DAITA 2

UITGAVE VAN DE HCC–DAI GEBRUIKERSGROEP

***** VOORWOORD *****

In dit nummer ontbreken de mooie hoofdjes zoals deze in de vorige DAITA hebben gestaan. De vele, maar vooral omvangrijke, bijdragen zijn daar schuldig aan. Daar kunnen we natuurlijk alleen maar blij om zijn. We hopen dat dit zo door mag gaan.
Wie heeft er leuke basic-programma's? Stuur ze aan het redactieadres:
G.J.W.Bremer, Vleugeltjesbloem 21, 1902 GH Castricum, Tel. 02518-51878.

Nog even iets over de lay out:
1. Houd links en rechts een marge van 2 cm aan. Dat is bij 12 kar./inch 80 kar. en bij 10 kar./inch 67 kar..
2. Houd een hoogte aan van max. 26,3 cm = 62 geprinte regels van 1/6 inch.
3. Gebruik vers lint, zodat de afdrukken goed zwart worden.
4. Zet in uw bijdrage steeds naam, adres, woonplaats en telefoonnummer.

***** INHOUD *****

****

En, waar is het is het stopkontakt?

********** DAI ITS SCREEN ON THE NEC PC-8023B-C MATRIXPRINTER **********

Dirk van der Werf
Neptunuslaan 10
1702 BN HEERHUGOWAARD
02207-14182

1) Het programma bestaat uit ;

a) een deel in machinetaal .
b) een deel in BASIC .

2) Het deel in machinetaal ;

Taak ;
- het versturen van controlcodes (zodra dit noodzakelijk is) naar de NEC printer .
- het maken van een kopie van het beeldscherm op papier.

Opmerkingen ;
- na de het maken van de scherm kopie is de printer weer in de situatie zoals voor het maken van de schermkopie .
- Bij voorkeur dient men te werken in vierkleuren modes.
- tekeningen of andere grafische weergaven in 'lagere' modes geven kleinere afdrukken op papier dan 'hogere'.

3) Het deel in BASIC ;

Taak ;
- het samenstellen van een string met controlcodes voor de printer .
- het al dan niet aanroepen van het deel in machinetaal indien dit programma al dan niet aanwezig is .
- het geven van een errormelding indien het programma in machinetaal niet is geladen of indien het scherm in mode 0 is .

Opmerkingen ;
- de string met controlcodes bevat in ieder geval XMAX+1 en codes voor de verticale spatiering .
- informatie over de string met controlcodes vindt men in de sourcelisting van het deel in machinetaal .

4) Het gebruik ;

a) Laad het programma in machinetaal ;
```
-UT
-Z3
-R
-B
```
b) Laad het BASIC deel ;
```
-NEW
-LOAD
```
c) Mergen met andere programma's en aanroepen in deze programma's door GOSUB 60000

5) Opmerkingen ;

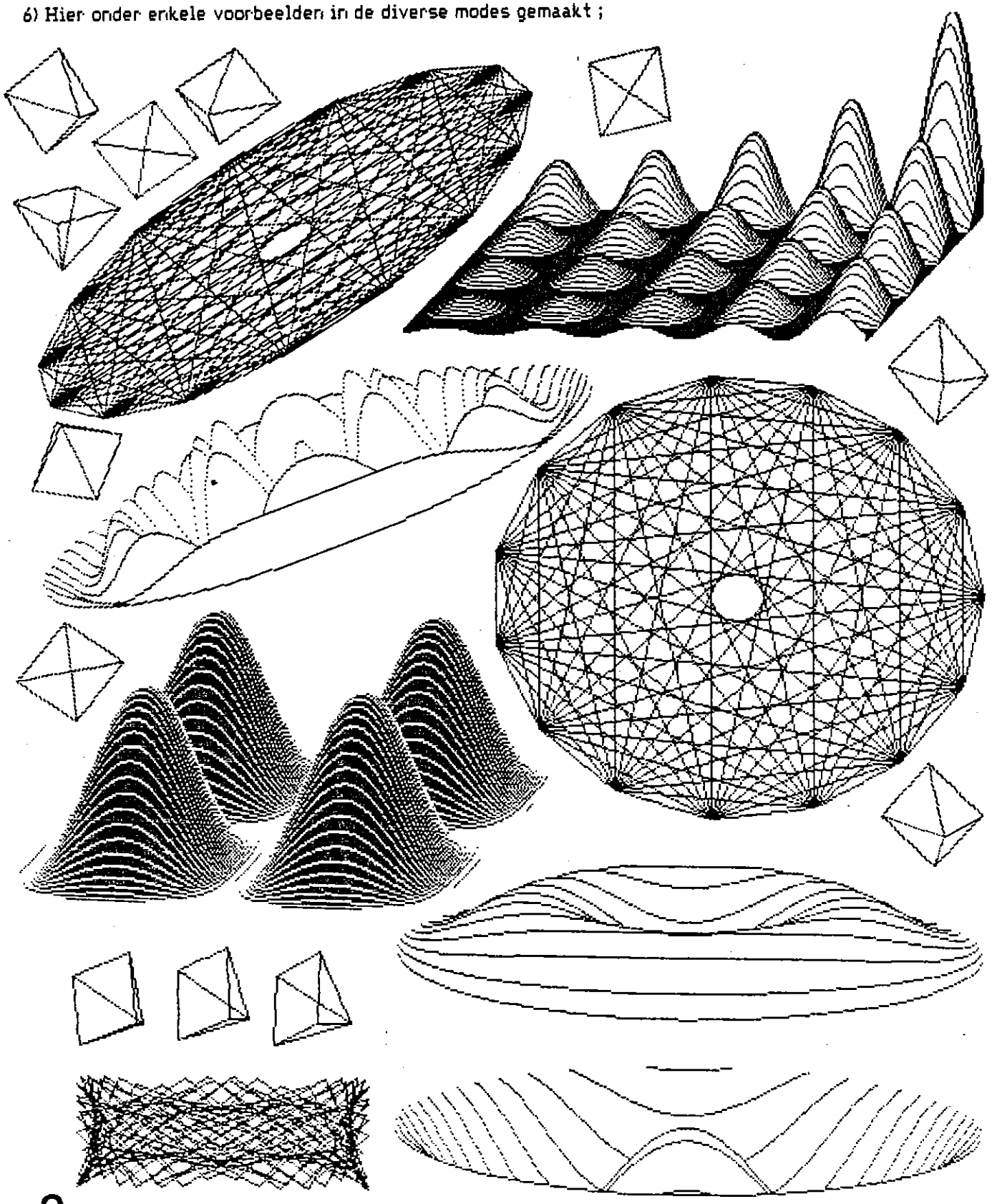
a) Bij RESET het volgende in UT herstellen ;
- S29B -30 -05
- S2DD -C3 -00 -04
- B
- NEW
- Programma opnieuw laden .

