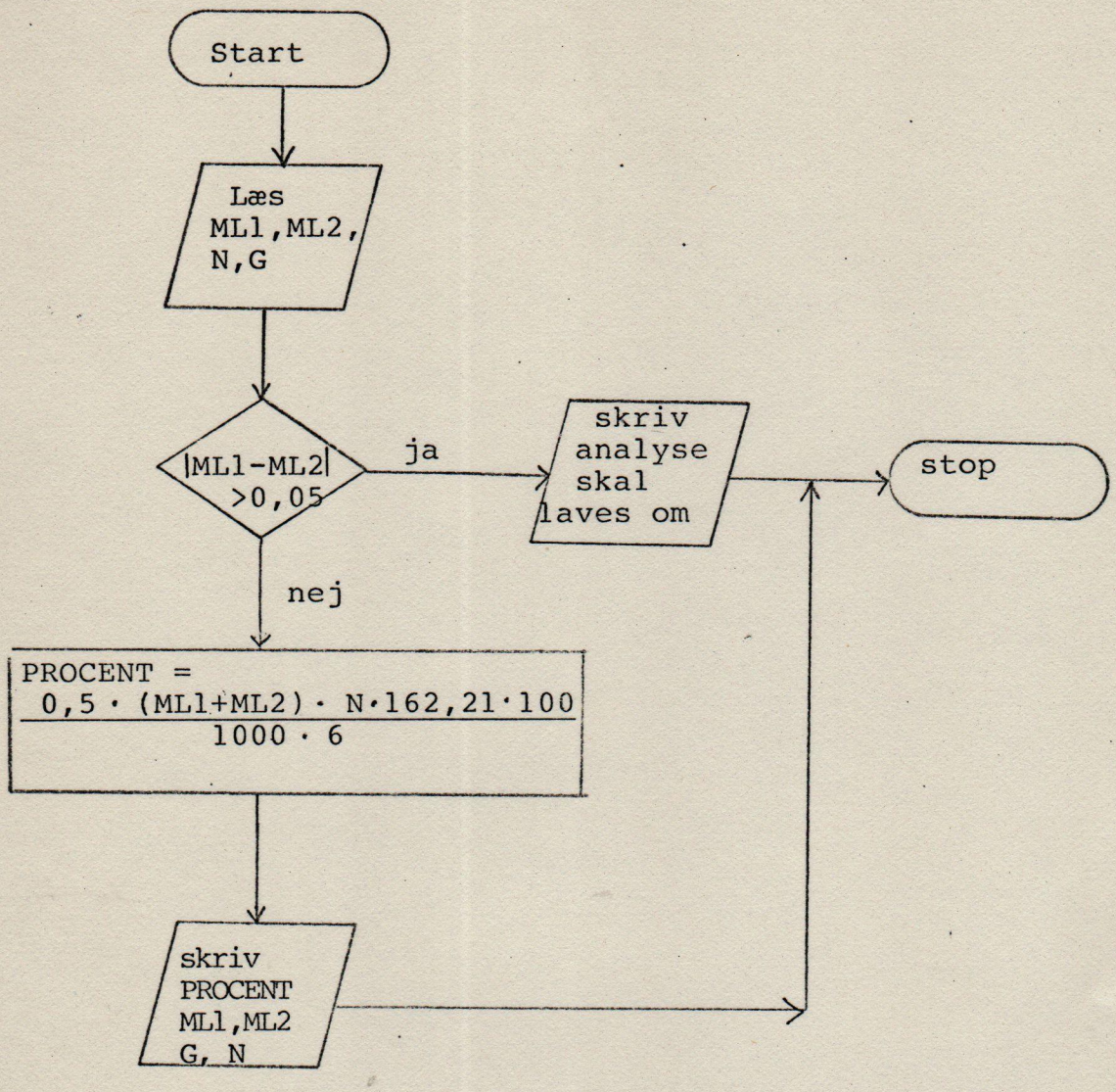
D) Forkortelser

Forbrugt ml $Na_2S_2O_3$	= ML1 og ML2
normalitet for $Na_2S_2O_3$	= N
gram afvejet analyse	= G
w/w% $FeCl_3$	= Procent

E) Indsættelse af forkortelser i beregningsudtryk

$$\text{PROCENT} = \frac{0{,}5\ (\text{ML1} + \text{ML2}) \cdot N \cdot 162{,}21 \cdot 100}{1000 \cdot G}$$

BLOKDIAGRAM




```
0010 PRINT "INDLÆS ML-FORBRUG VED DOBBELTBESTEMMELSEN"
0020 INPUT ML1, ML2
0030 PRINT "INDLÆS NORMALITET OG GRAM ANALYSE"
0040 INPUT N, G
0050 IF ABS(ML1-ML2))0.05 THEN
0060  PRINT "ANALYSEN SKAL LAVES OM"
0070 ELSE
0080  PROCENT:=0.5*(ML1+ML2)*N*162.21*100/(1000*G)
0090  PRINT "VÆGTPROCENTEN ER ";PROCENT
0100  PRINT
0110  PRINT "NÅR DER ER AFVEJET ";G;"GRAM ANALYSE"
0120  PRINT
0130  PRINT "OG FORBRUGT ";ML1;"OG ";ML2;"ML TITRERVÆSKE MED STYRKEN ";N;"N."
0140 ENDIF
0150 END
```

Løsning til opgave 7-3.

ANALYSE

A) Hvad skal indlæses?

