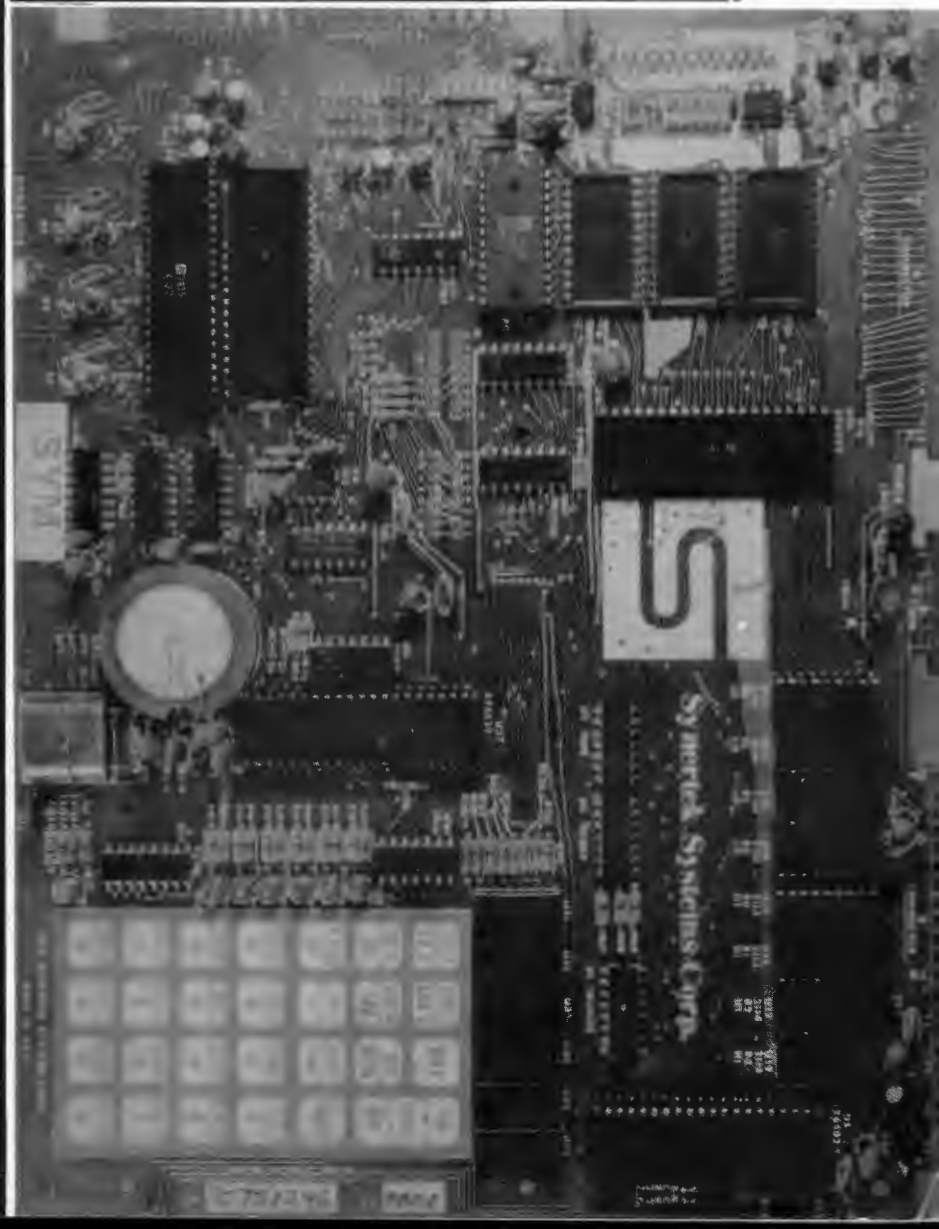
12 Håndbog for datamat-amatører

HCC 1978



INDHOLDSFORTEGNELSE

ALMENT OM PROGRAMMERBARE MASKINER

Sådan begyndte det	A	1
Den forventede udvikling	A	7
Talsystemer	A	21
Binær matematik	A	28
Logiske operationer	A	31

BIBLIOTEKET – PROGRAMMER

HP-25, Delefiltre	B	1
HP-25, Gæt et tal	B	3
HP-25, Likviditet	B	5
HP-25, Mastermind	B	11
KIM-1, Multimaze	B	13
IMSAI 8080, RAM-test	B	17
KIM-1, FUT-FUT	B	19
TI-59, Løn og skat	B	21
TI-59, Kørselsregnskab	B	27
BASIC, Lån, afdrag og renter	B	31
BASIC, Reaktionstid	B	37
TI-58/59, Omregn. mell. talsyst.	B	41
TI-58/59, Mastermind	B	43
SR-52, Sænke slagskibe	B	45
mek6800D2, MUSIC	B	49
PÅSKE	B	51
Rennummereringsprogram	**B**	**53**
LIFE	**B**	**54**

CPU-ARKITEKTUR

CPU-Arkitektur	C	1
Motorola M6800	C	5
Intel 8080	C	7
SC/MP	C	9
Signetics 2650	C	11
Intel 8048	C	13
Zilog Z-80	C	15

DATAMAT-LITTERATUR

Elementært om Microdatorer	D	1
The first Book of KIM	D	2

KLUBINFORMATION

Datamatklubber	K	1
Datamatamatører	K	11
Seminar, april 1978	K	17

LOMMEREGNERE

TI-Programmer	L	1
HP-25/25C	L	3
TI-59/PC-100	L	5

MIKRODATAMATER

Valg af mikrodatamat	M	1
Datamatkapacitet	M	5
KIM-1	M	11
KIM-1, Kontakter og dioder	M	15
Motorola M6800	M	19
TK-80, begyndersæt	M	25
Imsai 8048 CC	M	30
Nye datamater, april 1978	M	33
PET 2001	M	37
TELMAC 1800	M	41
KIM-1, udviddet udgave	M	43
ABC 80	M	45
SYM-1	**M**	**51**

PROGRAMMERINGSTEKNIK

Lær programmering	P	1
Subrutiner	P	49
Splitning af subrutiner	P	51
BASIC	P	55
BASIC, fortsat	**P**	

SELVBYGGERPROJECTER

IMSAI 8080	S	1
Z-80 mikrodatamat	S	11
77-68 selvbyggerdatamat	S	51
Z-80, fortsat	S	57
Z-80, fortsat	**S**	

YDRE ENHEDER

TV-skriver	Y	11
Pocket TTY	Y	13
TV-modulator	Y	16
Digital multiplexer	Y	19
ACT-IV skærmterminal	Y	25

RETTELSE

Der havde indsneget sig et par fejl i programmet LIFE fra forrige nummer af HFD, side B54. Vi bringer derfor denne gang en rettet udgave af dette blad, som bedes indsat i mappen istedet for forrige måneds udgave. Vi beklager.

Håndbog for datamat-amatører udgives i løsbladsformat af Telepress ApS, Greve Strandvej 42, 2670 Greve Strand. Tlf. (02) 90 86 00. Giro nr. 1 15 53 69. Tryk: Fraling Offset, Viby Sj. HFD udsendes til abonnenter som trykksag d. 1. torsdag hver måned. 1. nummer udgivet er nr. 9/1977. Et årsabonnement koster kr. 100,– incl. ringbind og porto. Ansvarshavende udgiver: H. Lind.

ISSN 0105–581X

Alment om programmerbare maskiner	A
Biblioteket - programmer	B
CPU-arkitektur	C
Datamat-litteratur	D
Interfacing	I
Klubinformation	K
Lommeregnere	L
Microdatamater	M
Programmeringsteknik	P
Selvbyggerprojekter	S
Tilbud fra læserne	T
Undervisningsudstyr	U
Ydre enheder	Y

Microcomputer KIM 1

NY REDUCERET PRIS NU KUN KR 1669,-.

Microcomputer KIM-1, komplet med user manual, systemdiagram, programmeringsreferencekort, programmeringsmanual og hardware manual incl. moms . . . . . ~~nu kr. 2.100,=~~
nu kr. 1.669,–

"The first book of KIM" på 176 sider, indeholdende afsnit for begynderen, underholdende og praktiske programmer, information om interfacing og avanceret brug etc. . . . . **kr. 75,–**

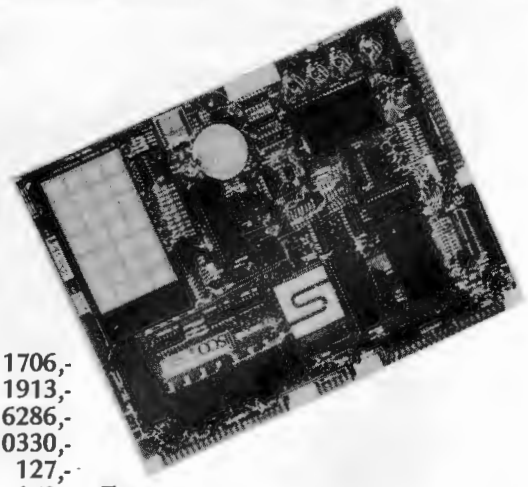
INSTRUTEK

Hovedkontor:
Christiansholmsgade
8700 Horsens
Tlf. 05 - 61 11 00

Øst:
Rødovrevej 155
2610 Rødovre
Tlf. 01 - 41 34 00

DET SYMPATISKE SYSTEM

SYM-1



Samlet og afprøvet incl. demo tape og dansk vejledning.
Kr. 2395,- incl. moms.
Let at anvende - let at udvide.
Kræver kun 5v-1,5a.

NY PRIS Kim 1	1706,-
NY PRIS Kim 1 incl. kabinet	1913,-
Feltron TTY	6286,-
T E C 502, dataskærm	10330,-
Tiny Basic	127,-
FCL 65, incl. mini manual	178,-
Spændingsforsyning fra	325,-
Båndoptager Philips N2214	566,-

Alle priser er incl. 20,25 % moms.

lisco

Microdata
Aprilvænget 6
6000 Kolding

Renummereringsprogram

Dette er et renummereringsprogram til en PET computer, det må frarådes på det kraftigste at bruge den på andre maskiner, det vil helt sikker forårsage alle mulige og umulige ulykker!
Programmet er et Basic program med høje linienumre, og det er beregnet til at "ligge oven i" et program man er i færd med at indtaste, så man man med mellemrum renummerere med komandoen RUN 3000. Hvis man er bange for at miste overblikket over et program med mange hopordrer, så kan man indsætte det gamle linienummer i de linier der skal hoppes til.
Dette kan f.eks. gøres således: 467 ——— kode ———: REM 467 så kan man finde linien igen efter omnummereringen.
Ide'en bag programmet er at når PET computeren lagrer et program, så vil der før hver linie være 5 såkaldte "overhead bytes". Den første af disse bytes indeholder et nul, det fortæller computeren at en ny linie begynder, de to næste bytes fortæller hvor lang linien er, og de to sidste bytes indeholder linienummeret. Det er disse to bytes som programmet udskifter.
Linie 29990: Hvis man prøvekører et program man er i færd med at indtaste, og programmet ikke er forsynet med en END sætning, så vil denne linie forhindre at man falder igennem til renummereringsrutinen.
Linie 30020: A og B opbevarer linienumrene.
Linie 30030: Her starter en løkke der gennemgår maskinens hukommelse. Hvis man hat mere end 8 kilobyte må det sidste tal gøres større.
Linie 30040: Denne linie undersøger om byte'n indeholder et nul, hvis ikke går programmet videre til næste byte.
Linie 30030: Her fastlægges linienummeret, A er lig med den lave byte og B er lig med den høje.
Linie 30060: Denne linie forhinder at programmet renummererer sig selv, når programmet møder en linie med nummeret 29990, så standser det og computeren udskriver BREAK IN 30060, dette tilkendegiver at renummereringen er færdig.
Linie 30070: Her sker så omnummereringen, læg mærke til at løkkevariablen I bliver forøget med 4, dette er nødvendigt fordi der kan godt befinde sig et nul i en af de bytes det indeholder linienummeret, og hvis programmet ikke hopper uden om dem vil det tro at det er starten på en ny linie.

```
29990 GOTO 30090
30000 REM ——— RENUMMERERINGS RUTINE ———
30010 REM ——— INDTAST  RUN 30000 ———
30020 A=0:B=0
30030 FOR I=1024 TO 8200
30040 IF PEEK(I)<>0 THEN NEXT I
30050 A=S+10:IF A 255 THEN B=B+1:A=A—256
30060 IF PEEK(I+3)=38 AND PEEK(I+4)=11 THEN STOP
30070 POKE I+3,A:POKE I+4,B:I=I+4
30080 NEXT I
30090 END
```

Program: LIFE

Dette er et LIFE program. Programmet er beregnet til en PET computer, men vil muligvis kunne bringes til at køre på andre maskiner hvis komandoerne PEEK og POKE forefindes og maskinen har en VDU der er memory-mapped.