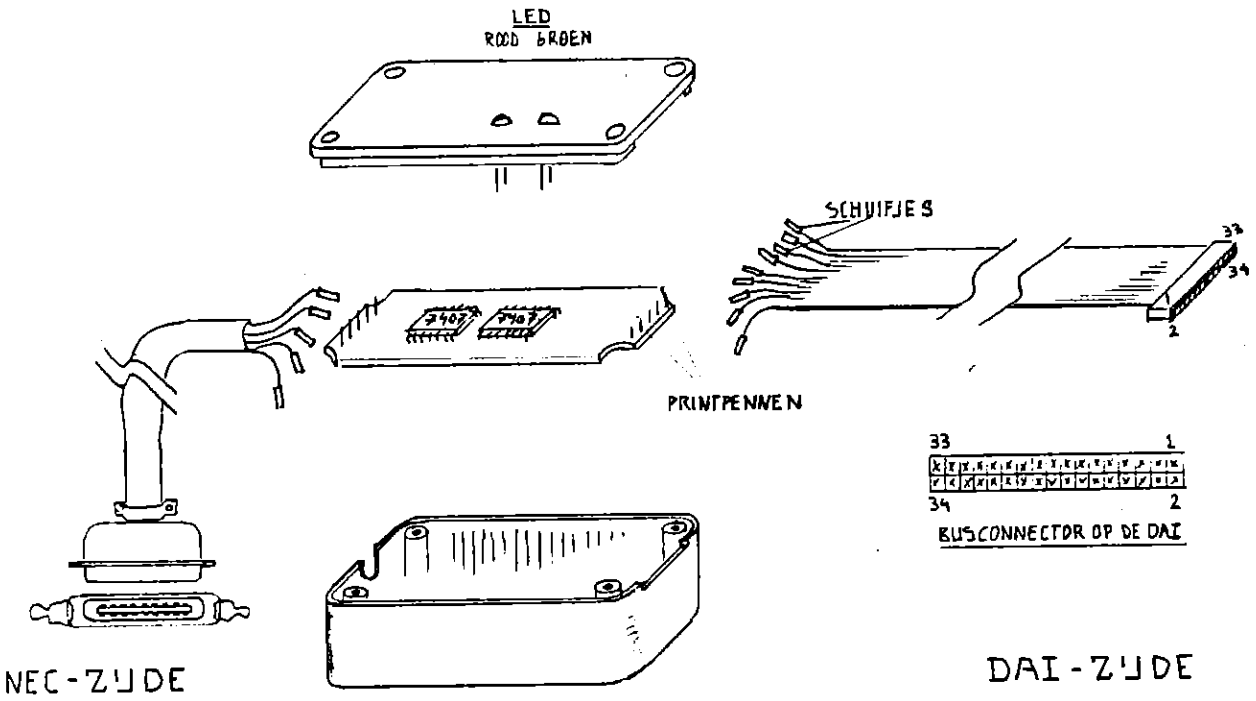
********** DAI ITS SCREEN ON THE NEC PC-8023B-C MATRIX PRINTER **********

5) Opmerkingen (vervolg) ;

b) POKE #131,3 in direct mode of in BASIC maakt het mogelijk om prints en listings op de matrix printer uit te voeren.
c) POKE #131,1 doet deze optie teniet .

6) Hier onder enkele voorbeelden in de diverse modes gemaakt ;