Analysens pH-værdi
grænseværdien for pH
tilsat ml titrervæske
ændring af pH-værdi

B) Hvad skal udlæses?

ml forbrugt titrervæske

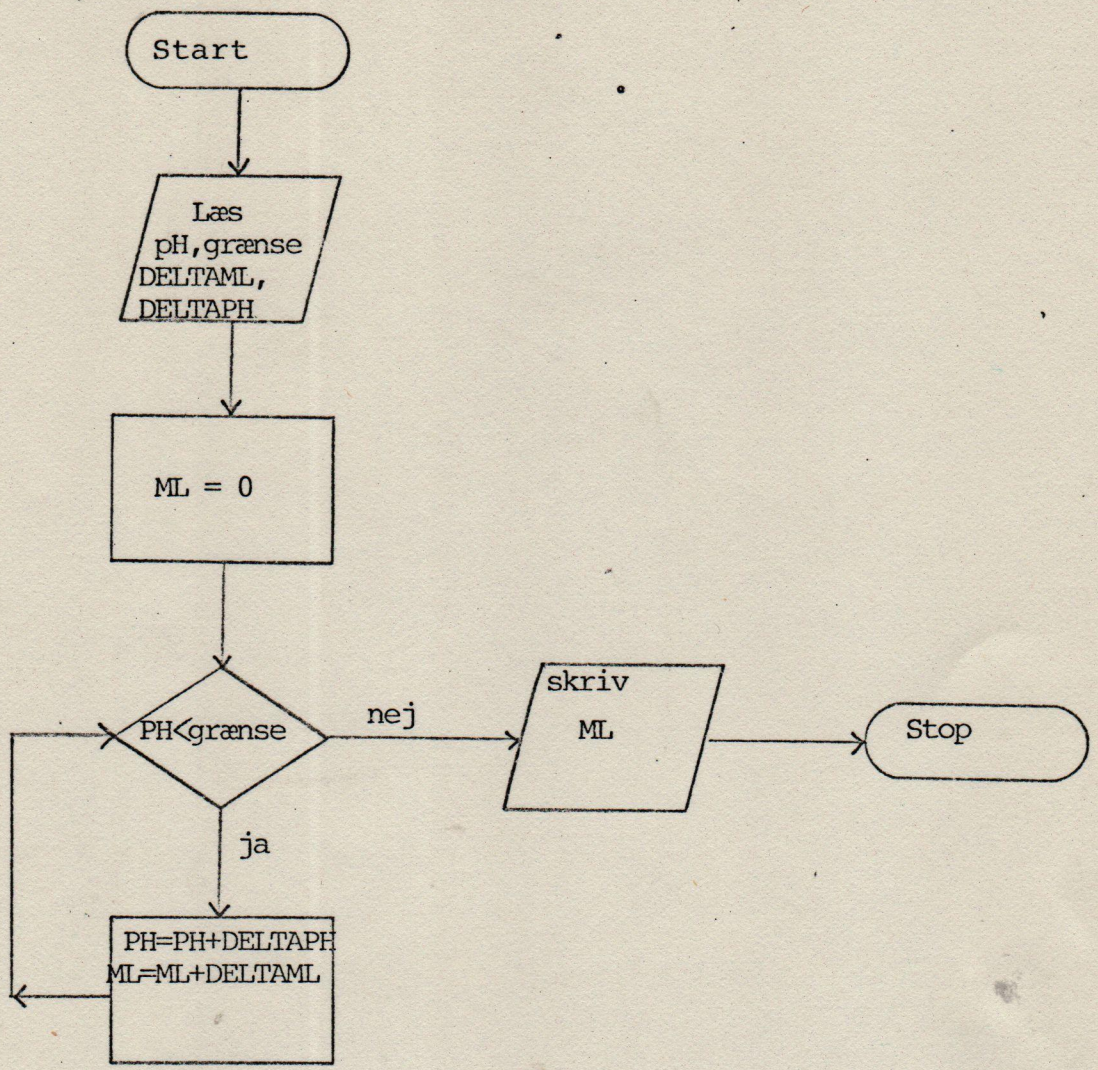
C) Forkortelser

Analysens pH-værdi	= pH
grænseværdi for pH	= GRÆNSE
ml titrervæske	= ML
tilsat ml titrervæske	= DELTAML
ændring af pH-værdi	= DELTAPH

D) Gennemløb flere gange

Tæller for pH: PH = PH + DELTAPH
tæller for ml: ML = ML + DELTAML
nulstilling af ML

BLOKDIAGRAM




```
0010 PRINT "INDLÆS ANALYSENS PH OG GRÆNSEVÆRDIEN FOR PH"
0020 INPUT PH, GRÆNSE
0030 PRINT "INDLÆS TILSAT ML TITRERVÆSKE OG ÆNDRING I PH"
0040 INPUT DELTAML, DELTAPH
0050 ML:=0
0060 WHILE PH<GRÆNSE DO
0070  PH:=PH+DELTAPH
0080  ML:=ML+DELTAML
0090 ENDWHILE
0100 PRINT "DER ER FORBRUGT ";ML;"ML TITRERVÆSKE"
0110 END
```

APPENDIX D

ORDLISTE

ORDLISTE:

ALFABETISKE DATA	Data, hvis tegnsæt kun omfatter bogstaver. EKS.: ETHANOL
ALFAMERISKE DATA	Data, hvis tegnsæt omfatter bogstaver, cifre, specialtegn og blanktegn. EKS.: KRONER: 25,-
ALFANUMERISKE DATA	Data, hvis tegnsæt omfatter bogstaver og cifre. EKS.: PROGRAM24
ARBEJDSLAGER	En del af det indre lager (se dette), hvor datamaten opbevarer indlæste data, gemmer mellemresultater og resultater.
ASCII KODE	American Standard Code for Information Interchange. En kode, hvor der til hvert enkelt tegn svarer en talværdi. Omfatter $128 = 2^7$ tegn (altså 7 bit pr. tegn).
BATCH-KØRSEL GRUPPEKØRSEL	En kørselsform, hvor der sker en ophobning af en større mængde da, herunder programmer, før opgaverne køres. Ved denne kørselsform er det ikke muligt at gribe ind i handlingsforløbet.
BCD KODE	Binary Coded Decimal = binært kodet decimaltal. En kode, hvor hvert enkelt decimale ciffer kodes i totalssystemet. Der bruges 4 bit for hvert decimalt ciffer. EKS.: 69 kodes som 0110`1001
BINÆRT TAL	Et tal i totals-systemet, dvs. bestående af 0 og 1. EKS.: 1101001
BIT	BInary digiT = binært ciffer . Et ciffer i totalssystemet: 0 eller 1.
BLOKDIAGRAM	Et blokdiagram er en beskrivelse af en proces ved hjælp af diagrammeringssymboler. I stedet for betegnelsen blokdiagram bruges ofte en af betegnelserne: DETAIL-BLOKDIAGRAM, PROCES-DIAGRAM eller RUTEDIAGRAM.
BUFFERLAGER	En række LAGERPLADSER, der bruges til at kompensere for forskelle i datas OVERFØRINGSTID eller i tidspunktet for hændelsers indtræden, når data transmitteres fra et apparatur til et andet, og eventuelt også til midlertidigt at LAGRE en kopi af data som en sikkerhedsforanstaltning mod MASKINFEJL. EKS.: Hvis en skærmterminal sender data hurtigere til en linjeskriver end denne kan skrive, må data først overføres til et bufferlager i linjeskriveren, før udskrivning kan påbegyndes.