Linie 80: Den første løkke på linien er en forsinkelsesløkke, så man kan nå at læse instruktionen i brugen af programmet. Dernæst kommer en løkke der vil udskrive et raster på skærmen så man kan placere sin figur rigtigt. Dette raster vil forsvinde når første generation vises.

Linie 90: AS er en dummy variabel, INPUT ordren bruges udelukkende til at starte programmet efter man har indtastet sin figur.

Linie 100 til 120: Her indlæses der data fra datasætningerne, af hensyn til at programmet skal køre så hurtigt som muligt er alle konstanter udskiftet med variable. Det skal bemærkes at hvis man ønsker sorte stjerner på hvid baggrund, så kan man udskifte 32 og 42 med 160 og 170, så skal man blot huske at trykke RVS-knappen før man indtaster sin figur.

Linie 130: Dette er en løkke der rydder en kilobyte af hukommelsen, i hver byte optælles hvor mange naboer den tilhørende celle har.

Linie 140 til 260: Dette sker i løkken der begynder i linie 140, hvis en celle er beboet så vil dens 8 naboer hver få et point. Hvis cellen er ubeboet går programmet videre til den næste celle.

Linie 270 til 330: Dette er udskrivningsrutinen, udskrivningen sker efter John Conway's genetiske regler: Overlevelse: Hvis en beboet celle har 2 eller 3 naboer, så vil cellen være beboet også i næste generation.

Død: Hvis en beboet celle har færre end 2 eller flere end 3 naboer, så vil cellen være ubeboet i næste generation.

Fødsel: Hvis en ubeboet celle har 3 naboer, så vil cellen være beboet i næste generation.

Linie 330 (og linie 130 og 260): Læg mærke til at i 3 af løkkerne mangler løkkevariablen I i NEXT-sætningen, dette får løkkerne til køre hurtigere.

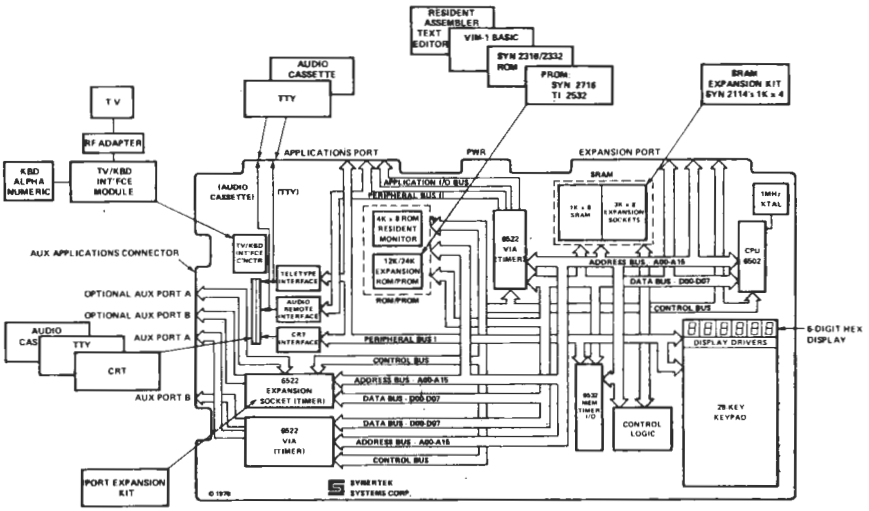
Linie 340: Her udskrives generationens nummer, CHRS(19) er en "home cursor" karakter.

```
 5 OPEN 4,4,0:CMD 4:LIST
 10 PRINT CHR$(147):PRINT TAB(13)"***LIFE***":PRINT:PRINT
 20 PRINT"DETTE PROGRAM SIMULERER JOHN CONWAY'S":PRINT
 30 PRINT TAB(11)"*GAME OF LIFE*":PRINT
 40 PRINT"PROGRAMMET STARTES VED AT FLYTTE":PRINT
 50 PRINT"CURSOREN MED CURSORKONTROLLERNE OG AN-":PRINT
 60 PRINT"BRINGE EN STJERNE (*) DE ONSKEDE STEDER":PRINT
 70 PRINT"DEREFTER TRYKKES ";
 71 PRINT CHR$(18)"RETURN";
 72 PRINT CHR$(146)
 80 FOR I=1 TO 6000:NEXT I:FOR I=32768 TO 33767:POKE I,46:NEXT I
 90 INPUT A$:A$=" "
 100 DATA 0,1,2,3,32,42,29809,29808,29807,29769,29768
 110 DATA 29767,29729,29728,29727,32808,33727,0
 120 READ A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,QR,S
 130 FOR I=2998 TO 4002:POKE I,A:NEXT
 140 FOR I=32768 TO 33767
 150 IF PEEK(I)<>F THEN 260
 160 IF I<Q THEN 200
 170 POKE I-G,PEEK(I-G)+B
 180 POKE I-H,PEEK(I-H)+B
 190 POKE I-J,PEEK(I-J)+B
 200 POKE I-K,PEEK(I-K)+B
 210 POKE I-M,PEEK(I-M)+B
 220 IF I>R THEN 260
 230 POKE I-N,PEEK(I-N)+B
 240 POKE I-O,PEEK(I-O)+B
 250 POKE I-P,PEEK(I-P)+B
 260 NEXT
 270 FOR I=32768 TO 33767
 280 IF PEEK(I)<>F THEN 310
 290 IF PEEK(I-L)=D OR PEEK(I-L)=C THEN 330
 300 GOTO 320
 310 IF PEEK(I-L)=D THEN POKE I,F:GOTO 330
 320 POKE I,E
 330 NEXT
 340 S=S+B:PRINT CHR$(19);S:GOTO 130
READY.
```

HFD december 1978

SYM-1

KIM-1 har efterhånden et par år på bagen, og selvom det er forsøgt at gøre den tidssvarende gennem diverse ekstra kort og forbedret software, har det været ventet, at en eller anden ville gøre brug af de nyeste kredse og bringe en opdateret udgave baseret på den kendte MOS 6502, Det blev Synertek, som med SYM-1 indfriede vore forventninger.



SYM-1 er ikke en afløser for KIM -1 er næsten tværtimod opstået som følge af KIM-1's store popularitet.
Hver gang et firma har succes med et produkt, vil et andet forsøge at gøre dem kunsten efter, og hvis det er muligt, så gerne lidt bedre. I dette tilfælde har tiden været med Synertek, som ikke alene har kunnet benytte sig af den nyeste teknoligi, men som følge af bekendtskabet med KIM -1 har erfaret, hvor en del af skoen måske trykker. Resultatet er blevet en datamat, som på mange punkter er identisk med KIM-1, men alligevel på flere drastiske punkter adskeller sig fra denne.

DEN MEKANISKE OPBYGNING

SYM-1 er opbygget i samme format som KIM-1 og er altså også en datamat på et enkelt kort. Rent fysisk er SYM-1 drejet en kvart omgang, så kortet vender bredsiden til, når det betjenes.
Den største umiddelbare forskel er tastaturet, som på SYM-1 består af

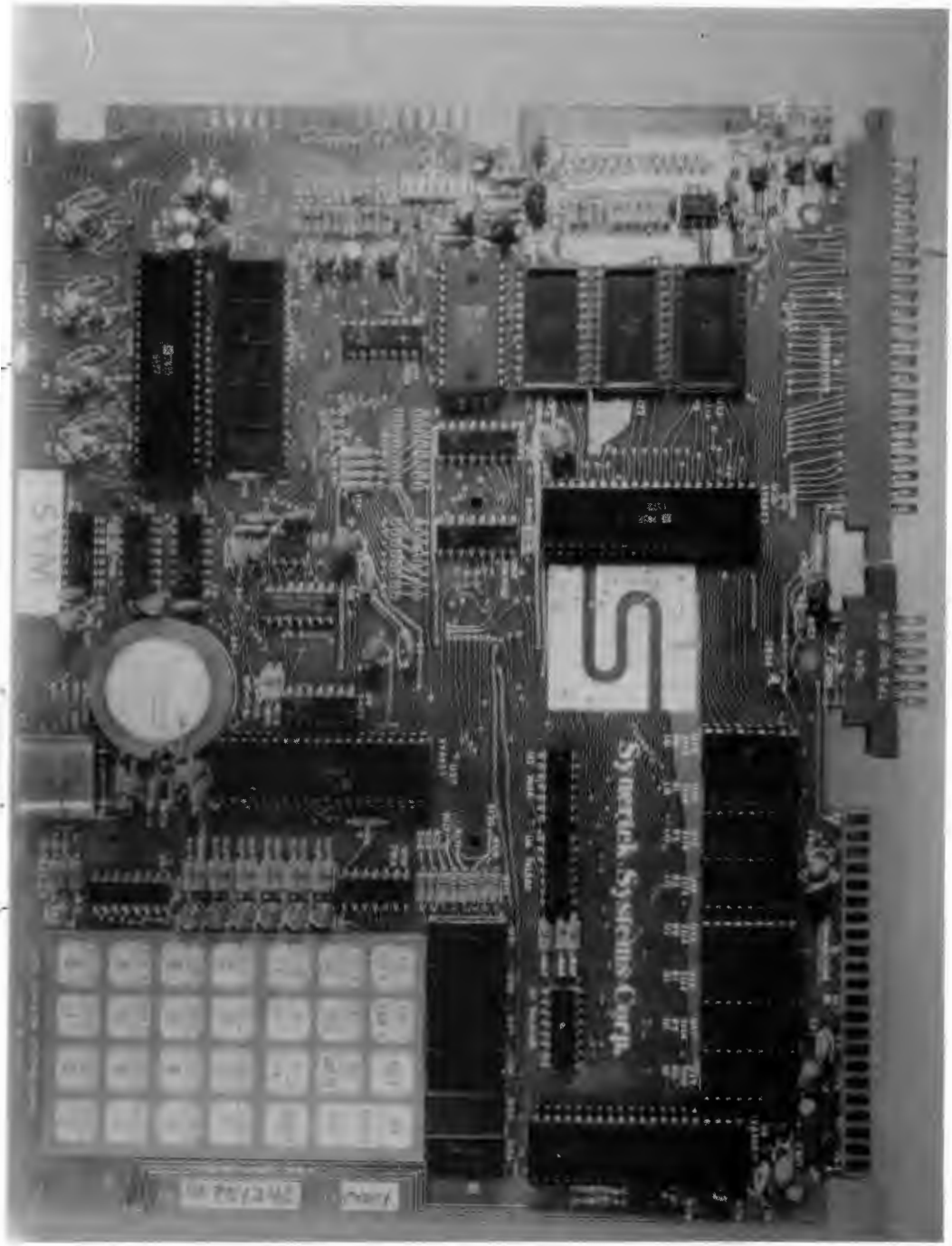
ikke færre end 28 taster. hvoraf hovedparten har to funktioner. Disse skyldes den avancerede monotor, SUPERMON, som gør betjeningen af datamaten endnu lettere, end tilfældet er med KIM-1.
SYM-1 kan også give lyd fra sig. Der er indbygget en "beeper", som gør indtastningen mere sikker, idet et "beep" kvitterer for indtastet information. Dette er en stor fordel, idet tasterne ikke selv indikerer for det nødvendige tryk. SYM-1 er forsynet med ialt 4 kantkonnektorer for tilslutning af spænding og ydre enheder som f. eks. 20 mA current-loop TTY, kassettebåndoptageren m.m.
Der er plads på kortet til yderligere kredse, idet både RAM på nuværende I K og ROM på 4 K kan udviddes direkte til hhv. 4 K og 24 K på kortet. Adresseringen giver mulighed for ialt 64K byte.
Yderligere er der blevet plads til et par LED's som er tilsluttet i forbindelse med strømforsyningen på 5 volt og audioudgangen.
Der er en lang række "lus" på kortet. Disse giver mulighed for at ændre flere af datamatens parametre som f. eks. baud-rate. Alt i alt virker den mekaniske opbygning solid og gennemtænkt, og det er ikke blot software, at SYM-1 giver større muligheder - datamat-amatøren med loddekolbe vil også finde, at der med SYM-1 er muligheder for at tilpasse datamaten til præcist egne formål.