CONNECTOR		7404 (7=GND 14=+5V)		CONNECTOR DCE-BUS	CONNECTOR PIN
2	DATA 1	2 → 1	DATA 1	PØBØ	16
3	DATA 2	4 → 3	DATA 2	PØB1	14
4	DATA 3	6 → 5	DATA 3	PØB2	12
5	DATA 4	8 → 9	DATA 4	PØB3	10
6	DATA 5	10 → 11	DATA 5	PØB4	9
7	DATA 6	12 → 13	DATA 6	PØB5	11
8	DATA 7	2 → 1	DATA 7	PØB6	13
9	DATA 8	4 → 3	DATA 8	PØB7	15
1	$\overline{\text{STROBE}}$	6 → 5	$\overline{\text{STROBE}}$	P1BØ	30
	GND		+5V		
32	$\overline{\text{FAULT}}$		+5V		
36	INPUT BUSY		INPUT BUSY	P2B1	27
—	+5V		+5V		1
16/17	GND		GND		4

```
1     FFFF          ;
2     FFFF          ;*************************************************
3     FFFF          ; Dit programma creeert een schermkopie van een    ;
4     FFFF          ; grafische tekening. Het behoort bij een BASIC    ;
5     FFFF          ; gedeelte en is van daaruit aan te roepen met     ;
6     FFFF          ; CALLM SCRCOPY,CM$. SCRCOPY=#450 en CM$ bevat een ;
7     FFFF          ; aantal instructies om de printer te initialiseren;
8     FFFF          ;*************************************************
9     FFFF          ;
10    FFFF          ;*************************************************
11    FFFF          ; ORG 400 brengt character zowel op het scherm als ;
12    FFFF          ; op de printer. Initialiseer eerst:               ;
13    FFFF          ; #2DD-#C3, #2DE-#00, #2DF,#04; #131,#03           ;
14    FFFF          ;*************************************************
15    FFFF          ;
16    FFFF                  ORG       400
17    0400          ;
18    0400          GICA    EQU       0FE00       Data versturen naar printer
19    0400          GICB    EQU       0FE01       Strobe adres
20    0400          GICC    EQU       0FE02       Input busy adres
21    0400          CWGIC   EQU       0FE03       Controlword GIC adres
22    0400          COPCW   EQU       61          Copy van CWGIC
23    0400          ;
24    0400 E5       PRPRSC  PUSH H                Save gebruikte regs
25    0401 F5               PUSH PSW
26    0402 216100           LXI H     COPCW
27    0405 3E89             MVI A     89          A:=Controlwrd MODE 0 (A=uit)
28    0407 BE               CMP M
29    0408 C42B04           CNZ       INIT        Indien nog niet geinit
30    040B F1               POP PSW               Character klaar voor verzenden
31    040C CD1304           CALL      TIMING      Verzend
32    040F EF               RST 5                 Ook op het scherm
33    0410                  DATA      3
34    0411 E1               POP H
35    0412 C9               RET
36    0413          ;
37    0413 D5       TIMING  PUSH D
38    0414 F5               PUSH PSW              H lost
39    0415 2102FE           LXI H     GICC        Voor input busy
40    0418 7E       LOOP    MOV A,M
41    0419 E602             ANI       2
42    041B C21804           JNZ       LOOP
43    041E 2101FE           LXI H     GICB
44    0421 F1               POP PSW
45    0422 3200FE           STA       GICA
46    0425 3600             MVI M     0           Strobe signaal
47    0427 3601             MVI M     1
48    0429 D1               POP D
49    042A C9               RET
50    042B          ;
51    042B 3203FE   INIT    STA       CWGIC
52    042E 77               MOV M,A               COPCW:=CWORD
53    042F 3E01             MVI A     1           Strobe hoog
54    0431 3201FE           STA       GICB
55    0434 C9               RET
```

```
0435          ;
0435          ;*************************************************
0435          ; Deze routine brengt  alleen  characters  naar  de;
0435          ; printer. Vooral door SCRCPY gebruikt.           ;
0435          ;*************************************************
0435          ;
0435 E5       PRPR    PUSH H
0436 F5               PUSH PSW
0437 216100           LXI H       COPCW
043A 3E89             MVI A       89
043C BE               CMP M
043D C42B04           CNZ         INIT
0440 F1               POP PSW
0441 CD1304           CALL        TIMING
0444 E1               POP H
0445 C9               RET
0446          ;
0446          ;*************************************************
0446          ; Deze routine is het hoofdprogramma van SCRCPY.  ;
0446          ; Er word een kopie gemaakt van het grafisch scherm;
0446          ; Aanroep zoals in de kop vermeld                 ;
0446          ;*************************************************
0446          ;
0446                  ORG         450
0450          ;
0450          VART    EQU         3FF         Start van variabelen tabel
0450          CM1$    EQU         VART-1      Adres met adres van
0450          CM2$    EQU         VART-3      commandstring
0450          RESTY1  EQU         VART-5      REST en Y1 afz gebruikt
0450          Y2Y     EQU         VART-7      Y2 en Y eveneens
0450          BGND    EQU         VART-9      Aangepaste achtergrondkleur
0450          BYTE    EQU         VART-0A
0450          ADBGND  EQU         9E          Adres van de achtergrond
0450          ;
0450 E5       SCRCPY  PUSH H                  Save regs
0451 D5               PUSH D
0452 C5               PUSH B
0453 F5               PUSH PSW
0454 5E               MOV E,M                 HL wijst naar string .....
0455 23               INX H                   pointer !
0456 56               MOV D,M
0457 EB               XCHG                    Nu wijst HL naar string
0458 23               INX H                   CM1$ bevindt zich na het 1
0459 22FE03           SHLD        CM1$        van CM$ uit .. BASIC
045C 7E               MOV A,M
045D 85               ADD L
045E 6F               MOV L,A
045F 3E00             MVI A       0           Status onberoerd
0461 8C               ADC H
0462 67               MOV H,A
0463 23               INX H                   CM2$=CM1$+lengte CM1$+1
0464 22FC03           SHLD        CM2$
0467 AF               XRA A                   Voor de zekerheid
0468 4F               MOV C,A
0469 67               MOV H,A
046A 6F               MOV L,A
046B EF               RST 5                   Vraag naar YMAX
046C                  DATA        27          B:=YMAX
046D 78               MOV A,B
046E D607             SUI         7           Start initialisatie
```

```
116  0470 6F          MOV L,A                    Y1:=YMAX-7
117  0471 3E07        MVI A      7
118  0473 67          MOV H,A                    REST:=7
119  0474 22FA03      SHLD       RESTY1
120  0477 22F803      SHLD       YZY
121  047A 3A9E00      LDA        ADBGND
122  047D E60F        ANI        F               Kleur=8X
123  047F 32F603      STA        BGND
124  0482 CDFD04      CALL       PR1$            Einde initialisatie
125  0485 CD9004      CALL       COPY
126  0488 CD0405      CALL       PR2$            Printer in norm stand
127  048B F1          POP PSW
128  048C C1          POP B
129  048D D1          POP D
130  048E E1          POP H
131  048F C9          RET
132  0490        ;
133  0490        ;************************************************
134  0490        ; In de volgende routine worden een aantal namen ge;
135  0490        ; bruikt ter verduidelijking. Het beeld wordt van  ;
136  0490        ; links naar rechts en van boven naar beneden in   ;
137  0490        ; groepen van 8 dots (van beneden naar boven)      ;
138  0490        ; gescand.                                         ;
139  0490        ; REST :aantal dots dat in een groep wordt gescand ;
140  0490        ;       normaal REST=7 ; laatste regel rest<7      ;
141  0490        ; X    :de actuele X (stap=1)                      ;
142  0490        ; Y    :de actuele Y (stap=1 of -8)                ;
143  0490        ; Y1   :de startwaarde van Y (stap=-8)             ;
144  0490        ; Y2   :deze bevat het aantal, nog in een groep te ;
145  0490        ;       bepalen dots of bits.                      ;
146  0490        ; BYTE :bevat het te versturen BYTE                ;
147  0490        ; De NEC verwacht een byte alsvolgt:               ;
148  0490        ; o LSB          een BYTE reeks    o   o           ;
149  0490        ; o              FF 88 FF 32       o   o o         ;
150  0490        ; o              geeft op          o   o           ;
151  0490        ; o              papier dus        o o o           ;
152  0490        ; o                                o   o o         ;
153  0490        ; o                                o   o o         ;
154  0490        ; o                                o   o           ;
155  0490        ; o MSB                            o o o           ;
156  0490        ; In CM$ uit BASIC ligt vast hoe lang de bytereeks ;
157  0490        ; is. In MODE 6 bijvoorbeeld 335=XMAX              ;
158  0490        ; Voorbeeld CM$ uit BASIC:                         ;
159  0490        ; 0F Er volgen 15 characters in CM$                ;
160  0490        ; 0B ER volgen 11 characters in CM1$               ;
161  0490        ; 0D CR                                )     )     ;
162  0490        ; 1B ESC                               )     )     ;
163  0490        ; 54 )Line spacing 16/144 inch         )     )     ;
164  0490        ; 31 )                                 )     )     ;
165  0490        ; 36 )                                 )     )     ;
166  0490        ; 1B ESC                               )CM1$ )     ;
167  0490        ; 53 Bit image mode                    )     )     ;
168  0490        ; 30 )Er volgen 336 bytes, waarna      )     )     ;
169  0490        ; 33 )de printer weer gereed is voor   )     )     ;
170  0490        ; 33 )nieuwe data.                     )     )CM$  ;
171  0490        ; 36 )                                 )     )     ;
172  0490        ; 03 Er volgen 3 characters in CM2$          )     ;
173  0490        ; 1B ESC                               )     )     ;
174  0490        ; 41 Linespacing 1/6 inch=66 regels pp )CM2$ )     ;
175  0490        ; 0D CR                                )     )     ;
176  0490        ;************************************************
177  0490        ;
```

```
178  0490 210000  COPY    LXI H      0
179  0493 E5      BYTES   PUSH H                 Save X
180  0494 2AF803          LHLD       Y2Y        DE:=Y2Y
181  0497 EB              XCHG
182  0498 AF              XRA A                  BYTE:=0
183  0499 32F503          STA        BYTE
184  049C 4B      BITS    MOV C,E                C:=Y
185  049D 21F603          LXI H      BGND        B:=BGND
186  04A0 46              MOV B,M
187  04A1 E1              POP H                  X Terug
188  04A2 D5      BIT     PUSH D                 Save Y2Y
189  04A3 C5              PUSH B                 Save BGND
190  04A4 EF              RST 5
191  04A5                 DATA       27          Get COL in ACCA
192  04A6 C1              POP B
193  04A7 D1              POP D
194  04A8 DAC304          JC         NEXTY1      Next line
195  04AB B8      NEXTY   CMP B                  COL-BGND
196  04AC CCDF04          CZ         BIT0
197  04AF C4E904          CNZ        BIT1
198  04B2 1C              INR E                  Y:=Y+1
199  04B3 E5              PUSH H                 Save X
200  04B4 15              DCR D                  Y2:=Y2-1
201  04B5 F29C04          JP         BITS
202  04B8 3AF503          LDA        BYTE
203  04BB CD3504          CALL       PRPR        Byte versturen
204  04BE E1              POP H
205  04BF 23      NEXTX   INX H
206  04C0 C39304          JMP        BYTES
207  04C3 2AFA03  NEXTY1  LHLD       RESTY1      HL:=RESTY1
208  04C6 7D              MOV A,L                A:=Y1
209  04C7 A7              ANA A
210  04C8 C8              RZ                     Klaar als Y1=0
211  04C9 D608            SUI        8
212  04CB D2D204          JNC        CONT1       IF Y1>=0
213  04CE 84              ADD H                  IF Y1<0
214  04CF 3C              INR A
215  04D0 67              MOV H,A                REST:=REST-Y1
216  04D1 AF              XRA A                  Y1:=0
217  04D2 6F      CONT1   MOV L,A                Y1:=new Y1
218  04D3 22FA03          SHLD       RESTY1
219  04D6 22F803          SHLD       Y2Y
220  04D9 CDFD04          CALL       PR1$
221  04DC C39004          JMP        COPY
222  04DF         *
223  04DF 3AF503  BIT0    LDA        BYTE        Shift byte 1* left
224  04E2 37              STC
225  04E3 3F              CMC
226  04E4 17              RAL                    Ao:=0
227  04E5 32F503          STA        BYTE
228  04E8 C9              RET
229  04E9         *
230  04E9 3AF503  BIT1    LDA        BYTE        Shift byte 1* left
231  04EC 37              STC
232  04ED 17              RAL                    Ao:=1
233  04EE 32F503          STA        BYTE
234  04F1 C9              RET
235  04F2         *
236  04F2 56      PR$     MOV D,M                D:=lengte string
237  04F3 23      NEXT    INX H
238  04F4 15              DCR D                  ENTRY PUSH H,CM$
```

```
239   04F5 F8              RM
240   04F6 7E              MOV A,M
241   04F7 CD3504          CALL      PRPR
242   04FA C3F304          JMP       NEXT
243   04FD           ;
244   04FD 2AFE03  PR1$    LHLD      CM1$          Entry CM1$
245   0500 CDF204          CALL      PR$
246   0503 C9              RET
247   0504           ;
248   0504 2AFC03  PR2$    LHLD      CM2$          Entry CM2$
249   0507 CDF204          CALL      PR$
250   050A C9              RET
251   050B                 END

ADBGND:009E BGND  :03F6 BIT   :04A2 BIT0  :04DF BIT1  :04E9
BITS  :049C BYTE  :03F5 BYTES :0493 CM1$  :03FE CM2$  :03FC
CONTI :04D2 COPCW :0061 COPY  :0490 CNGIC :FE03 GICA  :FE00
GICB  :FE01 GICC  :FE02 INIT  :042B LOOP  :0418 NEXT  :04F3
NEXTX :04BF NEXTY :04AB NEXTY1:04C3 PR$   :04F2 PR1$  :04FD
PR2$  :0504 PRPR  :0435 PRPRSC:0400 RESTY1:03FA SCRCPY:0450
TIMING:0413 VART  :03FF Y2Y   :03F8
```