BYTE	*Et antal "sammenhængende" bit; sædvanligvis 8. En BYTE kaldes ofte også en OKTET.*
CENTRALENHED	*Centralenheden består af REGNEENHEDEN, STYREENHEDEN og LAGERENHEDEN.*
DATABASE / DATABANK	*En samling filer.*
DATAFANGST / KONVERTERING	*Når man oversætter data til maskinlæsbar form, taler man om datafangst. (Der sker en konvertering af data). EKS.: Hulning af hulkort eller direkte indtastning på terminal.*
DATAFØDSEL	*Når man registrerer/nedskriver data første gang, taler man om datafødsel. EKS.: Udfyldning af indbetalingsblanketter i en bank.*
DATAMEDIUM	*Et datamedium er et middel til at repræsentere data på. EKS.: hulkort, magnetbånd og magnetplader.*
DATATRANSMISSION	*Overføring af data ved hjælp af elektriske signaler mellem to eller flere fjerntliggende punkter. Kan f. eks. ske via telefonnettet og/eller via satellit.*
DETAIL-BLOKDIAGRAM	*Som beskrevet under BLOKDIAGRAM, blot er et DETAIL-BLOKDIAGRAM så detaljeret, at det umiddelbart kan bruges som programmeringsforlæg.*
DIREKTE TILGANG	*Et lager med direkte tilgang er et lager, hvor tilgangstiden i det væsentligste er uafhængig af datas fysiske placering. EKS.: pladelager, tromlelager og det INDRE LAGER.*
DISK (-ENHED)	*Ordet DISK anvendes almindeligvis om en pladelagerstation (altså en ENHED). NB: DISK kommer af det engelske ord DISC, der betyder plade, men vi bruger altså ordet DISK om en enhed!!!*
DISKETTE	*En DISKETTE er en mindre, bøjelig og magnetiserbar plade; beskyttet af et paphylster. En DISKETTE er altså et MEDIUM.*
DISKETTESTATION	*DISKETTESTATIONEN er den ENHED, der bruges sammen med DISKETTER. Ofte kaldes en DISKETTESTATION også en FLOPPY-DISK.*
ENHED	*En maskin-enhed, der bruges til at læse/skrive data opbevaret på et DATAMEDIUM. EKS.: Hulkortlæseren læser hulkort, og pladelagerstationen læser/skriver fra/på magnetplader.*

ENSTRENGET INDICERET VARIABEL	*En-dimensional matrix, hvis elementer opsøges ved hjælp af et index, der kan være et tal eller et udtryk.* *EKS.: Registret N(9) er en en-dimensional matrix, der består af 10 pladser: N(0), N(1),..., N(9). Tallene 0,..., 9 kaldes indices.*
FERRITKERNELAGER	*Et lager med DIREKTE TILGANG bestående af en eller flere matricer med FERRITKERNER. Der er direkte tilgang, fordi hver enkelt bit/byte/ord har sin egen adresse. Det INDRE LAGER i en datamat kan være et FERRITKERNELAGER.*
FLOPPY-DISK	*Det samme som en DISKETTESTATION (Se dette).*
HULRIK	*HULkort Regnings Indbetalings Kort.* *EKS.: Girokort med hulkombinationer.*
INDDATA	*Data, der indlæses (ved hjælp af en inddataenhed).*
INDDATAENHED	*En maskin-enhed, der indlæser data til datamaten fra et inddatamedium.* *EKS.: Hulstrimmellæseren indlæser data fra en hulstrimmel.*
INDDATAMEDIE	*Et MEDIUM, hvor data opbevares på maskinlæsbar form.* *EKS.: Hulkort, dokument med MICR-skrift og magnetbånd.*
INDLÆSEENHED	*Se INDDATAENHED.*
INDRE LAGER	*LAGERENHEDEN indbygget i datamatens centralenhed (evt. et ferritkernelager). Det INDRE LAGER omfatter ARBEJDSLAGERET og PROGRAMLAGERET (se disse).*
INPUT	*INPUT betegner inddata til et program.*
K	*1 K = 1024 = 2^{10}. Bruges som forstavelse til byte/ord ligesom k i kg.* *EKS.: 10 K byte = 10·1024 byte.* *Man siger 10 kilo byte ligesom i kilo gram, selv om 1 K ≠ 1000.*
KARAKTER	*Det samme som TEGN. En dårlig "fordanskning" af det engelske ord for TEGN: CHARACTER.* *EKS.: A " ; 7 (4 tegn).*
KONTROLHULNING	*Efter at have hullet en stak hulkort overføres hulkortstakken til en "kontrolhullemaskine", og man indtaster det netop indtastede en gang til. Samtidig kontrollæser maskinen hulkortene. Opstår der nu uoverensstemmelse mellem det indtastede og et hulkort, har man sandsynligvis fanget en fejl. Dog kan det tænkes, at det er kontrolhulningen, der er forkert. Er dette tilfældet, ændres hulkortet selvfølgelig ikke. Kontrolhulningen fanger de fleste fejl, idet man sjældent huller samme fejl to gange.*