SUPERMON.
Denne monitor, som ligger i 4K ROM giver stor fleksibilitet i betjeningen af SYM-1. Det ses af tastaturet, at det er muligt at give kommandoer direkte som f.eks. LD 2, som betyder LOAD 2 = indlæsning fra kassettebånd med high-speed format.

Mere usædvanligt for en monitor i denne størrelse er muligheden for at flytte hele blokken ved hjælp af ordren BMOV.

Hvis der er tilsluttet en teletype eller et TV, kan man få stor glæde af ordren VER (verify), som udlæser 8 byte på en gang incl. checksum. Dette vil sammen med plus- og minustegnet, som flytter PC 8 byte frem og tilbage, gøre gennemlæsning af et program særdeles overskuelig, og disse funktioner kan selvfølgelig også benyttes ved programlistninger.
SUPERMON kan både benyttes udefra ved programredigernigeringer o. lign, og fra programmet, idet monitorens funktioner kan benyttes som suprutiner.

SOFTWARE.
Vi fortalte indledningsvist, at der til SYM-1 findes et glimrende udbud af software. Her tænker vi ikke blot på en nydelig, men lidt langsom udgave af BASIC, som kaldes for Tiny BASIC, men især på FOCAL.
Sidstnævnte er et programmeringssprog på samme niveau som BASIC, men med en række fordele omkring bl. a. adresseringen. Vi skal ikke gå i detaljer m.h.t. FOCAL, men vil blot fortælle en smule om netop dette adresseringssystem.
Linienumrene skrives som decimaltal, f.eks. 2.01, 2.02 o.s.v. De sættes i orden af oversætteren, ganske som i BASIC. Der er mulighed for at adresserne op til 99.99.
Opbygningen er gruppeorienteret, idet man ved suprutinekald kan udføre blot en enkelt linie 2.01, eller hele gruppen, f.eks. 2.0. Ligeledes kan f.eks. GOTO følges af en variabel, hvorved indirekte adressering er mulig. Navnet FOCAL er taget fra de tidligere sprog FORTRAN. (forretningsorienteret)FOCAL er til trods for sine mange muligheder for elegante programmeringsløsninger næsten lige så let at lære som BASIC. Dokumentationen lader en del tilbage at ønske, men med noget kendskab til BASIC og/eller ALGOL/FORTRAN skulle FOCAL ikke give andet end glædelige overraskelser.


Syntrek Systems Corp.

FOCAL giver ligeledes mulighed for en kopiering af lageret i binær form tilf.eks. kassettebånd eller disc.

Der bør måske gøres opmærksom på et mindre problem med FOCAL i denne forbindelse. Oversættelsen sker ad 2 gange. 1 omgang, når hver linie er indlæst, og 2. gang, når linien skal udføres. Indholdet af lageret er derfor ikke identisk med det indtastede og må derfor lagres udenom oversætteren, men med de rigtige adresser.

Det er også muligt af få oversætteren til at producere el liste af programmet, som derefter kan indlæses ganske som fra terminalen – dog skal man sørge for al lave en forsinkelse efter hver linie, idet oversætteren som bekendt foretager lidt behandling af programmet efter hver linie. Problemerne omkring program- og data konservering er ikke løst til absolut tilfredshed, men vi kan dårligt forestille os, at man med det sprog som FOCAL vil tillade, at løsningen fortsat mangler.

Her i HFD har vi beskæftiget os en smule med COMAL, et andet Highniveau sprog, som er baseret på 2 andre sprog, og hvor COMAL er til rådighed vil vi nok foretrække dette. FOCAL er dog så klar til en forbedring i forhold til BASIC, at vi absolut hilser velkommen, at dette sprog kan behandles på en SYM-1.

Husk dog, at både BASIC og FOCAL

SYM-1 SYSTEM RAM MEMORY MAP, SY6532

SYMBOL	ADDRESS	DEFAULT VALUE	COMMENTS
IRQVEC	A67F	80	IRQ Vector
	A67E	0F	
RSTVEC	A67D	8B	RESET Vector
	A67C	4A	
NMIVEC	A67B	80	NMI Vector
	A67A	9B	
UIRQVC	A679	80	User IRQ Vector
	A678	29	
UBRKVC	A677	80	User Break Vector
	A676	4A	
TRCVEC	A675	80	Trace Vector
	A674	C0	
EXEVEC	A673	88	'Execute' Vector
	A672	7E	
SCNVEC	A671	89	
	A670	06	Display Scan Vector
	A66F	4C	
URCVEC	A66E	81	
	A66D	D1	Unrecognized Command Vector
	A66C	4C	
	A66B	00	
	A66A	00	Not Used
	A669	00	
INSVEC	A668	89	
	A667	6A	In Status Vector
	A666	4C	
OUTVEC	A665	89	
	A664	00	Output Vector
	A663	4C	
INVEC	A662	89	
	A661	BE	Input Vector
	A660	4C	
YR	A65F	00	
XR	A65E	00	
AR	A65D	00	
FR	A65C	00	User Registers
SR	A65B	FF	
PCHR	A65A	8B	
PCLR	A659	4A	
MAXRC	A658	10	Max. No. Bytes/Record, Paper Tape
LSTCOM	A657	00	Last Monitor Command
TV	A656	00	Trace Velocity
KSHFL	A655	00	Hex Keyboard Shift Flag
TOUTFL	A654	90	In/Out Enable Flags
TECHO	A653	80	Terminal Echo
ERCNT	A652	00	Error Count
SDBYT	A651	4C	Baud Rate
PADBIT	A650	01	Number of Padbits on Carriage Return
P1H	A64F	00	
P1L	A64E	00	
P2H	A64D	00	16-Bit Parameters
P2L	A64C	00	
P3H	A64B	00	
P3L	A64A	00	

SYM-1 SYSTEM RAM MEMORY MAP, SY6532 Cont'd.

SYMBOL	ADDRESS	DEFAULT VALUE	COMMENTS
PARNR	A649	00	No. of Parameters Entered
	A648	00	
	A647	00	Not Used
	A646	00	
RDIG	A645	3F	Right-most Digit
DISBUF	A644	86	
	A643	6E	
	A642	6D	Display Buffer
	A641	00	
	A640	00	
SCRF	A63F	00	} Monitor Scratch Locations SCR0-SCRF
SCR0	A630	00	
JTABLE	A62F	D0	User Socket P3 (Jump Entry No. 7)
	A62E	00	
	A62D	C8	User Socket P2 (Jump Entry No. 6)
	A62C	00	
	A62B	03	0300 (Jump Entry 5)
	A62A	00	
	A629	02	0200 (Jump Entry 4)
	A628	00	
	A627	00	0000 (Jump Entry 3)
	A626	00	
	A625	8B	NEWDEV (Jump Entry 2)
	A624	64	
	A623	8B	TTY (Jump Entry 1)
	A622	A7	
	A621	C0	BASIC (Jump Entry 0)
	A620	00	
SCPBUF	A61F	—	} Scope Buffer, No Defaults (32 Locations)
	A600	—	

NOTES - SYSTEM RAM

	BAUD	SDBYT
BAUD RATE —	110	05
	300	4C
	600	24
	1200	10
	2400	06
	4800	01

TECHO — bit 7 — ECHO/NO ECHO
bit 6 — OUTPUT/NO OUTPUT

This bit is toggled everytime a control O (ASCII 0F) is encountered in the input stream

TOUTFL — bit 7 = enable CRT IN
bit 6 = enable TTY IN
bit 5 = enable TTY OUT
bit 4 = enable CRT OUT

M
53

kræver udbygning med terminal eller teletype, så et fult sæt ASC11-karakterer kan indtastes.

KONKLUSION

Hvis man ønsker til brug for uddannelse eller hobby an anskaffe en billig datamat, er SYM-1 et af de bedste tilbud idag. Prisen på ca Kr. 2.400,-. incl. moms virker rimlig.
Man bør dog være opmærksom på at hvis man ønsker at udbygge SYM-1 med TV-terminal og kassettebåndoptager, kommer man op på godt Kr. 9.000,-. og i denne udbyggede udgave kniber det lidt med konkurrenceevneb i sammenligning med f.eks. PET 2001 og ABC 80.
Alt i alt er SYM-1 dog så klar en forbedring i forhold til tidligere datamater på et enkelt kort, at vi spår den en stor del af det marked, som hidtil har været domineret af KIM-1 og TELEMAC 1800. PH

SYM-1 importeres og importeres af Fa. LISCO, Tlf (05) 56 86 82
Samme firma sælger også KIM-1. Memory Plus, diverse ydre enheder, software mm.