****** DAI ITS SCREEN ON THE NEC PC-8023B-C MATRIX PRINTER ******

```
60000 REM ROUTINE SCREENCOPY
60001 IF PEEK(#413)<>#D5 OR PEEK(#450)<>#E5 THEN 60011
60002 IF PEEK(#9D)=#FF THEN 60012
60003 MAX$=STR$(XMAX+1.0)
60004 IF LEN(MAX$)=6.0 THEN MAX$=MID$(MAX$,1,3):GOTO 60006
60005 MAX$="0"+MID$(MAX$,1,2)
60006 CM1$=CHR$(#B)+CHR$(13)+CHR$(#1B)+"T16"+CHR$(#1B)+"S0"+MAX$
60007 CM2$=CHR$(#3)+CHR$(13)+CHR$(#1B)+"A"
60008 CM$=CM1$+CM2$
60009 CALLM #450,CM$
60010 RETURN
60011 PRINT "MLP NOT PRESENT !":RETURN
60012 PRINT "NO COPY IN MODE 0 !":RETURN
```

Dirk van der Werf
Neptunuslaan 10
1702 BN Heerhugowaard
02207-14182

***** HET ZETTEN VAN EEN BREAKPOINT *****

Bij het uittesten van machinetaalprogramma's is het handig om de executie op een bepaald moment te kunnen onderbreken. Dit kan met het zetten van een breakpoint. Er zijn daarvoor twee handelingen nodig:

1. Op HEX 1Ø komt: C3 ØC EA = JMP EAØC
2. Op het gewenste punt in het programma komt: D7 (RST 2)

Merk op:
a. Een stack overflow wordt nu ook als een breakpoint afgehandeld.
b. Het breakpoint (D7) moet door uzelf weer vervangen worden.
c. Het gebruik van de LOOK-functie en breakpoints gaat niet.

Op het moment dat een breakpoint bereikt wordt springt de monitor terug naar het "prompt"-teken en blijven de registerinhouden onveranderd.

************ door W.Nijland *************

SPL: EEN CONDITIONELE MACRO-ASSEMBLER VOOR DE DAI-pc

(Door werkgroep Sphynx)

Na vele jaren met plezier met de DAInamic-Assembler (DNA) gewerkt te hebben begonnen de beperkingen hiervan toch wel erg zwaar voor ons te wegen. Na diverse mogelijkheden onderzocht te hebben besloten we om zelf een assembler te gaan ontwikkelen. Dit zou weliswaar wel heel wat van onze vrije tijd gaan vergen, maar had als grote voordeel dat we er enige ideeen van ons over hoe een assembler zou moeten werken erin zouden kunnen toepassen: maatwerk dus. Na enige maanden vele uren achter de computer te hebben doorgebracht is deze assembler, die we SPL (afgeleid van System Programming Language) hebben gedoopt, zover dat hij wat we noemen "export-rijp" is, m.a.w.: we vermoeden dat we de andere DAI gebruikers er ook een plezier mee kunnen doen.
Om duidelijk te maken wat SPL allemaal kan en niet kan zullen we hem hieronder vergelijken met DNA. Het is niet onze bedoeling DNA af te gaan kraken, zoals al gezegd hebben ook wij lange tijd met plezier met DNA gewerkt, maar dit is, voor zover wij weten, de meest gebruikte assembler voor de DAI, dus levert goed vergelijkings materiaal. We weten dat u van ons, als zijnde de ontwerpers van SPL, waarschijnlijk geen onbevooroordeelde vergelijking zult verwachten, dus nodigen wij u uit om zelf op de DAI-bijeenkomsten SPL te testen en uzelf te overtuigen.

ALGEMENE KENMERKEN

DNA is een aangepaste versie van de NTA80 assembler van Northern Technology Books, en oorspronkelijk TTY (teletype) georienteerd, met van de karakters MSB=1. M.b.v. enkele conversie-routines is deze assembler aangepast aan de DAI (onder andere was het nodig voor de in- en uitvoer het MSB van de karakters om te zetten, omdat de DAI-editor karakters met MSB=1 niet accepteert, en zelf MSB=0 gebruikt). Een knap stukje werk, maar het maakt het geheel er niet fraaier (en sneller) op.
SPL is als het ware op het lijf van de DAIpc geschreven, al zal het waarschijnlijk wel mogelijk zijn het voor andere computers te vertalen. In dit opzicht is vooral het geintegreerde gebruik van de DAI-screen-editor (denk aan BASIC "EDIT") van belang, en dit zal bij conversie naar andere systemen dan ook wel het grootste struikelblok vormen. Dit zijn echter problemen voor toekomstige afgunstige gebruikers van andere computers die ook SPL willen gebruiken.
DNA gebruikt voor de source-files twee buffers, de eerste (3000 e.v.) voor het inlezen en wegschrijven van de files en als edit-buffer, de tweede (na deze buffer) voor de opslag van de source-code en (tijdens de "#"-routines) de symbol-table. Een voordeel van dit gescheiden buffer-systeem is dat de source-code niet zo snel wordt aangetast bij bijvoorbeeld merging, als de ingelezen source-file niet meer bij de oude source-file past. Verder maakt het de MSB-conversie (zie boven) wat gemakkelijker. De grenzen van deze buffers worden bij het opstarten van de assembler vastgelegd.
SPL gebruikt een enkele source-buffer, onderverdeeld in een source-code deel en een symbol-table. De scheiding tussen deze buffers ligt echter niet vast, maar bestaat uit het gebied tussen de naar boven toe uitbreidende source-code en de naar beneden toe uitbreidende symbol-table (die tijdens de invoer van de source-code wordt opgebouwd). Dit is tevens de ruimte die door de editbuffer wordt gebruikt. Bij het wegschrijven wordt de source-code tegen de symbol-table aangeschoven, waardoor er een compact blok ontstaat. Na het