KONVERTERING	*Omsætning. At omsætte/omforme data fra repræsentation i et medium til et andet.* *EKS.: Man konverterer fra håndskrevet bilag til hulkort. Man kan også tale om konvertering af data fra hulkort til magnetbånd.* *Taler man om omskrivning af et decimaltal til et binært tal (eller omvendt), bruger man også ordet konvertering.*
LAGERENHED	*En del af datamatens CENTRALENHED. Også kaldet det INDRE LAGER (se dette).* *En lagerenhed kan også betegne en maskin-enhed, der kan læse/skrive på et lagermedium.* *EKS.: Båndstationen kan læse/skrive på magnetbånd. Tromlestationen kan læse/skrive på magnettromlen.*
LIGE PARITET	*Man tilføjer en bit (kontrolbit) til en datarepræsentation, hvis den indeholder et ulige antal 1-taller, for at opretholde et lige antal 1-taller. Ved lige paritet består ethvert bitmønster af et lige antal 1-taller.*
MEDIE	*Ethvert middel til datarepræsentation, herunder materialet, hvorpå data kan repræsenteres.* *EKS.: Hulkort, hulstrimmel, magnetbånd, magnetplade og diskette.*
MAGNETSKRIFT	*Skrift skrevet med magnetisk blæk i et bestemt mønster, så en magnetskriftlæser automatisk kan læse den. (se MICR-skrift).*
MATRIX	*Et udtryk fra matematikken dækkende bestemte n-dimensionale størrelser, der kan manipuleres efter visse regler. Inden for EDB-området anvendes termen matrix normalt om ethvert en- eller flerdimensionalt sæt af elementer af enhver art.* *EKS.: En enstrenget indiceret variabel er en 1-dimensional matrix.* *En ramme bestående af ferritkerner kaldes en 2-dimensional matrix.*
MICR-SKRIFT	*Magnetic Ink Character Recognition.* *EKS.: CMC-7 : Coded Magnetic Character, hvor 7-tallet angiver at ethvert tegn består af 7 streger. Anvendes på check. (se magnetskrift).*
MODEM	*Et apparatur, der MOdulerer og DEModulerer data i forbindelse med datatransmission via telefonnettet. Modulation er ændring fra bitmønstre til tonesignaler (som kan sendes via telefonnettet), og demodulation er den modsatte proces.*
NUMERISKE DATA	*Data, hvis tegnsæt omfatter cifre, decimalpunkt "." og fortegnene "+" og "-". Endvidere omfattes "E" til angivelse af 10-tals eksponenten.* *EKS.: -12.25 +59 3.00 -2.1E+4*

OCR-SKRIFT	*Optical Character Recognition. Optisk læsbar skrift. Skrift skrevet med en bestemt type, så en optisk læser automatisk kan læse den. EKS.: Anvendes på giro-kort.*
OFF-LINE	*Man taler om OFF-LINE databehandling, hvis en enhed indirekte styrer en anden enhed (dvs. med manuel indgriben). Det "modsatte" af ON-LINE.*
ON-LINE	*Man taler om ON-LINE databehandling, hvis en enhed direkte styrer en anden enhed (dvs. uden manuel indgriben). Det "modsatte" af OFF-LINE.*
OPRIK	*OPtisk læsbar Regnings Indbetalings Kort. EKS.: Giro-kort.*
OPTISK LÆSBAR SKRIFT	*Se OCR-SKRIFT og OPRIK.*
OUTPUT	*OUTPUT betegner uddata fra kørsel af et program.*
PARITETSKONTROL	*Kontrol af tegns validitet (gyldighed). Hvis man vedligeholder et lige eller ulige antal bit i et bitmønster, taler man om PARITETSKONTROL. Kontrollen består i at undersøge antallet af 1-taller i et tegns bitmønster (se LIGE og ULIGE PARITET).*
PROCES-DIAGRAM	*Se BLOKDIAGRAM.*
PROGRAMLAGER	*En del af det INDRE LAGER (se dette), hvor datamaten opbevarer programmet.*
REGISTER	*En samling lagerceller (en eller flere). En enstrenget indiceret variabel betegnes ofte som et register. Kan også beskrive en komplet samling data til brug ved en given opgave (EKS.: CPR-registret).*
REGNEENHED	*En del af datamatens CENTRALENHED. I regneenheden foregår den aktuelle databehandling: Addition, subtraktion, division, multiplikation, beregning af aritmetiske udtryk og logiske sammenligninger. Til disse formål har regneenheden et antal registre (enkelt-lagerceller).*
RUTEDIAGRAM	*Se BLOKDIAGRAM.*

SEKVENTIEL TILGANG	*Indirekte tilgang. Man skal søge sekventielt efter ønskede data eller ledig plads at skrive på.* *EKS.: Til data lagret på magnetbånd og hulstrimmel er der sekventiel tilgang.*
SKRIVEBORDSTEST / KASSETEST	*Gennemløb af et program med nogle konkrete værdier og samtidig nedskrivning af samtlige variables indhold i hele programforløbet.*
SPØRGEBOKS	*Et symbol, der illustrerer et logisk valg. Symbolet er en rhombe. I spørgeboksen er der to muligheder: JA eller NEJ.*
STYREENHED	*En del af datamatens CENTRALENHED. Styreenheden henter de enkelte ordrer fra PROGRAMLAGERET og styrer udførelsen af dem.* *EKS.: Styreenheden styrer ind- og udlæsning af data, overførsel mellem ydre LAGERENHEDER og det INDRE LAGER, og beregningerne i REGNEENHEDEN.*
TESTDATA	*Et sæt inddata, der tester om et givent program virker efter hensigten.*
TIDSTRO DATABEHANDLING	*Anvendelse af en datamaskine i forbindelse med et fysisk system, hvor operationerne i datamaskinen finder sted så hurtigt i forhold til forløbene i det fysiske system, at resultaterne af operationerne kan anvendes til styring af det fysiske system (tidstro styring).*
TILGANGSTID	*Tidsrummet fra STYREENHEDEN beordrer en dataoverføring til eller fra lageret, og indtil denne operation er afsluttet.*
TILGANGSTYPE	*Den form for tilgang, der er til et givent lager.* *EKS.: DIREKTE TILGANG (ON-LINE lager):* *kernelager, pladelager og tromlelager.* *INDIREKTE TILGANG (Sekventiel tilgang):* *magnetbånd, hulkort og hulstrimmel.*
32 BIT BYTE	*En BYTE bestående af 32 BIT. En sådan BYTE betragtes som én 'enhed'. (se BIT og BYTE).* *EKS.: 10011110000010101111000010110011*
UDDATA	*Data, der udskrives (ved hjælp af en uddataenhed). Resultatet af en programkørsel.*