SYM-1 COMMAND SUMMARY

Command HKB/TTY	Code ASCII	Number of Associated Parameters 0	1	2	3
MEM M	4D	Memory Examine and modify, begin at (OLD)	Memory Examine and modify, begin at (1)	Memory Search for byte (1) in locations OLD - 2	Memory Search for byte (1) in locations 2 - 3
REG R	52	Examine and modify user registers PC,S,F,A,X,Y			
GO G	47	Restore all user registers and resume execution at PC	Restore user registers except PC=(1) S=FD, monitor return address is pushed on stack		
VER V	56	Display 8 bytes with address and checksum, begin at (OLD)	Display 8 bytes with address and checksum, begin at (1)	Display (1)-(2), 8 bytes per line, with addresses and cumulative checksums	
DEP D	44	Deposit to memory, beginning at (OLD), CRLF/address after 8 bytes, auto spacing	Deposit to memory, beginning at (1)		
CALC C	43		Calculate 0-(1),or two's complement of (1)	Calculate (1)+(2) or displacement	Calculate (1)+(2)-(3) or displacement with offset
BMOV B	42				Move all of 2 thru 3 to 1 thru 1 + 3 - 2
JUMP J	4A		Restore user registers except PC=entry (1) of JUMP TABLE, S=FD, monitor return pushed on stack		
SDBL SD	*10			Store high byte of (1) in (2)+1 then low byte of (1) in (2); Good for changing vectors	
FILL F	46				Fill all of (2)-(3) with data byte (1)
WP W	57		Write protect user RAM according to low 3 digits of (1)		
LD1 L1	*12	Load first KIM format record found into locations from which it was saved	Load KIM record with ID=(1) into locations from which it was saved	(1) must=FF. Load first KIM record found, but start at location (2)	
LD2 L2	*13	Load first high speed record found into locations from which it was saved	Load high speed record with ID = (1)		(1) must=FF. Load first high speed record found into 2 - 3
LDP LP	*11	Load paper tape in DEMON format. To signal end of file for tape without EOF record, type ;00 CR in on-line mode			
SAVP SP	*1C			Save paper tape locations (1)-(2) in DEMON format. To create end of file record, unlock punch, switch to local mode, lock punch, type ;00 CR	
SAV1 S1	*1D				Save cassette tape locations (2)-(3) with ID = 1, KIM format
SAV2 S2	*1E				Save cassette tape locations 2 - 3 with ID = 1, high speed format
EXEC E	45		Get monitor input from RAM, starting at (1)	Get monitor input from RAM, starting at (2) and store (1) for later use at A64C	Get monitor input from RAM, starting at (3) and store (1) and (2) for later use at A64E and A64C

OPERATIONAL AND SPECIAL KEY DEFINITION (ON-BOARD KEYBOARD ONLY)

Key	ASCII or *Hash Code	Description/Use
CR	0D	Carriage Return (terminates all command strings)
+	2B	Advance eight bytes
–	2D	Retreat eight bytes; also used to delimit parameters
→	3E	Advance one byte or register
←	3C	Retreat one byte
USR0-7	*14-*1B	All USR keys transmit the indicated Hash Code when entered as a command
SHIFT	None	Next key entered is upper position of the selected key
RST	None	System RESET. System RAM reinitialized to default values
DEBUG ON	None	Turn hardware Debug function "ON"
DEBUG OFF	None	Turn hardware Debug function "OFF"
ASCII	None	Next two keys entered Hex will be combined to form one ASCII character

CAI-KORT

LIDT TEORI

Når ens datamat er begyndt at arbejde, finder man hurtigt ud af, at det er rart at kunne gemme et program i en ydre enhed, og derfra genindlæse det, hvis noget går galt eller når man har haft slukket for datamaten.

Den professionelle bruger kan hertil benytte ydre enheder som disk eller digitale båndstationer; men prisen på sådanne enheder betyder desværre, at de for mange amatører må forblive en drøm.

Men da behovet er til stede, er der udvist megen opfindsomhed, som på dette felt især er gået i retning af at anvende en almindelig kassettebåndoptager. Dette er absolut muligt, og der findes endda flere systemer med hver sine fordele.

Det nog mest kendte er Kansas City Standard, som også er det ældste. Denne metode, som oprindelig blev foreslået som standard på et møde mellem de største amatørdatamatproducenter, har den store fordel, at det næsten er ligegyldigt, hvor ringe en kassettebåndoptager, man anvender. Til gengæld er princippet langsomt. Det betød til at begynde med ikke ret meget, men efterhånden som programmerne blev større, blev det mere og mere utilfredsstillende.

Forskellige fabrikanter søgte derfor at sætte hastigheden op, men samtidig steg omkostningerne, fordi der kræves mere og mere af kassettebåndoptagerne og den logik, som forbinder datamat og båndoptager.

Samtidig forsvandt i mange tilfælde muligheden for at »indspille« et program på én type datamat, og læse det igen på en af et andet fabrikat. Ja, i de værste tilfælde kunne programmet kun læses igen af den samme båndoptager, som havde indspillet det.

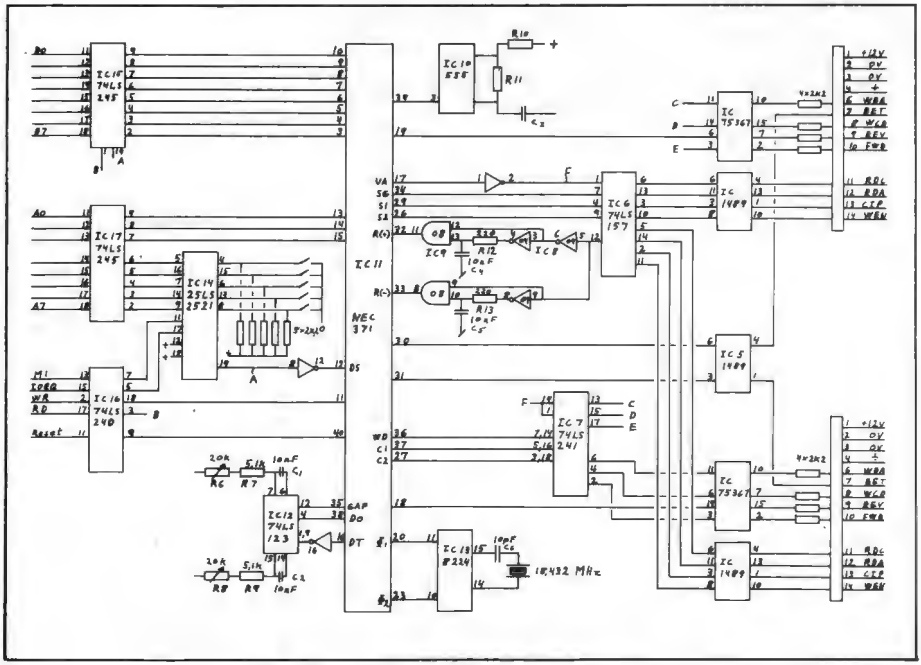
Af disse forbedrede systemer, er dét fra Digital Group nok det mest kendte. Det arbejder næsten 4 gange hurtigere end Kansas City Standard, og da det ikke går helt til den teoretiske grænse for, hvad en kassettebåndoptager kan klare, virker det rimeligt godt.

Bortset fra hastigheden arbejder de 2 principper efter samme retningslinier, og man kan ofte ombygge et interface fra det ene til det andet, blot ved at skifte nogle kondensatorer og modstande.

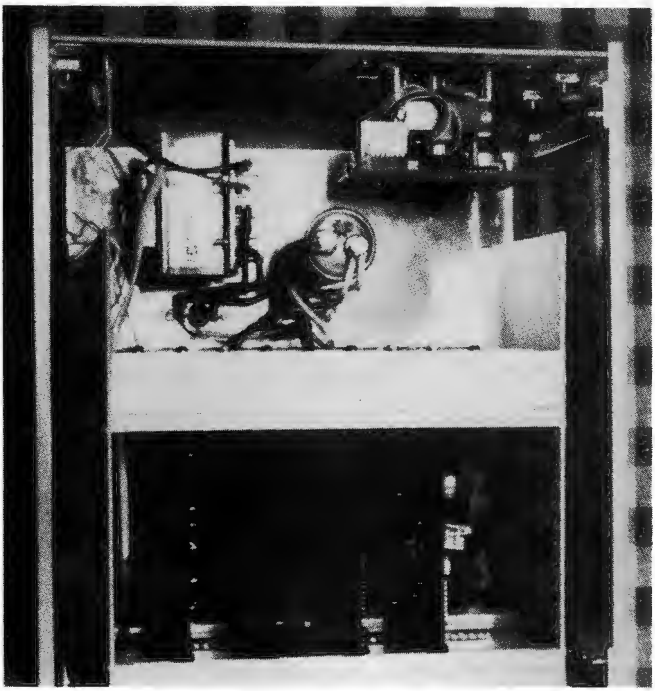
Digital Group princippet kendes også under navnet »Dr. Suding's Princip«.

Et helt andet princip er imidlertid det tredje blandt amatører anvendte system, kendt under navnet »Tarbell«. Her går man helt til den yderste grænse for, hvad en kassettebåndoptager kan præstere; og hvor de 2 foregående principper anvender 2 toner, bruger dette princip fluxvendinger.

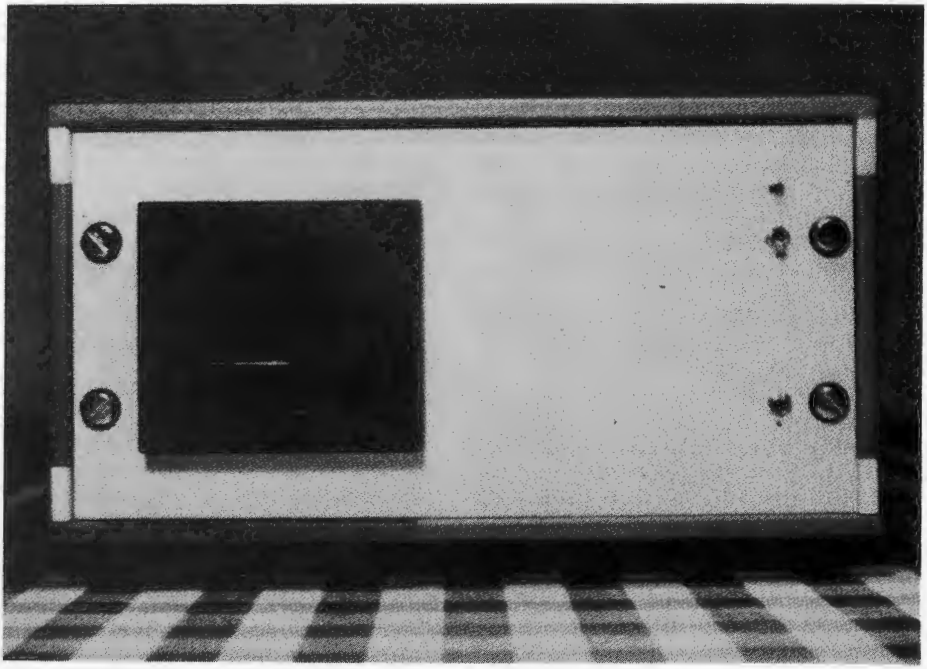
Herved er det muligt at arbejde ca. fem gange hurtigere end Kansas City Standard.



CAI-KORT



Et kig ned ovenfra
i den lille udgave.
Som det ses,
er der god plads.



De små kort giver mange muligheder for indbygning. Her er en version, hvor kassen er 25 cm bred og 14 cm høj. Kassettebåndoptageren ses til venstre. Maskinen er udstyret med 16K ROM- og 32K RAM.

Men hvor hurtige er de forskellige systemer egentlig? Som du måske husker fra SIO-kortet, angiver man hastigheden i en serietransmission i baud, som simpelthen betyder »bit pr. sekund«.
Tænker vi os, at vi skal indlæse et program på 20K ord, består dette af 20.000 × 8 bit = 160.000 bit. Vi kan nu lave en tabel over, hvor lang tid, dette tager:

Telexmaskine	50 baud	= 3.200 sek.
Teletype	110 baud	= 1.450 sek.
Kansas City	300 baud	= 533 sek.
Dr. Suding	1.100 baud	= 145 sek.
Tarbell	1.500 baud	= 106 sek.

Denne tabel siger lidt om de forskellige systemer, men den er ikke hele sandheden. Ofte er det nemlig nødvendigt at indlæse flere gange, før man får en fejlfri læsning.
Hertil kommer, at nogle fabrikanter ikke gemmer bit for bit, men laver hver byte om til 2 hexadecimale tal. Herved fordobles ovennævnte tider, men det er til gengæld muligt at finde fejl, medens programmet indlæses.