inlezen wordt de nieuwe source-code naar onderen geschoven tot het begin van de source-buffer of, bij merging, het einde van de oude source-code. Het is hierbij mogelijk dat de ingelezen source-file te groot blijkt te zijn, zodat het een deel van de oude source-code overschrijft. Dit is een nadeel van het een-buffer systeem, maar als grootste voordeel geeft het een veel economischer geheugen-gebruik. Overigens is het tijdens het gebruik van SPL mogelijk het begin van de source te verschuiven zodat de source-buffer groter wordt, of juist kleiner om het ontwikkelde machinetaal-programma meer ruimte te geven.
DNA heeft zijn source-file in ASCII-formaat (MSB=1) staan, en stelt de symbol-table alleen tijdens de "#"-routines op. Het voordeel hiervan is een gemakkelijke in- en uitvoer. Verder is er geen vertaal-routine nodig naar ASCII-formaat.
SPL heeft zijn source-file in een speciale pseudo-code staan, met de labels e.d. in een aparte symbol-table, waar in de source-code m.b.v. pointers naar verwezen wordt. Dit vergt bij de in- en uitvoer wat meer vertaal-werk, maar het zorgt voor zeer compacte opslag en een snelle assemblage.
De verhouding source-code (inclusief symbol-table) / object-code (machinetaal programma) is bij DNA ongeveer 6 : 1, bij SPL ongeveer 5 : 2. Bij DNA is hierbij de tweede buffer niet bij de source gerekend. Bij SPL wordt de verhouding door het gebruik van veel commentaar snel ongunstiger, door het gebruik van veel absolute adressen (i.p.v. labels) gunstiger.
SPL kan maximaal een source-file van ongeveer 35K bewerken, wat een machinetaal-programma van zo'n 14K oplevert. Hierbij wordt dan wel de source aangetast, mag dit niet gebeuren dan kan SPL een machinetaalprogramma van ongeveer 10K produceren. Hierbij is geen rekening gehouden met het gebruik van DS ("RES") en macro's, die natuurlijk met een beperkte source veel object kunnen genereren.
DNA kan, bij een buffergrootte van 2*16K, een machinetaal-programma van zo'n 2.5 a 3K genereren. Volgens de geruchten bestaat er ook een versie waarbij de buffers niet even groot behoeven te zijn (/X.), maar die hebben wij nooit van DAInamic ontvangen.
Bij DNA schijnt het mogelijk te zijn d.m.v. het #O-commando een file weg te schrijven, die daarna m.b.v. een BASIC-programma als machinetaal-programma in het geheugen kan worden gezet, waardoor het mogelijk is ook op de plaats van de assembler een programma te plaatsen. Dit is bij ons echter nog nooit gelukt.
Bij SPL kan bij het assembleren een offset worden meegegeven waarna het machinetaal-programma d.m.v. het UTILITY M(ove)-commando op zijn plaats kan worden gebracht, of kan worden weggeschreven waarna het met een offset kan worden ingelezen.
DNA bevindt zich in een vrij laag deel van het geheugen (1100-2FFF) waardoor het gebruik van BASIC-programma's en MODE 3-6 geen bijzondere moeilijkheden geven.
SPL staat vrij hoog in het geheugen (ongeveer 8800-B300, juiste afmetingen bij het schrijven van dit artikel nog niet bekend), waardoor het gebruik van deze zaken wat moeilijker ligt. Op aanvraag zijn er echter aangepaste versies leverbaar, die laag in het geheugen staan.
DNA kent speciale mnemonics, zoals ZAR, JEQ, C< en STOP.
SPL kent deze mnemonics niet, omdat hij voor iedere mnemonic de standaard Intel-code (b.v. C9 voor RET) gebruikt, en dus bij het listen geen onderscheid kan maken tussen JEQ en JZ. Verder maakt hij gebruik van min of meer standaard assembler-directives, dus DB, DW en DS i.p.v.resp. DATA/ASC,

DBL en RES. Daarnaast kent hij extra directives, zoals MACRO en IF. Door de uitgebreide hoeveelheid assembler-directives zijn de MOV X,X-mnemonics (b.v. MOV A,A) vervallen, deze dienen als data d.m.v. DB te worden ingevoerd.
SPL assembleert gemiddeld ongeveer 2K machinetaalprogramma per seconde. De snelheid van DNA hebben we nog niet gemeten, maar is volgens ons belangrijk langzamer.
DNA kent als operatoren tussen de labels alleen de + en -. Omdat SPL met macro's en conditionele assemblage ging werken is het aantal operatoren uitgebreid tot optellen, aftrekken, vermenigvuldigen,delen,modulo,shift-left,shift-right,and,or en xor. In conditionele(-IF)-statements groter, kleiner, gelijk, ongelijk, AND en OR. Evaluatie vindt nu nog van links naar rechts plaats, misschien dat er in een latere versie van SPL haakjes en algebraische volgorde bijkomen.
DNA herkent hexa-decimale waarden aan het feit dat ze door een ":" worden voorafgegaan. Dit kan doordat die dubbele punt ook in de source-file wordt opgeslagen.
Bij SPL worden de nummerieke waarden direct als een- of twee-bytes getallen opgeslagen. Of ze in decimale, hexa-decimale, octale dan wel binaire vorm moeten worden gelist hangt af van de stand van de in- en uitvoervectoren. Achter de nummerieke waarde wordt een letter geplaatst die de stand van de vectoren aangeeft. Bij de invoer speelt dit ook; of iets als decimaal of octaal wordt beschouwd hangt af van dezelfde vectoren. De letter achter de nummerieke waarde wordt hierbij als extra controle gebruikt. Omdat hexa-decimale waarden niet door een ":" o.i.d. worden voorafgegaan moet er, indien ze met een letter (A-F) beginnen, een 0 voor worden geplaatst ter onderscheiding van labels.
De in- en uitvoer vectoren kunnen op twee manieren worden omgezet: ten eerste door de default-toestand te wijzigen door een commando, b.v. H voor hexa-decimale adressen; ten tweede m.b.v. de speciale directive (IO), gevolgd door de desbetreffende letter, waardoor tijdens de in- of uitvoer de vectoren worden omgezet. Dit speciaal met het oog op hard-copy.
Bij ASCII-strings (DNA: ASC) en bij 1-byte operands (b.v. CPI '0') kan men d.m.v. ' en " aangeven of het een karakter met MSB=0 resp. MSB=1 betreft. Verder kan men ook 1-byte operands vervangen door een label. Bij 2-byte operands kan men van $ gebruik maken, dat de huidige waarde van de "assemblage-pointer" aangeeft.