UDDATAENHED	*En maskin-enhed, der skriver data fra datamaten på et uddatamedium.* *EKS.: Båndstationen skriver data på magnetbånd.* *Pladelagerstationen skriver data på magnetplader.*
UDDATAMEDIE	*Et MEDIUM, hvorpå data skrives på maskinlæsbar form.* *EKS.: Magnetbånd og hulstrimmel (som begge også er inddatamedier).*
UDLÆSEENHED	*Se UDDATAENHED.*
ULIGE PARITET	*Man tilføjer en bit (kontrolbit) til en datarepræsentation, hvis den indeholder et lige antal 1-taller, for at opretholde et ulige antal 1-taller. Ved ulige paritet består ethvert bitmønster af et ulige antal 1-taller.*
YDRE LAGERENHED	*En lagerenhed, der ikke til stadighed er i forbindelse med CENTRALENHEDEN, men indeholder data på maskinlæsbar form.* *EKS.: Pladelagerstationen og magnetbåndsstationen.*

APPENDIX E

REGNING MED BINÆRE TAL

REGNING MED BINÆRE TAL

Titalsystemet

Det talsystem vi til daglig anvender til beregning er titalsystemet eller decimalsystemet. I titalsystemet anvendes cifrene 0,1,2,... og 9.

I et tal har hvert ciffer en værdi, der bestemmes af dets position i tallet. Et tal som f.eks. 1376 kan udtrykkes på formen:

$$1376 = 1 \cdot 1000 + 3 \cdot 100 + 7 \cdot 10 + 6 \cdot 1$$
$$= 1 \cdot 10^3 + 3 \cdot 10^2 + 7 \cdot 10^1 + 6 \cdot 10^0$$

Betydningen af cifrene afhænger af deres placeringer blandt 1'ere, 10'ere, 100'ere, 1000'ere, o.s.v., altså en potens af 10, således at et ciffers værdi multipliceres med 10 hver gang man går en plads til venstre.

I decimalsystemet er antallet af ciffersymboler 10, som kaldes systemets grundtal.

Totalsystemet

I et talsystem, hvor vi kun har to cifre, 0 og 1, bliver grundtallet 2. Her er betydningen ikke afhængig af tallets placering blandt 1'ere, 10'ere, 100'ere med videre, men derimod af tallets placering blandt 1'ere, 2'ere, 4'ere, 8'ere, 16'ere, o.s.v., altså en potens af 2.

2^0 2^1 2^2 2^3 2^4 2^5 2^6
32 64

Totalsystemet kaldes også det binære talsystem og hvert ciffer i et binært tal kaldes en bit.

Omsætning fra decimaltal til binært tal.

Når man skal omsætte et decimaltal til binært tal, kan man benytte sig af følgende punkter:

1: Hvis decimaltal = 0, gå til punkt 6.
2: Find største potens af 2, der er mindre end eller lig med decimaltallet.

3: Nedskriv den fundne potens af 2.

4: Udfør beregningen:

decimaltal = decimaltal - potens af 2

5: Gå til punkt 1

6: Opskriv den almindelige potensrække.

7: Opskriv det binære tal nedenunder, idet et 1-tal markerer at potensen skal benyttes, og et nul markerer at potensen ikke skal benyttes.

Eksempel:

Omsætningen af decimaltallet 87 til binært tal

87
(64) 2^6
23
23
(16)
7
7
(4)
3
3
(2)
1
1
(1)
0

Benyttede 2-potenser

$2^6 - 2^4 - 2^2 - 2^1 - 2^0$

Almindelig potensrække

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64	32	16	8	4	2	1
1	0	1	0	1	1	1

D.v.s. 87 skrives binært som 1010111.

64 + 16 + 4 + 2 + 1 = 87

Omsætning fra binært tal til decimaltal.

Vi ønsker at omsætte 1001011. Tallet opskrives og nedenunder opskrives, fra højre mod venstre, de tilsvarende potenser af 2. Herefter angiver et 0, at 2-potensen skal udelades og et 1, at 2-potensen skal medtages. Nu kan tallet omskrives

1	0	0	1	0	1	1
64	32	16	8	4	2	1

$1001011 = 1 \cdot 64 + 0 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 75$

Addition af binære tal.

Additionen foregår ligesom i decimalsystemet, d.v.s. når summen bliver lig grundtallet fås resultatet 0 + 1 i mente.

decimaltal: 7 + 3 = 10
binære tal: 1 + 1 = 10

Eksempel:

```
  1101
+  111
------
 10100
------
```

Subtraktion af binære tal.

Den eneste forskel fra decimalsystemet er, at "låneværdien" er 2.

Eksempel:

```
  1101
+  110
------
   111
------
```

Multipliplikation af binære tal.

To binære tal multipliceres på samme måde, som kendes fra decimalsystemet.

Eksempel:

```
  1101 • 101
------------
  1101
 0000x
1101xx
-------
1000001
-------
```

APPENDIX F

KODEFORMER

I dette afsnit vil vi vise de forskellige koder til repræsentation af data, som blev nævnt i forbindelse med gennemgangen af datamedier i afsnit 5.

Fælles for alle koderne er, at de er opbygget af et eller andet bitmønster.

HOLLERITHS SYSTEM

Holleriths system anvendes på hulkortet, som er inddelt i 12 lodrette hullepositioner (se fig. 5.1.), d.v.s. koden bygger på et 12-bitmønster.

De 9 nederste vandrette rækker benyttes til at markere cifrene 1-9, f.eks. markeres cifret 7 med et hul i række nr. 7. Cifret nul markeres i række 10.

Bogstaver kodes med ét hul i rækkerne 10-12, og med et hul i rækkerne 1-9.