Desuden skal tiderne forøges med ca. 30 procent, fordi hver databyte skal tilføjes start- og stopbit.
Selv om de 2 sidstnævnte systemer må siges at være hurtige nok til almindeligt amatørbrug, er der dog stadig et problem. Man kan nemlig let komme ud for, at ens program er så stort, at det ikke på en gang kan være i RAM-lageret. Det må derfor overflyttes del for del. Men dette kan normalt ikke lade sig gøre med en almindelig kassettebåndoptager, da start og stop er rent mekaniske funktioner, som det er vanskeligt at ombygge, så datamaten kan kontrollere dem.
Disse styringsproblemer betyder, at en almindelig kassettebåndoptager egentlig ikke er særlig velegnet til vort brug, men der har faktisk ikke været nogen alternativer.

Heldigvis er det ikke kun amatørerne, der er i klemme. Heller ikke det professionelle marked kan i længden affinde sig med, at en enkelt perifer enhed er dyrere end selve datamaten.
Der er derfor begyndt at komme forskellige ydre enheder frem, hvor man, ved at give afkald på de mest udspekulerede faciliteter, er i stand til at sænke prisen.
Et eksempel herpå blev første gang vist i foråret på Hannovermessen, hvor den vakte temmelig stor opsigt.
Det var en rigtig lille digital kassettebåndoptager med alle de vigtigste egenskaber for denne maskintype, men som kun kostede ca. eh femtedel af, hvad man var vant til.
Det foregik på Philips' store stand, og giraffen hedder M-DCR 220.
Kort fortalt arbejder enheden med 6.000 baud, kan køre både forlæns og baglæns, kan genlæse, hvis der har været fejl, gemmer op til 128K byte, har 2 spor og er udstyret med de nødvendige kontrolsignaler.
Vi finder, at denne båndoptager er en virkelig gevinst for datamatamatørerne og giver en række nye muligheder, som bør udnyttes. Derfor konstruerede vi kortet, som i det følgende skal beskrives.

DIAGRAMMET

Vor trofaste læser har forlængst bemærket, at vi har en fiks idé med at gøre alting så simpelt, som muligt. Det præger også dette kort. Det er nemlig opbygget omkring den integrerede kreds NEC-371, som er beregnet til at styre båndoptagere. Herved slipper vi nemt om ved, dels omsætningen mellem serie og parallel, dels styringen af kontrolsignalerne. I tilgift får vi automatisk retur og CRC-undersøgelse. Det sidste er en meget brugt metode til at undersøge, om de læste data er rigtige.
Gennem de sædvanlige drivere føres datalinierne op til 371. Ligeledes bruges den sædvanlige decodning til at fortælle kredsen, når den er udpeget.
371 er egentlig beregnet til at arbejde sammen med 8080 og bruger de samme clocksignaler, som denne CPU. Disse har vi ikke, og de dannes derfor på kortet ved hjælp af en krystaloscillator.
Hastigheden, hvormed der læses og skrives, bestemmes af en anden oscillator, hvis frekvens ved hjælp af R 10 stilles til 12 kHz.
371 er beregnet til at styre 2 båndoptagere. Til at fortælle, hvilken der i et givet øjeblik skal i brug, er den udstyret med en speciel udgang, kaldet UA. Dette signal føres gennem en inverter og videre til kredsene IC6 og 7, som skifter mellem de to enheder.
Herfra går signalerne gennem kredsene IC1-4. Båndoptageren er nemlig opbygget med CMOS-logik, som arbejder med 12 V, og disse kredse er nødvendige for at omsætte mellem spændingerne.
Når man gemmer data på bånd, er det meget almindeligt at gøre det i blokke med hver et bestemt antal byte. Mellem hver blok er der så et tomt område, hvor båndoptageren kan starte og stoppe. IC 12 overvåger, om man er inde i et sådant tomt område.
Nærmere beskrivelse af, hvorledes 371 fungerer, findes i den manual, som følger med kredsen. Her er der også et forslag til den software, som er nødvendig. Den kan dog ikke bruges uændret. Her er i det hele taget et problem. Hvis vi ønsker at kunne udveksle programmer, er det nødvendigt, at vi alle bruger samme format. I praksis sker dette lettest ved at bruge samme software. I COMAL og den store monitor findes dette program, og vi håber, det kan blive standard.

EFTERSKRIFT

Dette er sidste nummer af HFD. Ringbindet er fyldt og vi står tilbage med et problem. Vi er nemlig slet ikke færdige med at beskrive de forskellige kort, men efterhånden har vi en nogenlunde fornemmelse af, hvordan det hele ender.
Vi regner med, at der bliver ca. 20 forskellige kort. I øjeblikket arbejder vi på et kort til CRT for almindeligt fjernsyn, et kort til floppy disk og et parallel I/O kort. Derefter følger et monitorkort med både RAM og ROM, et kort til at drive en printer og et kort for analog I/O. Længere ud i fremtiden spekulerer vi på kort for farveskærm og speech synthesizer, så vi kan tale til vor datamat — og den til os.
HFD fortsættes som specialsider i POPULÆR ELEKTRONIK.

Z-80

TEKSTVARIABLE

Vi har hidtil kun betragtet variable, hvis værdi kan være **tal.** Sådanne variable kaldes også **talvariable** eller **numeriske** variable. I følgende sætning:

```
NUMMER=8
```

er NUMMER således en **talvariabel,** som tildeles **værdien** 8. Vi har også set eksempler på **tekstkonstanter** eller **konstante tekster.** I følgende sætning:

```
PRINT "SALGSPRISEN ER: ";SPR
```

er SALGSPRISEN ER: " en **tekstkonstant.** Hver gang, sætningen udføres, udskrives **den samme tekst,** og det kan kun ændres ved, at man skriver programmet om og indsætter en anden konstant tekst eller evt. sletter linjen.
I de fleste BASIC-versioner har man også mulighed for at anvende **tekstvariable,** altså variable hvis værdi er **tekster** i stedet for tal. For at belyse sammenhængen mellem tekstkonstanter og tekstvariable kan vi betragte følgende korte meddelelse, som anvendes dagligt over det ganske land:

```
DET VIL GLÆDE                    AT SE
                          TIL FØDSELSDAG
DAG DEN    /    KL.     TIL KL.    .
JEG BOR                             .
S.U.
```

På stregerne kan naturligvis indsættes passende ord og tal, så der fremkommer en meningsfuld invitation. Den fortrykte tekst kan betragtes som en mængde af tekstkonstanter, mens stregerne kan siges at repræsentere en række **variable tekster.** Man kan ikke forestille sig, at to kort bliver udfyldt ens. Vi vil nu skrive et program, som kan udskrive sådanne fødselsdagsinvitationer, når man blot indtaster de fornødne tekster til programmet.
I COMAL ser programmet således ud:

```
0010 DIM AFS$(20),MODT$(20),DAG$(7),DATO$(5)
0020 DIM ANK$(5),AFG$(5),ADR$(30)
0030 REM ** INDDATA **
0040 INPUT "AFSENDER (KUN FORNAVN):      ",AFS$
0050 INPUT "MODTAGER (KUN FORNAVN):      ",MODT$
0060 INPUT "UGEDAG (STAVES FULDTUD):     ",DAG$
0070 INPUT "DATO (DD/MM):                ",DATO$
0080 INPUT "BEGYNDER KL. (TT.MM):        ",ANK$
0090 INPUT "SLUTTER KL. (TT.MM):         ",AFG$
0100 INPUT "ADRESSE (GADE HUSNR):        ",ADR$
0110 REM ** UDSKRIFT **
0120 PRINT "DET VILLE GLAEDE ";AFS$;" AT SE ";MODT$;" TIL"
0130 PRINT
0140 PRINT "FOEDSELSDAG ";DAG$;" DEN ";DATO$;", KL. ";ANK$
0150 PRINT
0160 PRINT "TIL KL. ";AFG$;"."
0170 PRINT
0180 PRINT "JEG BOR ";ADR$
0190 PRINT
0200 PRINT "S.U."
```

De to første sætninger i programmet er DIM-sætninger, og når de udføres, bliver der som sædvanlig reserveret plads i lageret til data. I dette tilfælde skal data imidlertid ikke være tal men tekster, og det viser man ved at anbringe et $-tegn efter den variables navn. Når fx. den første sætning udføres, bliver der afsat plads til **20 tegn** under navnet AFS$, 20 tegn under navnet MODT$, 7 tegn under navnet DAG$ og 5 tegn under navnet DATO$. Når linje 40 udføres, skriver systemet først:

AFSENDER (KUN FORNAVN):

og venter på, at der skal blive indtastet et navn. Lad os tænke os, at navnet BETTINA bliver indtastet. Dette navn bliver af systemet naturligvis blot opfattet som en **række tegn** (en tekst), og rækken bliver lagret i det område, der er reserveret den variable AFS$. Vi kan forestille os følgende billede:

AFS$: BETTINA

hvor jeg har antydet, at de overskydende pladser står tomme. I virkeligheden er det ikke tilfældet, men det er uden betydning, hvad der står på disse pladser, idet systemet opretholder et **pil**, der altid peger på det **sidste tegn** i den tekst, som er blevet læst ind i AFS$. Man siger også, at AFS$ har **fået tildelt** teksten "BETTINA" som **værdi.** Hvis man forsøger at indtaste en tekst, der er mere end 20 tegn lang, går de tegn, der følger efter det 20'nde, tabt, men man skal ikke forvente nogen fejlmelding fra systemet. I de følgende sætninger administreres indtastning og tildeling af de øvrige tekster som værdier for de tilsvarende tekstvariable.
Når indtastningen er slut, udføres PRINT-sætningerne. Lad os se på den første:

```
PRINT "DET VILLE GLAEDE ";AFS$;"
AT SE ";MODT$;" TIL"
```

Efter nøgleordet PRINT følger en række tekstkonstanter og tekstvariable, adskilt ved **semikolon**. Hvis MODT$ har været "LOTTE", giver sætningen følgende udskrift:

DET VILLE GLAEDE BETTINA AT SE LOTTE TIL

Læg godt mærke til de indsatte **mellemrum.** Hvis man ikke husker dem, kan man risikere en udskrift som denne:

DET VILLE GLAEDEBETTINAAT SELOTTETIL

og det forhøjer næppe forventningerne til festen. Når semikolon bruges som skilletegn mellem **tekster,** sætter systemet **ikke** automatisk mellemrum, således som det er tilfældet, når semikolon bruges som skilletegn mellem tekster og **tal.** Ved at tænke lidt over sagen vil læseren indse, at dette er en stor fordel, idet programmøren da har det fuldstændige herredømme over den sammensatte teksts udseende. Man kan også anvende **komma** som skilletegn mellem tekstelementer, men det kan give anledning til ret ukontrollable udskrifter og bør i almindelighed undgås. Derimod kan komma undertiden med fordel benyttes som skilletegn mellem tekster og tal.