<u>COMMANDO'S</u>

SPL kent, naast de normale commando's zoals list (normaal, met adressen, symbol-table, fouten en hard-copy), edit, assembleer (ook met offset), wegschrijven-inlezen-mergen en Basic/Utility ook nog:
<u>Move:</u> verplaats gebied (zoals bij de meeste commando's aangeduid door regelnummers of labels) tot achter een andere regel of label. Een gewijzigde versie hiervan copieert het gebied, dus laat het ook op de oude plaats staan.
<u>Change:</u> verander de naam van een label, vervang een label (in de operand-kolom) door een nummerieke waarde of omgekeerd of wijzig de waarde van een nummerieke waarde.
<u>Sort:</u> sorteer de symbol-table in alfabetische volgorde. De symbol-table wordt opgebouwd in de volgorde waarin de labels in het programma worden ingevoerd. Bij merging is het echter noodzakelijk dat de symbol-tables in alfabetische volgorde staan (in SPL kan je dan ook geen source-file met ongesorteerde symbol-table wegschrijven). Verder worden bij het sorteren de overtollige labels (waar niet door de sour-

ce naar verwezen wordt, b.v. labels uit een ge"Kill"d gebied, een bekend probleem bij BASIC) uit de symbol-table verwijderd. Ongebruikte labels, dus labels die wel in de label-kolom voorkomen, maar niet als operand worden gebruikt, worden wel gemeld, maar niet verwijderd.
Find: kan op twee manieren worden gebruikt:
voor een hexadecimaal adres list het de source-regel waarvan de object-code op dat adres terecht komt. Dit is vooral handig bij het debuggen, in samenwerking met Utility L(ook), als men daar heeft gezien dat er op een bepaald adres iets gebeurt en men wil weten welke source-regel em mnemonic daarbij hoort.
voor een label list het de source-regel waar het (voor de eerste maal) in de label-kolom voorkomt en het geheugenadres waar het eerste byte van die regel in de source-file staat. Dit laatste is vooral handig indien het ontwikkelde machinetaalprogramma wat wij noemen "de source heeft opgeblazen". In de bij SPL geleverde documentatie staat een korte cursus "ophalen van gevalen steken", waarmee het vaak nog mogelijk is zo'n opgeblazen file te herstellen. In sommige gevallen is het natuurlijk handiger de back-up source-file in te lezen.
Verify: met behulp van dit commando en zijn tegenhanger (control) kan men de check-sum van assembler en/of source-file bepalen. Dit voor het geval dat men niet zeker weet wat het machinetaal-programma in ontwikkeling allemaal gaat doen, respectievelijk gedaan heeft, met source en assembler.
Size: dit print de volgende adressen: begin source-buffer / eind source-file / begin symbol-table / eind symbol-table (voor een bepaalde SPL-assembler een vast adres) / wegschrijf-adres file (begin adres source-code indien die tegen de symbol-table wordt aangeschoven, zoals bij het wegschrijven gebeurt) / minimum wegschrijf adres merge-file (van een te mergen file mag het wegschrijf-adres niet lager dan dit adres zijn geweest).
Cold: te vergelijken met "/" uit DNA, alleen wordt hier ook een nieuw begin-adres van de source-buffer gevraagd.
Warm: dit vraagt om een nieuw begin-adres van de sourcebuffer, en verschuift de source-code hierheen.

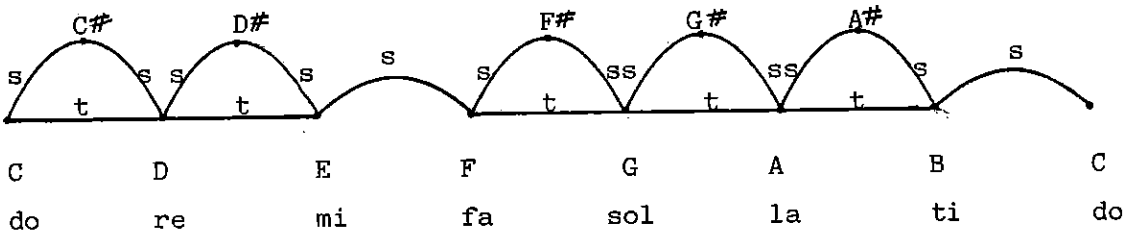
LEVERINGSCONDITIES

SPL wordt op cassette geleverd met op die cassette tevens de source-file van een disassembler die SPL-sourcefiles produceert (maakt gebruik van de SPL-assembler), alsmede een programma waarmee men DNA-source-files om kan zetten in SPL-formaat. Bij deze cassette wordt de nodige documentatie verstrekt. Speciale versies van SPL (b.v. SPL op lagere geheugen-adressen) worden, indien mogelijk, op verzoek geleverd met eventuele levertijd en meerprijs. Ditzelfde geldt voor vernieuwde SPL-versies. Om eventuele krakers een hoop moeite te besparen kan ook een source-file van SPL zelf en/of een listing geleverd worden. Het copy-right blijft natuurlijk voorbehouden aan werkgroep Sphynx. SPL wordt genummerd geleverd en de kopers komen op een afleverings-lijst.
De prijs van SPL (op geluids-cassette, gehaald of op bijeenkomsten) is f50,-. Verdere prijzen op aanvraag. Even ter vergelijking: DNA kost, tegen clubprijs, 1750 BFR (ongeveer f100,-) incl. verzendkosten; een Micro-soft assembler in de winkel kost ongeveer f125,-. Een beetje redelijk computerboek kost daar ongeveer f40,- a f80,-.
Voor nadere informatie en bestellingen: Werkgroep Sphynx, p/a Gebr.Rens, Grote Sloot 101, 1754 JC Burgerbrug.
Tel.nr. 02268-1788, 19.00-21.00 uur.

MUZIEK GENEREREN (vertaald uit de nieuwe DAI-manual)

Deze paragraaf beschrijft hoe de DAI geluidsgenerator gebruikt kan worden om muziek te maken. Het wiskundige verband tussen noten zal worden beschreven voor de Majeur toonladder. Een procedure voor het genereren van de toonladder in Majeur volgt hieronder.

MUZIEKNOTEN EN DE TOONLADDER IN MAJEUR



C	D	E	F	G	A	B	C
do	re	mi	fa	sol	la	ti	do

Het bovenstaande diagram laat de samenstelling zien van de toonladder in Majeur. De natuurlijke noten C, D, E enz. stellen de witte toetsen op een piano voor, terwijl de kruizen (#) de zwarte toetsen voorstellen.

Van de eerste C (links) naar de tweede C (rechts) noemen we een octaaf. Een octaaf stelt een 2:1 frequentieverhouding voor. M.a.w. om een willekeurige toon drie octaven hoger te verkrijgen, hebben we een toename in frequentie van acht maal de oorspronkelijke (toon)frequentie nodig, want $2*2*2=2\uparrow 3=8$

Om een toon van C naar C# te verhogen is het nodig dat de toon verhoogd wordt met een halve toon. Een halve toon stelt een verhoging in frequentie voor in een verhouding van $\sqrt[12]{2}:1$. D.w.z. $freq(C\#) = freq(C)*\sqrt[12]{2}$.