A = 12-hul + 1-hul	J = 11-hul + 1-hul	S = 0-hul + 2-hul
B = 12-hul + 2-hul	K = 11-hul + 2-hul	T = 0-hul + 3-hul
C = 12-hul + 3-hul	L = 11-hul + 3-hul	U = 0-hul + 4-hul
D = 12-hul + 4-hul	M = 11-hul + 4-hul	V = 0-hul + 5-hul
E = 12-hul + 5-hul	N = 11-hul + 5-hul	W = 0-hul + 6-hul
F = 12-hul + 6-hul	O = 11-hul + 6-hul	X = 0-hul + 7 hul
G = 12-hul + 7-hul	P = 11-hul + 7-hul	Y = 0-hul + 8-hul
H = 12-hul + 8-hul	O = 11-hul + 8-hul	Z = 0-hul + 9-hul
I = 12-hul + 9-hul	R = 11-hul + 9-hul	Æ = 8-hul + 3-hul
		Ø = 8-hul + 4-hul
		Å = 8-hul + 3-hul + 11-hul

Fig. 7.1. Holleriths bogstavkode.

BCD-TALSYSTEMET.

Når man taler om BCD-koden (Binary Coded Decimal), taler man om et talsystem, der er en mellemting mellem binært og decimal talsystem. BCD står for binærkodet decimaltal og bygger på, at hvert enkelt ciffer i et decimaltal omkodes til binært tal. I alt anvendes fire

binære cifre til at angive en talværdi.

For tallene 0,1,2,.....og 9 har vi:

$0 = 0000_{BCD}$	$5 = 0101_{BCD}$
$1 = 0001_{BCD}$	$6 = 0110_{BCD}$
$2 = 0010_{BCD}$	$7 = 0111_{BCD}$
$3 = 0011_{BCD}$	$8 = 1000_{BCD}$
$4 = 0100_{BCD}$	$9 = 1001_{BCD}$

Et decimaltal kan derefter angives i BCD-notation ved at oversætte ciffer for ciffer. F.eks. har vi:

$$75 = 0111\ \ 0101_{BCD}$$
$$189 = 0001\ \ 1000\ \ 1001_{BCD}$$

BCD-koden anvendes til datarepræsentation bl.a. på hulstrimler og på magnetbånd, hvor den binære repræsentation sker henholdsvis ved hul/ikke-hul og magnetiseret/ikke-magnetiseret plet. Men da data også består af bogstaver og specialtegn, er det nødvendligt at udvide BCD-koden, så alle nødvendige tegn kan blive repræsenteret.

EBCDIC-KODEN

EBCDIC (Extended Binary Coded Decimal Interhange Code) står for udvidet -BCD- udvekslingskode, idet BCD-koden udvides fra 4 bit til mindst 6 bit for at kunne indeholde alfabetet. For eksempel ser bitmønstret for nogle af bogstaverne sådan ud:

A = 110001	J = 100001	S = 010010
B = 110010	K = 100010	T = 010011
I = 111001	R = 101001	Z = 011001

Hvis man ser på de 4 sidste bit og tolker dem v.h.a. BCD-koden gælder, at for A og J svarer de til tallet 1, for B, K og S til tallet 2 og I, R og Z til tallet 9.

Sammenligner man med Holleriths kode i figur F-1, er det let at

drage en parallel, idet A og J her har et 1-hul, B,K og S et 2-hul o.s.v.. Alfabetet er altså blevet inddelt på samme måde som i Holleriths kode, idet de sidste fire bit svarer til tallene fra 1-9.

De 2 første bit i EBCDIC-koden viser også en afspejling af Holleriths kode, idet 12-hul er "oversat" til bitmønstret 11, 11-hul til bitmønstret 10 og 0-hul til bitmønstret 01.

ASCII-koden

En af de mest almindelige og udbredte kodeformer, er den amerikansk opfundne (konstruerede) kode ASCII-koden, der står for American Standard Code for Information Interchange, altså en standard kode der kan benyttes til udveksling af informationer mellem et EDB-systems bestanddele (centralenhed til linieskriver m.v.) eller mellem EDB-systemer. Koden er standardiseret i den forstand, at der til hvert tegn (store og små bogstaver, cifre, specialtegn og styretegn) er tildelt et fast heltal, som de forskellige leverandører af EDB-udstyr så kan indrette deres materiel/programmer efter. På næste side er vist en tabel over denne kode. Tegn med værdier fra 0 til 31 er styretegn, hvormed man styrer de ydre enheder (linieskift, vognretur, komprimeret skrift, linieafstand o.s.v.) . Disse styretegn er normalt ikke synlige, men kan på visse intelligente terminaler m.v., vises som specielt udformede tegn. Navnene på styretegnene ser måske mystiske ud, men f.eks betyder tegn nr. 13 (CR) Carriage Return, altså vognretur. Det er ikke nødvendigvis alle styretegnene der giver mening i et bestemt udstyr.

Tegnet med værdien 32 er en blank, altså et mellemrum. De resterende tegn, er tegn de fleste har stødt på før. I den viste tabel er der til hvert ciffer angivet et decimaltal, der så er omregnet til det oktale talsystem og gengivet under kolonnen oktal (udtales ok'tal). Et oktalt tal er et tal noteret i 8-tals-systemet, der har den fordel, at man hurtigt kan se bitmønstret for et bestemt tegn. Har vi f.eks. tegnet "M" er dets decimalværdi ifølge tabellen 77, der omregnet til 8-tals-systemet giver 115. Ethvert ciffer i dette talsystem kan repræsenteres ved 3-bit, så tallet 115_8 kan hurtigt oversættes til bitmønstret.

```
oktalt:      1   1   5
bit   :    001 001 101
```

Decimal- og oktal-talværdier af ASCII-tegn

COMET mikrodatamaten overholder denne standard, men der er visse af koderne (blandt de specielle fra 0-31) som ikke er anvendt i øjeblikket.