Jeg har ikke hidtil omtalt denne facilitet, og det skyldes, at brugen af komma kan være ret forskellig fra system til system. Det tilrådes læseren at konsultere systemhåndbogen for at finde ud af, hvordan **kommaer** i PRINT-sætninger virker. I sådanne tilfælde kan det i øvrigt anbefales, at man udfører nogle eksperimenter med forskellige udskrifter for at få en fornemelse af, hvordan tegnene virker.
Vi vil afslutte gennemgangen af "fødselsdagsprogrammet" med at give et eksempel på en udskrift fra det:

```
DET VILLE GLAEDE BETTINA AT SE LOTTE TIL

FOEDSELSDAG ONSDAG DEN 18/10, KL. 15.15

TIL KL. 17.30.

JEG BOR VIBEN 38

S.U.
```


ØVELSE
Gå udskriften igennem og sammenlign med PRINT-sætningerne i programmet. Sæt streg under de tekstdele, der er fremkommet som udskrifter af **værdier for tekstvariable.**
Det tilsvarende program i standard-BASIC er vist herunder. Det er ikke meget forskelligt fra COMAL-programmet, men man er som sædvanligt henvist til at bruge variabelnavne, der kun består af ét bogstav eller et bogstav og et ciffer. Brugen af $-tegn er den samme som i struktureret BASIC. Betydningen af at have lange variabelnavne til rådighed fremtræder dog også i dette lille program. Således er man nødt til at kalde DAG$ for D$ og må derfor kalde DATO$ for D1$. At D$ og D1$ har disse betydninger er ikke uden videre indlysende for eventuelle udenforstående!

```
0010 DIM A$(20),M$(20),D$(7),D1$(5)
0020 DIM A1$(5),A2$(5),A3$(30)
0030 REM ** INDDATA **
0040 INPUT "AFSENDER (KUN FORNAVN):      ",A$
```

```
0050 INPUT "MODTAGER (KUN FORNAVN):      ",M$
0060 INPUT "UGEDAG (STAVES FULDTUD):     ",D$
0070 INPUT "DATO (DD/MM):                ",D1$
0080 INPUT "BEGYNDER KL. (TT.MM):        ",A1$
0090 INPUT "SLUTTER KL. (TT.MM):         ",A2$
0100 INPUT "ADRESSE (GADE HUSNR):        ",A3$
0110 REM ** UDSKRIFT **
0120 PRINT "DET VILLE GLAEDE ";A$;" AT SE ";M$;" TIL"
0130 PRINT
0140 PRINT "FOEDSELSDAG ";D$;" DEN ";D1$;", KL. ";A1$
0150 PRINT
0160 PRINT "TIL KL. ";A2$;"."
0170 PRINT
0180 PRINT "JEG BOR ";A3$
0190 PRINT
0200 PRINT "S.U."
```

MERE OM TEKSTVARIABLE

I det følgende vil vi se, hvorledes et mere omfattende COMAL/BASIC-program opbygges. Vi skal herunder lære flere interessante detaljer vedrørende programmering, og vi skal se, hvorledes man kan arbejde med tekstvariable i en større sammenhæng. I teksten vil jeg henvise til den programliste, som findes på side — . Programmet er delt op i en række **procedurer (underprogrammer)**, og jeg vil gennemgå disse enkeltvis. Først skal der dog kort redegøres for, hvad programmet som helhed udretter. Med en given tekst som inddata, analyserer det teksten og tæller op, hvormange **vokaler, konsonanter, cifre** og **specialtegn,** der er i teksten. Endvidere tæller programmet, hvor mange **ord,** der er i teksten. Der er altså tale om et program, som udfører processer med tekster, og som giver statistiske resultater. Måske én og anden af læserne synes, det er noget »skolelæreragtigt«, men de principper, der bruges i programmet, har meget almene anvendelser. Det er faktisk således, at programmer, der arbejder med tekster, er vigtigere i moderne databehandling end programmer, der arbejder med numeriske data.

Lad os begynde med at betragte proceduren INDTEKST (80—170). I linie 90 reserveres der plads i lageret til 2000 tegn under navnet TXT$ og til 80 tegn under navnet LINJE$. Når linje 100—120 udføres, skriver programmet:

```
INDTAST TEKSTEN:
>
```

og venter på, at operatøren skal indtaste en tekst (en linje). Den indtastede tekst tildeles den variable LINJE$ som værdi. I linje 130—160 finder vi en løkke-struktur, vi ike har set før.. Styresætningen i linje 130 ser således ud:

```
WHILE LINJE$<>"SLUT" DO
```

og kan oversættes til: "SÅLÆNGE den indtastede tekst er forskellig fra "SLUT", skal følgende udføres". Det, der skal udføres, er naturligvis afsnittet, der afsluttes med ENDWHILE-sætningen. Man bemærker, at de samme **relationsoperatorer** (<, >, <=, => og <>), som anvendes ved sammenligninger mellem talværdier, også kan anvendes mellem tekster. Det Boolske udtryk: LINJE$ <>"SLUT" har værdien **sand,** sålænge værdien af LINJE$ **ikke** er lig med "SLUT", altså så længe operatøren ikke har indtastet en linje, der kun består af det ene ord. I linje 140 finder vi sætningen:

```
LET TXT$=TXT$, LINJE$, " "
```

Der er flere ting at bemærke om denne tildeling. På højre side finder man tegnet **komma** brugt som **sammenkædningstegn (konkateiationstegn).** Tildelingen udføres analog med tildelinger som fx.:

```
A=A+B
```

Først udføres højre side, og den derved fremkomne værdi tildeles den variable på venstre side som ny værdi. Lad os tænke os, at TXT$ har denne værdi:

```
I DENNE TID KOERER MANGE
```

og LINJE$ denne værdi:

```
TUSINDE DANSKERE TIL AVENTOFT
```

Når linje 140 derpå udføres, sammenkædes den sidste tekst med den første til:

```
I DENNE TID KOERER MANGE TUSINDE
DANSKERE TIL AVENTOFT
```

og den således fremkomne tekst **tildeles TXT$** som ny værdi. Læg også mærke til, at der hver gang kædes et mellem (" ") til afslutningen af teksten. Hvis man ikke gør det, bliver den næste linje kædet på uden mellemrum. Resultatet kune fx. blive følgende:

```
I DENNE TID KOERER MANGETUSINDE
DANSKER TIL AVENTOFT
```

Man skal se sig godt for, når man arbejder med tekstvariable!
I linje 150 er der lejlighed til at indtaste en ny linje, og umiddelbart derpå bliver sætningen, der består af det ene ord END-WHILE udført. Det bevirker, at systemet **går tilbage** til WHILE-sætningen, hvor det bliver undersøgt, om den nye linje skulle bestå af ordet "SLUT" og ikke andet. Hvis det **ikke** er tilfældet (LINJE$ <> "SLUT" er **sand**), udføres linje 140 og 150 atter, med det resultat, at den nye linje kædes på teksten i TXT$. Hvis derimod den indtastede tekst netop er ordet "SLUT", har det Boolske udtryk i WHILE-sætningen værdien **falsk,** og systemet **fortsætter da udførelsen med sætningen efter** END-WHILE. Alle **højere** programmeringssprog indeholder en sådan WHILE-løkke. I standard BASIC har man den naturligvis ikke (standard BASIC er **ikke** noget højere sprog!). Jeg skal senere vise, hvorledes man **oversætter** WHILE-løkker til BASIC. — Vi kan sammenfattende beskrive virkningen af proceduren således: Der indtastes en række linjer, og disse hægtes på teksten i TXT$, efterhånden som de ankommer. Når operatøren taster ordet "SLUT", standses udførelsen af løkken. Det bør bemærkes, at TXT$ indeholder **den tomme tekst,** umiddelbart efter, at DIM-sætningen i linje 90 er udført. Dette kan jævnføres med, at tal-tabeller indeholder et 0 i hver komponent, umiddelbart efter at de er dimensioneret. Den første linje, der indtastes, bliver altså anbragt forrest i TXT$.

ØVELSE
Hvorfor kan programmet:

```
REPEAT
   INPUT LINJE$
   TXT$=TXT$,LINJE$," "
UNTIL LINJE$="SLUT"
```

ikke bruges i procedure INDTEKST?

Når INDTEKST er afsluttet, bliver TAELOP (190—260) udført. Også i denne finder man en række detaljer om tekstvariable. I linje 200 bliver en række **tællere** sat lig med 0 (**nulstillede**). Tildelingen i linje 210 vender vi tilbage til, da den kræver en nærmere forklaring. I linjen 220 finder vi sætningen:

```
FOR NR=1 TO LEN(TXT$)
```

Denne sætning indleder en løkke, hvis afslutning findes i linje 250 med sætningen:

```
NEXT NR
```

Vi finder i FOR-sætningen også en ny **funktion,** nemlig LEN (LENgth), hvis værdi er **længden** af den tekst, der er givet som argumentation for funktionen. Længden af en tekst er det **antal tegn** (alt iberegnet,

altså også fx. mellemrum), den **indeholder.** LEN(TXTS) angiver altså det samlede antal tegn, der er blevet tildelt TXT$. Dette tal må imidlertid være det samme som **nummeret på det sidste tegn** i TXT$, og det er netop det, vi er interesseret i.. Lad os tænke os, at TXTS indeholder i alt 125 tegn. Så vil LEN(TXTS) blive udregnet til værdien 125, og derpå bliver linjerne 230 –240 i FOR...NEXT-løkken udført **for NR lig med** 1, 2, 3, ..., 125. I linje 230 finder vi sætningen:

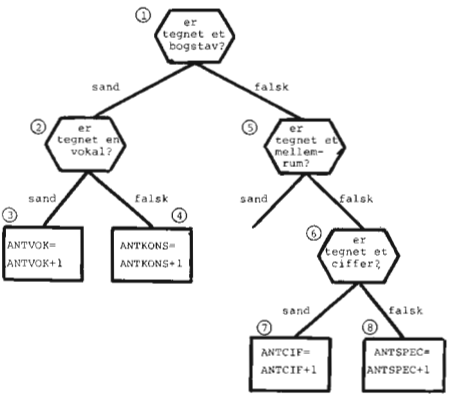
```
LET TEGN$=TXTS(NR)
```

Lad os tænke os, at TXT$ begynder således:

```
DU DANSKES VEJ TIL ROS OG MAGT...
```

Når NR nu sættes til 1, siger sætningen i linje 230, at systemet skal sætte TEGN$ lig med TXT$(1), hvilket betyder **tegn nummer 1** i TXT$. TEGN$ får altså tildelt værdien "D". Derpå kalder proceduren ANALYSE, som vi ser på om lidt. Når det er sket, udføres NEXT NR, og det bevirker, at NR sættes til 2. Igen udføre tildelingen i linje 230 med det resultat, at TEGN$ tildeles værdien af TXT$(2), hvilket vil sige **tegn nummer 2** i TXT$, altså "U". Igen udføres ANALYSE, NR sættes lig med 3, og 230 udføres atter. Således går det videre, indtil NR er lig med nummeret på det sidste tegn i TXT$. Så udføres 230 og 240 for sidste gang, og TAELOP afsluttes.

Vi kommer nu til proceduren ANALYSE (490—570), som er kernen i hele programmet, og som også er noget vanskeligere at skrive og forstå, end de øvrige. Jeg giver derfor en detaljeret gennemgang af den. Lad os først se på, hvad den skal udføre. Når vi kommer til ANALYSE fra kaldet i linje 240, har den variable TEGN$ fået tildelt værdien af et tegn i TXT$, og vi skal nu søge at finde ud af, om dette tegn er en vokal, en konsonant, et ciffer eller et specialtegn. Vi skal også finde ud af, om der begynder et nyt ord på dette sted, da vi også skal have talt, hvor mange ord, teksten indeholder. Vi har altså noget at se til. Før jeg skriver en sådan procedure, laver jeg et diagram over de logiske beslutninger, der skal træffes i proceduren. Det diagram, jeg benytter, kaldes et strukturdiagram og er først indført af professor Keneth Bowles ved University of California (UCLA).