Het volgende programma zal de toonladder in Majeur demonstreren voor vijf octaven. De tonen kunnen vergeleken worden met tabel 1.

```
1ØØ   CLEAR 1ØØØ: SOUND OFF
11Ø   SEMI=2.Ø↑(1.Ø/12.Ø)
12Ø   LOWC=44Ø.Ø/(SEMI↑33.Ø)
13Ø   ENVELOPE Ø 15
14Ø   FOR N%=Ø TO 59
15Ø   IF N% MOD 12=Ø THEN RESTORE
16Ø   READ A$:FRQ=LOWC*(SEMI↑N%)
17Ø   SOUND Ø Ø 15 Ø FREQ(FRQ)
18Ø   PRINT A$, "FREQ=";FRQ
19Ø   WAIT TIME 2Ø :SOUND OFF
2ØØ   NEXT N%
21Ø   STOP
22Ø   DATA DO,DO#,RE,RE#,MI,FA,FA#,SOL,SOL#,LA,LA#,TI
```

TABEL 1

Noot	Freq. Hz Octaaf 1	Freq. Hz Octaaf 2	Freq. Hz Octaaf 3	Freq. Hz Octaaf 4	Freq. Hz Octaaf 5
DO	66	131	262	523	1Ø47
DO#	69	139	277	554	11Ø9
RE	73	147	294	587	1175
RE#	78	156	311	622	1245
MI	82	165	33Ø	659	1319
FA	87	175	349	698	1397
FA#	92	185	37Ø	74Ø	148Ø
SOL	98	196	392	784	1568
SOL#	1Ø4	2Ø8	415	831	1661
LA	11Ø	22Ø	44Ø	88Ø	176Ø
LA#	117	233	466	932	1865
TI	123	247	494	988	1976

Om de toon van C naar D te verhogen is het nodig dat deze wordt verhoogd met een hele toon. Een hele toon stelt een frequentieverhoging voor in de verhouding $\sqrt[6]{2}$:1. D.w.z. freq(D) = freq(C)*$\sqrt[6]{2}$
Opmerking: 1 hele toon= 2 halve tonen.
($\sqrt[6]{2} = \sqrt[12]{2} * \sqrt[12]{2}$)

HET GENEREREN VAN TOONLADDERS

Het is mogelijk om noten in een of meerdere toonladders te genereren. Dit door het verhogen of verlagen van de frequentie met het gewenste aantal halve tonen.
Voorbeeld:
Om van C naar F te gaan vereist een verhoging van 2 hele tonen + 1 halve toon = 5 halve tonen.

$$\text{freq(F)} = \text{freq(C)} * \sqrt[12]{2} * \sqrt[12]{2} * \sqrt[12]{2} * \sqrt[12]{2} * \sqrt[12]{2}$$
$$= \text{freq(C)} * (\sqrt[12]{2})^5$$

De basis (stem)toon van de toonladder is A=44Ø Hz. Door negen halve tonen naar beneden te gaan kan de standaard C gegenereerd worden. Het genereren van toonladders komt meestal neer op het omhoogschuiven van een geheel aantal halve tonen t.o.v. de standaard C.

Als slotvoorbeeld een programma om de gewone do, re, mi volgorde te laten horen:

```
1ØØ    CLEAR 1ØØØ:SOUND OFF
11Ø    SEMI=2.Ø↑(1.Ø/12.Ø)
12Ø    LOWC=44Ø/(SEMI↑9):REM C 9 SEMITONES DOWN
13Ø    ENVELOPE Ø 16
14Ø    FOR N%=1 TO 8
15Ø    READ D%
16Ø    FRQ=LOWC*(SEMI↑D%):REM D% SEMITONES UP FROM C
17Ø    SOUND Ø Ø 15 Ø FREQ(FRQ)
18Ø    WAIT TIME 2Ø:SOUND OFF
19Ø    NEXT N%
2ØØ    STOP
21Ø    DATA Ø,2,4,5,7,9,11,12
```

Spraaksynsthese

Het flexibele karakter van de DAI geluidsgenerator (de Intel 8253) maakt het aantrekkelijk om zelfs spraak te simuleren.
Het Basic van de DAI PC stelt de gebruiker in de gelegenheid om zelf te expirimenteren, maar de TALK faciliteit vergroot de snelheid en de flexibiliteit nog aanzienlijk.

Het aanroepen van het TALK statement.

```
TALK    BØØØ
TALK   VARPTR(X(Ø))
```

Het TALK commando in is feite een mini interpreter. De bovengenoemde aanroepen hebben tot gevolg dat de TALKcode interpreter start met het interpreteren van de speciale TALK codes op het gespecificeerde start-adres. De hieronder staande tabel geeft aan wat de verschillende TALK codes voor functie hebben.

Code	Aantal DATA bytes	Functie
0	2	Frequentie inst.kan Ø
2	2	,, ,, 1
4	2	,, ,, 2
8	1	Volume inst. kan Ø
9	1	,, ,, 1
A	1	,, ,, 2
B	1	,, ,, ruisgener.
C	2	Vertraging in 13 µS eenh.
D	2	Roep mach.taal routine
FF	-	Einde Talk routine.

Het instellen van een frequentie.
De gewenste frequentie wordt aangegeven door de twee DATA bytes volgend op de code Ø, 2 of 4. De berekening van de juiste DATA gaat als bij het SOUND-statement. Willen we b.v. 8ØØ Hz bepalen dan gaat dat als volgt: PRINT HEX$(FREQ(8ØØ)) , wat als antwoord geeft : 9C4.

Het instellen van het volume.
Dit wordt bereikt door het gebruik van de code 8, 9 of A, met daarop volgend een DATA byte. Er zijn 16 volume niveau's beschikbaar nl. 0- F.

Vertragen.
De code C en de daarop volgende twee DATA bytes bepalen hoelang een ingestelde frequentie moet blijven klinken. De eenheid van vertraging is 13 µS. De maximale vertraging is dus FFFF*13µS = ± 0.9 Seconden.

Indien de TALKcodes onvoldoende vrijheid mochten bieden, dan is er de mogelijkheid om met behulp van code D gevolgd door een (hexadecimaal) adres. Deze code biedt dus de mogelijkheid om machinetaalprogramma's aan te roepen zodat bepaalde stukjes TALK-programma herhaald kunnen worden.

Denk er wel aan dat de DATA, indien deze uit twee bytes bestaat, in de conventionele INTEL notatie moet worden ingevoerd.
Voorbeeld: Om het drie bytes TALKcommando D Ø123 uit te voeren moet er dus een machinetaalroutine op 23Ø1 staan !!

Beschouw het volgende voorbeeld van een TALK-programma:

Adres	TALK code	DATA	Functie
#AØØØ	Ø	#C4,9	Zet kanaal Ø op frequentie 8ØØ Hz (#9C4).
AØØ3	8	#F	Zet kanaal Ø op maximaal volume.
AØØ5	#C	#FF,#FF	Zet maximale vertraging (.88 sec.).
AØØ8	8	Ø	Zet kanaal Ø op minimaal volume.
AØØA	2	#A,#1A	Zet kanaal 1 op frequentie 3ØØ Hz (#1AØA).
AØØA	9	#F	Zet kanaal 1 op maximaal volume.
AØØF	#C	#FF,#FF	Zet maximale vertraging (.88 sec.).
AØ12	9	Ø	Zet kanaal 1 op minimaal volume.
AØ14	4	#2Ø,#4E	Zet kanaal 2 op frequentie 1ØØ Hz (#4E2Ø).
AØ17	#A	#F	Zet kanaal 2 op maximaal volume.
AØ19	#C	#FF,#FF	Zet maximale vertraging (.88 sec.).
AØ1C	#A	Ø	Zet kanaal 2 op minimaal volume.
AØ1E	#ØD	#21,#AØ	Spring naar machine code subroutine (#AØ21).