Tegn	Dec.	Oktal	Tegn	Dec.	Oktal	Tegn	Dec.	Oktal	Tegn	Dec.	Oktal
NUL	0	0	blank	32	40	@	64	100	`	96	140
SOH	1	1	!	33	41	A	65	101	a	97	141
STX	2	2	"	34	42	B	66	102	b	98	142
EXT	3	3	#	35	43	C	67	103	c	99	143
EOT	4	4	$	36	44	D	68	104	d	100	144
ENQ	5	5	%	37	45	E	69	105	e	101	145
ACK	6	6	&	38	46	F	70	106	f	102	146
BEL	7	7	'	39	47	G	71	107	g	103	147
BS	8	10	(	40	50	H	72	110	h	104	150
HT	9	11	)	41	51	I	73	111	i	105	151
NL	10	12	*	42	52	J	74	112	j	106	152
VT	11	13	+	43	53	K	75	113	k	107	153
FF	12	14	,	44	54	L	76	114	l	108	154
CR	13	15	-	45	55	M	77	115	m	109	155
MSP	14	16	.	46	56	N	78	116	n	110	156
MNL	15	17	/	47	57	O	79	117	o	111	157
DLE	16	20	0	48	60	P	80	120	p	112	160
DC1	17	21	1	49	61	Q	81	121	q	113	161
DC2	18	22	2	50	62	R	82	122	r	114	162
DC3	19	23	3	51	63	S	83	123	s	115	163
DC4	20	24	4	52	64	T	84	124	t	116	164
NAK	21	25	5	53	65	U	85	125	u	117	165
SYN	22	26	6	54	66	V	86	126	v	118	166
ETB	23	27	7	55	67	W	87	127	w	119	167
CAN	24	30	8	56	70	X	88	130	x	120	170
EM	25	31	9	57	71	Y	89	131	y	121	171
SUB	26	32	:	58	72	Z	90	132	z	122	172
ESC	27	33	;	59	73	Æ	91	133	æ	123	173
FS	28	34	<	60	74	Ø	92	134	ø	124	174
VP	29	35	=	61	75	Å	93	135	å	125	175
RS	30	36	>	62	76	↑	94	136	~	126	176
US	31	37	?	63	77	_	95	137	Del	127	177

RESERVEREDE ORD I COMAL-80

		sætning	Kommando	Funktion
ABS(x)	Beregner den numeriske værdi af x (|x|)			x
AND	Logisk og			(x)
ATN(x)	Beregner arcustangens til x.	x		
AUTO	Starter den automatiske linienummerering		x	
BSTR$(x)	Omformer et tal (0..255) til binær tegn repr.			x
BVAL(x$)	Omformer binær tegnstreng til heltal			x
CALL	Kald af maskinsprogs-program	x		
CASE	Bruges i konstruktionen CASE-WHEN-OTHERWISE-ENDCASE, der på baggrund af et udtryks værdi udvælger et program-afsnit			
CAT	Udskriver kataloget over de på et baggrundslager værende filer	x	x	
CHAIN	Henter et program af typen xxxx. CSB fra baggrundslageret og udfører det	x		
CHR$(x)	Omformer værdien af x (0-255) til et ASCII-tegn			x
CLEAR	Renser skærmen for tekst. program bevares!	x	x	
CLOSE	Lukker en åben fil	x	x	
CON	Fortsætter en afbrudt programafvikling		x	
COS(x)	Beregner cosinus til x (x i radianer).			x
CURSOR	Positionerer cursoren (det lysende kvadrat) på skærmen	x	x	
DATA	Angiver inddata til brug for READ-sætningen	x		
DEF	Definer en brugerfunktion (skal begynde med FN)	x		
DEL	Sletter en eller flere programlinier		x	
DELETE	Sletter en fil fra baggrundslageret	x	x	
DIM	Dimensionerer størrelsen af variable	x		
DIV	Heltalsdivision			(x)
EDIT	Editerer (retter) i eksisterende pro-gramlinier		x	
END	Afslutter ethvert program	x		
ENTER	Henter en programfil af typen xxxx.CML over i primærlageret		x	

		Sætning	Kommando	Funktion
EOD	Anvendes til at afgøre om allé data fra en DATA-sætning er læst (EOD≙ End Of Data)			(x)
EOF(n)	Anvendes til at afgøre om alle data fra en sekventiel fil er læst (EOF= End Of File).			(x)
ERR	Anvedes til at afgøre om der under programafviklingen opstår en ikke fatal fejl.			(x)
ERRTEXT$(x)	Udskriver systemets fejltekst nr. x.			x
ESC	Anvendes til at afgøre om tasten [ESC] nedtrykkes			(x)
EXEC	Udfører en underprogramdel (procedure) EXEC = Execute	x		
EXP(x)	Beregner e (= 2.7183) opløftet til potensen x.			x
FALSE	Anvendes til at give en varibel værdien "Falsk"			(x)
FOR	Bruges i konstruktionen FOR-TO/DOWTO-STEP-NEXT, der aktiverer programsætningerne mellem FOR og NEXT et antal gange bestemt af styrevariabel og stopværdi	x		
FRAC(x)	Finder decimaldelen af det reelle tal x			x
GETUNIT	Finder den aktuelle magnetiske baggrundsenhed	x	x	
GLOBAL	Anvendes til at gøre variable i hovedprogrammet tilgængelige i en lukket procedure	x		
GOSUB	Kalder et underprogram defineret ved et linienummer og et RETURN	x		
GOTO	Ubetinget hop	x		
IF	Bruges i konstruktionerne IF-THEN, IF-THEN-ELSE-ENDIF og IF-THEN-ELIF-ELIF-ELSE-ENDIF. Udfører eller overspringer sætninger afhængig af logisk værdi (TRUE,FALSE).	x		