For de fleste tekster er det mest sandsynligt, at det tegn, vi skal undersøge, er et **bogstav.** Vi ser derfor først efter, om det er tilfældet. Hvis tegnet er et bogstav, er der kun to muligheder, nemlig at det er en **vokal** eller en **konsonant.** Følgelig undersøger vi det, og er dermed færdig med det tilfælde, at tegnet er et bogstav.
Hvis **ikke** tegnet er et bogstav, ser vi efter, om det er et **mellemrum**, og hvis det **ikke** er tilfældet, undersøger vi, om det er et **ciffer.** Er det ikke et ciffer, vil vi regne det for at være et **specialtegn.** Af grafen fremgår, at hvis tegnet er et mellemrum, skal der ikke foretages yderligere, men undersøgelsen er afsluttet. Ud fra denne analyse kan vi beskrive en plan for undersøgelsen af tegnet:

```
IF tegnet er et bogstav THEN
   udfør 2
ELSE
   udfør 5
ENDIF
```

Vi går videre og udfylder blokkene 2 og 5:

```
TEGN$<>" "

IF tegnet er et bogstav THEN
   IF tegnet er en vokal THEN
```

```
      udfør 3
   ELSE
      udfør 4
   ENDIF
ELSE
   IF tegnet ikke er et mellemrum
   THEN
      udfør 6
   ENDIF (intet alternativ)
ENDIF
```

Blokkene 3 og 4 er blot optællinger, og vi ved, hvordan det skal gøres. Blok 6 skal indsættes, og vi har til slut blok 5:

```
IF tegnet ikke er et mellemrum THEN
   IF tegnet er et ciffer THEN
      udfør 7
   ELSE
      udfør 8
   ENDIF
ENDIF
```

Blokkene 7 og 8 er også optællinger, som vi uden videre kan programmere. Den lidt besynderlige blanding af programmeringssprog og almindelig »stenografdansk« virker måske lidt besynderligt, men metoden er let at forstå, og hvis man ikke kan lide at skrive fx. IF kan man blot skrive **hvis**. Den viste metode for algoritmeudvikling kaldes også top-down, og består altså kort i, at man begynder med en meget kort og overskuelig beskrivelse af processen, og derpå gør denne beskrivelse mere og mere detaljeret, indtil det niveau, hvor resten er trivielle enkeltheder.

Vi skal derpå have de mange Boolske udtryk, som foreløbig er beskrevet på dansk, oversat til COMAL. Lad os begynde med denne: "tegnet er et bogstav". Idet vi regner med, at tegnet er tildelt TEGN$ som værdi, kan det pågældende Boolske udtryk formuleres således i COMAL:

```
"A"<=TEGN$ AND TEGN$<="Z"
```

Her står egentlig blot: »tegnet følger efter eller er lig med "A" og kommer før eller er lig med "Z"«. Tegn, for hvilket dette gælder, er netop bogstaver i det engelske alfabet. Hvis man er den lykkelige ejer af et system, som tillader benyttelse af danske bogstaver, bliver det Boolske udtryk naturligvis til:

```
"A"<=TEGN$ AND TEGN$<="Å"
```

Ordet AND er en såkaldt logisk (eller Boolsk) **operator** og betegner det samme som det **logiske "og"** i matematikken. Hvis p og q er to Boolske udtryk, vil det Boolske udtryk:

```
p AND q
```

være sand, når p og q **begge er sande,** og ellers falsk.
Ovenstående Boolske udtryk er altså sandt, når tegnet ligger mellem "A" og "Z" ("Å") inklusive. I mange nyere BASIC-versioner findes de logiske operatorer, dette gælder bl.a. den på mange måder veludviklede PET-BASIC. De to andre Boolske operatorer er OR og NOT, der svarer til matematikkens **logiske "eller"** og **logiske "negation".** Hvis p og q er to Boolske udtryk, vil det sammensatte Boolske udtryk:

```
p OR q
```

være **falsk,** når både og p og q er falske, og ellers sandt.
Det Boolske udtryk:

```
NOT p
```

har altid den **modsatte sandhedsværdi** af p. Såvel OR som NOT findes i COMAL og også i fx. PET-BASIC.
Vi fortsætter med at se på denne: "tegnet er et ciffer", som kan udtrykkes på helt tilsvarende måde:

```
"0"<=TEGN$ AND TEGN$<="9"
```

idet et tegn er et ciffer, når det ligger mellem "0" og "9" inklusive.

Udtrykket: "tegnet er **ikke** et mellemrum" er det letteste af dem alle:
(tegnet er **forskelligt fra mellemrum**). For den, der kan lide det mere udsøgte, kan man også udtrykke det således:

```
NOT TEGN$ = " "
```

Det første må dog foretrækkes, da det er hurtigere for systemet at udføre.
Tilbage har vi det Boolske udtryk: "tegnet er en vokal", som ikke er helt så simpelt som de øvrige. Grunden hertil er den enkle, at vokalerne ikke danner nogen sammenhængende delfølge af tegnsættet, således som alfabetet og cifrene. Vi henlægger beregningen af sandsynlighedsværdien til en særlig procedure VOKAL (510 –570), som vi vil se på, inden vi gør ANALYSE FÆRDIG. Når VOKAL udføres, har VOKAL$ fået tildelt værdien "AEIOUY" — altså mængden af vokaler i det engelske alfabet. Vi begynder med at analysere tildelingen i linje 550:

```
LET VOKAL=(TEGN$=VOKAL$(J))
```

I parentesen på højre side af tildelingstegnet står det Boolske udtryk:

```
TEGN$=VOKAL$(J)
```

Hvis dette udtryk har værdien **sand,** bliver VOKAL sat lig med **1,** og hvis det har værdien **falsk,** sættes VOKAL lig med **0.** En variabel, der bruges på denne måde, kaldes en **Boolsk variabel** (»sandhedsvariabel«). Efter denne tildeling udføres instruktionen:

```
UNTIL VOKAL OR J=6
```

Det sammensatte Boolske udtryk efter REPEAT består af to komponenter, nemlig den Boolske variabel VOKAL og det simple Boolske udtryk: J=6. Den variable VOKAL bliver af systemet behandlet ganske som et simpelt Boolsk udtryk, idet den regnes for at have værdien **sand,** hvis den er lig med 1, og værdlien **falsk,** hvis den er lig med 0. Denne fortolkning modsvarer fuldstændig de omstændigheder, under hvilke VOKAL får tildelt sin værdi. Man kan sige, at VOKAL »husker«, hvilken sandhedsværdi udtrykket TEGN$=VOKAL$(J) havde, da linje 540 blev udført.
Boolske variable bliver undertiden kaldt »flag«, og man siger, at »flaget er sat«, når den variable har værdien sand, og at »flaget er taget ned«, når den variable har værdien falsk. Billedet af et flag, der bliver sat eller taget ned, er ganske rammende. Som et flag kan være signal for omgivelserne, er en Boolsk variabel også et signal, der kan læses i andre dele af programmet. Udførelsen af løkken standser altså, hvis VOKAL har værdien sand, dvs. TEGN$ er blevet identificeret som en af vokalerne i VOKAL$, eller J er lig med 6, hvilket er ensbetydende med, at hele rækken af vokaler er blevet »set igennem«.
Procedure ANALYSE er nu næsten færdig. Vi mangler blot at få forklaret sætningerne i linje 310 og 390, der tilsammen varetager optællingen af ord i teksten. Disse to sætninger er til gengæld skrevet efter nogen omtanke. Lad os prøve at betragte følgende tekst: " OLE OLSEN KAN NÅ LANGT MED DEN FART". Vi læser det første tegn og konstaterer, at det er et mellemrum. Det gælder også det næste, og først det tredje tegn viser sig at være et bogstav. **Her begynder altså et ord.** Det tæller vi, og noterer os samtidigt, at vi nu er begyndt at »stave« et ord. Det næste tegn — det 4'de — er også et bogstav, men det betyder **ikke,** at vi nu har fundet et nyt ord! Tværtimod, vi er midt i et. Der sker ikke noget i denne sammenhæng interessant, før vi læser det 6'te tegn. Det er et mellemrum, og det betyder, at nu er **det ord,** vi begyndte på for lidt siden, **slut.** Næste gang, vi møder et bogstav, **begynder altså et nyt ord.** Hvordan får et program til at finde ud af sådan noget? Læseren kan prøve at forestille sig ordene i teksten ovenfor som en række vogne i et tog, der kører forbi en lille mand med et stort, rødt flag. Læseren står på den anden side af banen, og hver gang en vogn kører forbi, bliver den lille mand og hans store, røde flag skjult for læseren af vognen. For at få at vide, hvor mange vogne, »ordtoget« indeholder, behøver læseren blot at tælle, hver gang den lille mand og hans store, røde flag **forsvinder** ud af syne. Lad os igen se på denne tekst: " OLE OLSEN KAN NÅ LANGT MED DEN FART.", og lad os tænke os, at teksten »kører forbi« manden med flaget. Flaget kan ses, indtil det første bogstav "O" ruller ind, og der tælles "én". Flaget er stadig dækket, når "L" og "E" passerer, men kommer derpå til syne igen, og vi gør os klar

til at tælle næste ord-vogn. Når næste "O" ruller ind, forsvinder flaget påny, og der tælles til "to". Således fortsættes til hele ord-toget er kørt forbi, og der er blevet talt til otte. Lad os prøve at beskrive det på en måde, der ligger nærmere ved vort programmeringssprog:

```
først kan man se flaget
så kommer teksten, tegn for tegn:
    IF det tegn, der ankommer, er et
    bogstav THEN
        IF flaget kan ses THEN flaget
        forsvinder, nyt ord tælles
        ...
    ELSE ( tegnet er ikke et bogstav )
        flaget kan nu ses
    ...
    ENDIF
```

I programmet har jeg kaldt flaget ORDSLUT, fordi det kan »ses«, når vi **ikke** er inde i et ord. ORDSLUT er naturligvis en Boolsk variabel, der altså kan antage værdierne **sand** (=1) eller **falsk** (=0). For at gøre programmet så let forståeligt som muligt, har jeg indført to **konstanter** TRUE og FALSE, som er sat til hhv. 1 og 0 i begyndelsen af programmet (linje 40). I stedet for at skrive fx.:

```
LET ORDSLUT=1
```

når vi ønsker den Boolske variabel ORDSLUT skal fortolkes som **sand,** kan vi i stedet skrive:

```
LET ORDSLUT=TRUE
```

På tilsvarende måde kan vi give ORDSLUT værdien **falsk** ved tildelingen:

```
LET ORDSLUT=FALSE
```

Måske læseren synes, det er at gå over åen efter vand, men det er vigtigt, at man giver de forskellige størrelser navne og lader dem optræde i sammenhænge, der gør det helt tydeligt, hvad de står for. Hvis vi blot skriver: ORDSLUT=1, kan det betyde så meget, og det er ikke umiddelbart klart, hvilken rolle ORDSLUT spiller.

Når man derimod ser tildelingen: ORDSLUT=TRUE, er man ikke i tvivl om, at ORDSLUT bærer omkring med en **sandhedsværdi** og altså anvendes som Boolsk variabel. Man ved da en hel del om, i hvilke sammenhænge man kan vente at møde den.
Tilbage står blot at konstatere, at programmets forløb styres fra linje 690—740. De to procedurer INITIAL og UDDATA er så simple, at forklaringer turde være overflødige.

På side — er vist, hvorledes programmet kan skrives i almindelig BASIC. Omskrivningen er i det store og hele foretaget efter de tidligere nævnte regler, men der er dog grund til at nævne et par detaljer. WHILE-løkken i INTEKST udføres af sætningerne i linje 220—250. I almindelighed »oversættes« en WHILE-løkke således:

```
WHILE p DO          xx IF not p THEN yy
   programblok         programblok
ENDWHILE               GOTO xx
                    yy ...
```

Læseren kan selv overbevise sig om, at de to programmer er ækvivalente. I det aktuelle tilfælde har vi imidlertid en mulighed for at oversætte WHILE-løkken på en måde, der faktisk gør den kortere i BASIC ind i COMAL. Den INPUT-sætning, som er anført i linje 220, kan bruges både som start på løkken og undervejs i denne, idet vi her har muligheden for at springe tilbage til den fra linje 250. Man kan lettest udtrykke det på den måde, at løkken omfatter linjerne 220—250, og at vi **springer ud af den** (linje 230), hvis L$ er lig med "SLUT". I den nyeste COMAL-version, som bl.a. findes til MPS 2000 (Mogens Pelles datamat), har man som tidligere nævnt indført etn ny løkke-struktur, LOOP...ENDLOOP, der netop med fordel kan anvendes her:

```
LOOP
   INPUT LINJE$
   IF LINJE$="SLUT" THEN EXIT
   TXT$=TXT$,LINJE$," "
ENDLOOP
```

Denne løkke svarer ganske nøje til BASIC-strukturen ovenfor, men man er ikke bundet til at holde regnskab med linjenumrene, som man skal i BASIC.
Der er også grund til at hæfte sig ved den del af BASIC-programmet, som undersøger om et forelagt tegn er vokal (linje 730—810). Her har jeg benyttet en FOR... NEXT-løkke:

```
FOR J=1 TO 6
   IF T$=V$(J) THEN 790
NEXT J
V=0
RETURN
V=1              (790)
RETURN
```

Hvis **ikke** tegnet er en vokal, løber løkken til ende, uden at der hoppes til 790. Som følge heraf bliver V sat lig med 0 (770) og proceduren afsluttes (780).
Hvis tegnet derimod **er** en vokal, springes til 790, hvor V sættes lig med 1 og proceduren afsluttes (800). Det er her benyttet, at man kan anbringe RETURN hvor som helst i et underprogram. Denne facilitet skal anvendes med nogen forsigtighed. Hvis man senere ændrer i programmet kan man risikere, at man kommer til at indsætte sætninger, der aldrig bliver udført! Det her nævnte underprogram er dog så kort og overskueligt, at der næppe er nogen fare på færde. Som regel er det sikrest at lade alle udhop gå til sidste linje i underprogrammet, der består af RETURN-sætningen.
Det skal dog også bemærkes, at nogle BASIC-versioner kan anvende tal-variable som Boolske variable på samme måde som COMAL. I sådanne versioner vil man kunne skrive sætningen i linje 480 således:

```
IF NOT O1 THEN 510
```

Det kan måske forekomme at være en tvivlsom gevinst, da man alligevel ikke uden videre kan se, hvad O1 står for. I BASIC-versioner med Boolske operationer kan linjerne 460 og 470 slås sammen til:

```
IF T$<"A" OR T$<"Z" THEN 600
```

```
LIST
0010 PROC INITIAL
0020   REM (*INITIALISERING*)
0030   DIM VOKAL$(10),TEGN$(1)
0040   LET TRUE=1; FALSE=0
0050   LET VOKAL$="AEIOUY"
0060 ENDPROC INITIAL
0070 REM //-----------------//
0080 PROC INDTEKST
0090   DIM TXT$(2000),LINJE$(80)
0100   PRINT "INDTAST TEKSTEN: "
0110   PRINT
0120   INPUT "> ",LINJE$
0130   WHILE LINJE$<>"SLUT" DO
0140     LET TXT$=TXT$,LINJE$," "
0150     INPUT "> ",LINJE$
0160   ENDWHILE
0170 ENDPROC INDTEKST
0180 REM //---------------//
0190 PROC TAELOP
0200   LET ANTORD=0; ANTVOK=0; ANTKONS=0; ANTCIF=0; ANTSPEC=0
0210   LET ORDSLUT=TRUE
0220   FOR NR=1 TO LEN(TXT$)
0230     LET TEGN$=TXT$(NR)
0240     EXEC ANALYSE
0250   NEXT NR
0260 ENDPROC TAELOP
0270 REM //---------------//
0280 PROC ANALYSE
0290   REM (*BOGSTAV?*)
0300   IF "A"<=TEGN$ AND TEGN$<="Z" THEN
0310     IF ORDSLUT THEN LET ORDSLUT=FALSE; ANTORD=ANTORD+1
0320     EXEC VOKAL
0330     IF VOKAL THEN
0340       LET ANTVOK=ANTVOK+1
0350     ELSE (*SAA MAA DET VAERE EN KONSONANT*)
0360       LET ANTKONS=ANTKONS+1
0370     ENDIF
0380   ELSE (*ALTSAA IKKE NOGET BOGSTAV*)
0390     LET ORDSLUT=TRUE
0400     IF TEGN$<>" " THEN
0410       REM (*CIFFER?*)
0420       IF "0"<=TEGN$ AND TEGN$<="9" THEN
```

```
0430          LET ANTCIF=ANTCIF+1
0440        ELSE (*SAA MAA DET VAERE ET SPECIALTEGN*)
0450          LET ANTSPEC=ANTSPEC+1
0460        ENDIF
0470      ENDIF
0480    ENDIF
0490 ENDPROC ANALYSE
0500 REM //---------------//
0510 PROC VOKAL
0520   LET J=0
0530   REPEAT (*SØG I MAENGDEN AF VOKALER*)
0540     LET J=J+1
0550     LET VOKAL=(TEGN$=VOKAL$(J))
0560   UNTIL VOKAL OR J=6
0570 ENDPROC VOKAL
0580 REM //---------------//
0590 PROC UDDATA
0600   PRINT
0610   PRINT "ANTAL ORD:                              ";ANTORD
0620   PRINT "ANTAL VOKALER:                          ";ANTVOK
0630   PRINT "ANTAL KONSONANTER:                      ";ANTKONS
0640   PRINT "ANTAL SPECIALTEGN:                      ";ANTSPEC
0650   PRINT "ANTAL CIFRE:                            ";ANTCIF
0660   PRINT
0670 ENDPROC UDDATA
0680 REM //---------------//
0690 REM (*HOVEDPROGRAM*)
0700 EXEC INITIAL
0710 EXEC INDTEKST
0720 EXEC TAELOP
0730 EXEC UDDATA
0740 END (*SLUT PROGRAM TEKSTSTAT*)
```



```
LIST
0010 REM ** STYREPROGRAM **
0020 REM ** UDFØR INITIALISERING **
0030 GOSUB 0120
0040 REM ** UDFØR INDTEKST **
0050 GOSUB 0180
0060 REM ** UDFØR OPTAELLING **
0070 GOSUB 0290
```

```
0080 REM ** UDFØR UDDATA **
0090 GOSUB 0830
0100 END
0110 REM //---------------//
0120 REM ** INITIALISERING **
0130 DIM V$(10),T$(1)
0140 LET V$="AEIOUY"
0150 RETURN
0160 REM ** INIT SLUT **
0170 REM //---------------//
0180 REM ** INDTEKST **
0190 DIM T1$(2000),L$(80)
0200 PRINT "INDTAST TEKSTEN: "
0210 PRINT
0220 INPUT L$
0230 IF L$="SLUT" THEN GOTO 0270
0240 LET T1$=T1$,L$," "
0250 GOTO 0220
0260 RETURN
0270 REM ** SLUT INDTEKST **
0280 REM //---------------//
0290 REM ** TAELOP **
0300 LET A1=0
0310 LET A2=0
0320 LET A3=0
0330 LET A4=0
0340 LET A5=0
0350 LET O1=1
0360 FOR N=1 TO LEN(T1$)
0370   LET T$=T1$(N)
0380   REM ** UDFØR ANALYSE **
0390   GOSUB 0440
0400 NEXT N
0410 RETURN
0420 REM ** SLUT TAELOP **
0430 REM //---------------//
0440 REM ** ANALYSE **
0450 REM ** BOGSTAV? **
0460 IF T$<"A" THEN GOTO 0600
0470 IF T$>"Z" THEN GOTO 0600
0480 IF O1=0 THEN GOTO 0510
0490 LET O1=0
0500 LET A1=A1+1
```

```
0510 REM ** VOKAL? **
0520 GOSUB 0730
0530 IF V=0 THEN GOTO 0560
0540 LET A2=A2+1
0550 GOTO 0580
0560 REM ** ALTSAA EN KONSONANT **
0570 LET A3=A3+1
0580 REM ** SLUT VOKAL ELLER KONSONANT **
0590 GOTO 0700
0600 REM ** ALTSAA IKKE NOGET BOGSTAV **
0610 LET O1=1
0620 IF T$=" " THEN GOTO 0700
0630 REM ** CIFFER? **
0640 IF T$<"0" THEN GOTO 0680
0650 IF T$>"9" THEN GOTO 0680
0660 LET A4=A4+1
0670 GOTO 0700
0680 REM ** MAA VAERE ET SPECIALTEGN **
0690 LET A5=A5+1
0700 RETURN
0710 REM ** SLUT ANALYSE
0720 REM //---------------//
0730 REM ** VOKAL **
0740 FOR J=1 TO 6
0750   IF T$=V$(J) THEN GOTO 0790
0760 NEXT J
0770 LET V=0
0780 RETURN
0790 LET V=1
0800 RETURN
0810 REM ** SLUT VOKAL **
0820 REM //---------------//
0830 REM ** UDDATA **
0840 PRINT
0850 PRINT "ANTAL ORD:                ";A1
0860 PRINT "ANTAL VOKALER:            ";A2
0870 PRINT "ANTAL KONSONANTER:        ";A3
0880 PRINT "ANTAL CIFRE:              ";A4
0890 PRINT "ANTAL SPECIALTEGN:        ";A5
0900 PRINT
0910 RETURN
0920 REM ** SLUT UDDATA **
0930 REM //---------------//
```

Håndbogen er færdig,
15 måneder tog det at komme igennem,
og nu har abonnenterne et opslags-
værk, som man kan læse og få
gavn af igen og igen.

Dette betyder naturligvis ikke at vi er
færdige med data.
Stoffet vil blive behandlet i special-
afsnit af vort månedsblad
Populær Elektronik.

I PE fortsætter igangværende stof,
og nye spændende artikler
begynder.

Læs om hvordan man udbygger sit stereoanlæg med en ultrabas eller hvilke muligheder der er for at bygge sig et komplet højttalersystem.
Læs om hvordan man kan lave en omdrejningstæller eller en tyverialarm.
Læs om Walkie-Talkie.
LÆS
ELEKTRONIK–ÅRBOGEN

POPULÆR ELEKTRONIK's STORE ELEKTRONIK ÅRBOG

ELEKTRONIK–ÅRBOGEN henvender sig til folk der kan li' at bygge højttalere og elektroniske konstruktioner samt rode med walkie talkie.
Bogen indeholder interessante højttalerforslag med flere forskellige ultrabasser, sidesystemer og sammensætning af optimale løsninger..
Elektronikkonstruktionerne omfatter små såvel som større opgaver.
Desuden er der Walkie-stof af den kendte ekspert på området Jørgen Weiberg....

Højttalerkonstruktioner med 18", 15", 13" og 8" basenheder.
Lukkede kabinetter basreflex og transmissionline konstruktioner.
Systemer der svinger fra en byggepris på 200 kr til 12000 kr.
Elektronikkonstruktioner med den kendte printservice.

49⁸⁵

Ønskes bogen på efterkrav, så ring til os.
Efterkravsgebyr: 6 kr.