		Sætning	Kommando	Funktion
IN	Anvendes til at afgøre om en tekst forekommer i en anden			(x)
INIT	Initialiserer en nyindsat diskette. D.v.s. gør disketten kendt for systemet.	x	x	
INP(x)	Anvendes til at aflæse port nr. x (0..255) Maskinsprogsfunktion).			x
INPUT	Lægger én eller flere fra tastaturet indtastede værdier over i brugerdefinerede variable.	x		
INPUT-FILE	Lægger en eller flere fra en fil indlæste værdier over i brugerdefiderede variable.	x		
INT(x)	Finder det største heltal som er mindre end eller lig med x.			x
IVAL(X$)	Anvendes til at omsætte texten X$ der er en tegnepræsentation af et heltal, til et heltal.			x
LEN(x$)	Anvendes til at finde antallet af tegn i textvariablen x$.			x
LIST	Udskriver hele eller dele af programmet i primærlageret på skærm, linieskriver eller diskette (xxxx.CML).		x	
LOAD	Indlæser et program af typen xxxx.CSB fra disketten til primærlageret. Der er altså tale om et program der er overført til disketten med kommandoen SAVE.		x	
LOG(X)	Anvendes til at beregne den naturlige logaritme af X (ln(X))			x
LOOP	Bruges i konstruktionen LOOP-EXIT-ENDLOOP hvor sætningerne mellem LOOP og END-LOOP udføres et af brugeren bestemt antal gange (betinget sætning med EXIT)	x		
MAT	Tildeler ethvert element i en indiceret variabel den samme værdi (f.eks. 0)	x		
MOD	Finder resten ved heltalsdivision. Resten er et heltal af typen heltal eller reelt tal.			(x)

		Sætning	Kommando	Funktion
NEW	Sletter programmet i primærlageret		x	
NOT	Logisk negation.			(x)
OPEN-	FILE. Åbner en fil (kartotek) for læs-			
	ning (READ), skrivning (WRITE) eller			
	begge dele (RANDOM).	x	x	
OR	Logisk eller			(x)
ORD(X$)	Anvendes til at omsætte et tegn X$			
	til dets ASCII-værdi (0..255)			x
OUT(x)	Anvendes til at sende en værdi (0..255)			
	til en port (0..255). (Maskinsprogs-			
	funktion)	x	x	
PAGE	Fører papiret i printeren frem til			
	toppen af næste side	x	x	
PEEK	Anvendes til at bestemme indholdet på			
	en adresse i primærlageret (maskin-			
	sprogsfunktion)			x
POKE	Anvendes til at lagre en værdi på en			
	adresse i primærlageret (maskinsprogs-			
	funktion).	x	x	
POS(X$,Y$)	Anvendes til at beregne hvor mange			
	positioner (regnet i tegn) inde i teks-			
	ten Y$ teksten X$ starter.			x
PRINT	Udskriver data på et uddatamedie	x	x	
PRINT-	FILE. Udskriver data i ASCII-format på			
	en fil.	x		
PRINT-	USING. Udskriver data på et uddatamedie			
	styret af en format-sætning	x		
PROC	Bruges i konstruktionen PROC-ENDPROC,			
	der definerer en procedure (et under-			
	program der senere udføres med			
	ordren EXEC.	x		
QUIT	Afslutter Comal-80, og der vendes til-			
	bage til CP/M-styresystemet	x	x	
RANDOM	Giver den indbyggede tilfældigtals-			
	generator en "tilfældig" startværdi,			
	til brug for funktionen RND.	x	x	
READ	Læser data fra en DATA-sætning over i			
	brugerdefinerede variable.	x		

		Sætning	Kommando	Funktion
READ	Læser data på binær form fra en fil	x		
RELEASE	Undersøger om alle filer på baggrundslageret er lukkede	x	x	
REM	Benyttes i programmer hvor orienterende bemærkninger ønskes indsat. Kan erstattes af // eller !	x		
RENAME	Omdøber et gammelt filnavn til et nyt af brugeren specificeret filnavn.	x	x	
RENUM	Omnummererer et programs linienumre, eller flytter et antal programlinier.		x	
REPEAT	Bruges i konstruktionen REPEAT-UNTIL, der aktiverer programsætningerne mellem REPEAT og UNTIL, indtil betingelsen angivet efter UNTIL er opfyldt.	x		
RESTORE	"Spoler" tilbage til starten af en DATA-sætning.	x		
RND	Benyttes til at finde et pseudotilfældigt tal i et af brugeren defineret interval.			x
ROUND(x)	Afrunder et reelt tal til nærmeste heltal.			x
RUN	Starter eksekveringen af programmet i primærlageret.		x	
SAVE	Udskriver programmet i primærlageret på disketten under et af brugeren angivent navn (xxxx.CSB).		x	
SELECT-OUTPUT	Omdirigerer uddata fra PRINT-sætninger til den af brugeren angivne enhed (DS:, LP:,···).	x	x	
SGN(x)	Bestemmer fortegnet for X.			x
SIN(x)	Beregner sinus til x (x i radianer).			x
SIZE	Angiver brugt lagerplads og resterende lagerplads i primærlageret.		x	
SPC$(x)	Danner en tekst bestående af X blanke.			x
SQR(X)	Beregner kvadratroden af X (X ≥ 0).			x
STOP	Stopper programafviklingen med en angivelse af, hvor programmet stoppede	x		

		Sætning	Kommando	Funktion
STR$(X)	Omsætter værdien af X til en tekst (ASCII-repræsentation).			x
TAB	Tabulator indstilling	x	x	
TAB(X)	Positionerer skrivehoved/cursor i positionen X.			x
TAN	Beregner tangens til X (X i radianer).			x
TRAP-	ERR. Ændrer fortolkerens reaktion på nedtrykning af [ESC] tasten.	x	x	
TRUE	Anvendes til at give en variabel værdien "sand".			(x)
TRUNC(X)	Afrunder et reelt tal X til et heltal ved bortkastning af decimalerne.			x
UNIT	Udpeger en baggrundsenhed som standardenhed.	x	x	
VAL(X$)	Omsætter tegnrepræsentationen X$ af et reelt tal til et reelt tal.			x
VARPTR(X)	Beregner variablen X's absolutte adresse i primærlageret (maskinsprogsfunktion).			x
WHILE	Bruges i konstruktionen WHILE-ENDWHILE, der aktiverer programsætningerne mellem WHILE og ENDWHILE sålænge betingelsen efter WHILE er opfyldt.	x		
WRITE-	FILE. Udskriver data i binær form på en fil	x		