Als extra voorbeeld voegen we een kleine subroutine in machinetaal toe die terugkeert naar de TALK-interpreter en er voor zorgt dat de interpreter continueert op adres# AØØØ.

Adres	Code	
#AØ21	#21,#ØØ,#AØ	LXI,H met AØØØ
AØ24	#C9	Return

Het H,L register paar wordt geladen met het adres van de volgende instructie van de TALK-interpreter.

Het volgende basicprogramma voert het boven gegeven TALK-programma uit.

```
1Ø      CLEAR 1ØØØ:B%=#AØØØ
2Ø      READ A%:IF A%<Ø GOTO 4Ø
3Ø      POKE B%,A%:B%=B%+1:GOTO 2Ø
4Ø      TALK#AØØØ
5Ø      STOP
1ØØ     DATA Ø,#C4,9,8,#F,#C,#FF,#FF,8,Ø
11Ø     DATA 2,#A,#1A,9,#F,#C,#FF,#FF,9,Ø
12Ø     DATA 4,#2Ø,#4E,#A,#F,#C,#FF,#FF,#A,Ø
13Ø     DATA#ØD,#21,#AØ,#21,Ø,#AØ,#C9,-1
```

Een alternatief om het zelfde resultaat te verkrijgen, maar op een veiligere manier, is door ruimte te reserveren in de HEAP door een dummy array te dimensioneren en hierin de TALK codes te zetten.
Het volgende programma demonstreert dit.

```
5       REM IMP INT
1Ø      CLEAR  1ØØØ:DIM Q%(1ØØ):B%=VARPTR(Q%(Ø))
2Ø      READ A%:IF A%
```

```
5         REM IMP INT
1Ø        CLEAR 1ØØØ:DIM Q(1ØØ):B=VARPTR(Q(Ø))
2Ø        READ A:IF A<Ø GOTO 4Ø
3Ø        POKE B,A:B=B+1: GOTO 2Ø
4Ø        READ F:IF F<Ø GOTO 5Ø:READ S:GOSUB 1ØØØ:GOTO 4Ø
5Ø        TALK VARPTR(Q(Ø))
6Ø        STOP
1ØØ       DATA Ø,#C4,9,8,#F,#C,#FF,#FF,8,Ø
11Ø       DATA 2,#A,#1A,9,#F,#C,#FF,#FF,9,Ø
12Ø       DATA 4,#2Ø,#4E,#A,#F,#C,#FF,#FF,#A,Ø
13Ø       DATA #ØD,Ø,Ø,#21,Ø,Ø,#C9,-1
14Ø       DATA 32,34,35,1,-1

1ØØØ      J=4:UF=VARPRT(Q(F/J))+(F MOD 4)-1
1Ø1Ø      US=VARPRT(Q(S/J))+(S MOD 4)-1
1Ø2Ø      POKE UF,US IAND #FF
1Ø3Ø      POKE UF+1,US SHR 8
1Ø4Ø      RETURN
```

***** HET UITBREIDEN VAN DE DTP EDITBUFFER *****

Bij het maken van program a's in DAI Tiny Pascal kan het voorkomen dat de beschikbare editruimte te klein is. Hieronder volgt een publicatie om de editbuffer te vergroten.
Verander de volgende adressen in:
3ØØ7 - 1Ø 3Ø2F - 1Ø 4EB5 - 1Ø
5E7D - 1Ø 5EC2 - 1Ø 4EDB - 8F
Save dit programma voordat u het runt en geef het als label mee: PASCAL V3.1A.
Sourceprogramma's van derden, gecreëerd met V3.1 kunnen alleen via een ingreep verwerkt worden door V3.1A.
Verander ook adres 55E9 in: 55E9 - ØØ en 55EA - 9Ø.
Dit is nodig omdat anders een gedeelte van de editbuffer verloren gaat tijdens het vertalen.
Het vertaalde programma kan ook werken zonder het Pascal-programma op zich.
Dit gaat als volgt:

1. Schrijf het Pascal-programma;
2. Compileer het programma;
3. Vertaal het programma !!Let op!! met als start van de objectcode adres Ø9C5 (dit is standaard in het gewone programma) en noteer het eindadres van de vertaalde P-codes.
4. Ga naar Utility en schrijf en schrijf op cassette: WØ2EC (eindadres van de code's) (naam programma).

Voor het executeren van het programma:
R (naam programma);
Z3
GØ9C5
Veel succes!!

In de volgende utgave van DAITA komt de listing van "VIER OP EEN RIJ" in Pascal.

********** door T.Verberkt **********

```
Dit programma demonstreert Pascal (DTP) in een grafiekendemonstratie

Programma grafiek V2.2 13-6-1982!

VAR I,A,XX,X,Y,C,Y1,Y2,X1,X2,T,SP : INTEGER;

PROC MODE(M);
    BEGIN
    MEM[#7DØ]:=M; CALL(#7FD)
    END ! MODE !;

PROC CURSOR(X,Y);
    BEGIN
    MEM[#7D5]:=X; MEM[#7D6]:=Y; CALL(#93E)
    END;

PROC COLORT(C1,C2,C3,C4);
    BEGIN
    MEM[#7DØ]:=C1; MEM[#7D1]:=C2; MEM[#7D2]:=C3;
    MEM[#7D3]:=C4; CALL(#7DE)
    END;

PROC DRAW(X1,Y1,X2,Y2,C);
    BEGIN
    MEM[#7D3]:=X1 AND 255; MEM[#7D4]:=X1 SHR 8;
    MEM[#7D5]:=X2 AND 255; MEM[#7D6]:=X2 SHR 8;
    MEM[#7D1]:=Y1; MEM[#7D2]:=Y2; MEM[#7DØ]:=C;
    CALL(#886)
    END ! DRAW !;

FUNC XMAX;
    BEGIN
    CALL(#912); XMAX:=256 * MEM[#7D4] + MEM[#7D3]
    END;

PROC TEKEN ! GRAFIEK !;
    VAR X1,X2,Y1,Y2: INTEGER;
    BEGIN
    X1:=8Ø;
    X2:=A; Y1:=Ø; Y2:=1ØØ;
    DRAW(X1,Y1,X2,Y2,7);
    DRAW(X1-8Ø,Y1+5Ø,X2-X2+XMAX,Y2-A,7);
    DRAW(X1,Y1+1ØØ,A,Y2-1ØØ,7);
    DRAW(X1-X1+XMAX,Y1+5Ø,X2-X2,Y2-Y2+A,7)
    END;

PROC DOE;
  BEGIN
       MODE(4);A:=-3;
    REPEAT A:=A+3;TEKEN UNTIL A>3Ø
  END;
```

```
PROC WAITTIME(T);
   BEGIN
     MEM[#1BE]:=T AND 255;MEM[#1BF]:=T SHR 8;
     REPEAT UNTIL (MEM[#1BE]=Ø) AND (MEM[#1BF]=Ø)
   END;

BEGIN HOOFDPROGRAMMA!
  XX:=Ø
    REPEAT
      MODE (#FF);MEM[#75]:=32;
        FOR I:=1 TO 3 DO BEGIN
          WRITE(12);COLORT(8,Ø,8,8);
          MEM[#BA2D]:=#57;CURSOR (3,12);
          WRITE('GRAFIEK     ');
          T:=5Ø;WAITTIME(T);WRITE(12); MEM[#BA2D]:=#57;
          CURSOR(1,12);WRITE(' WRITE IN PASCAL ');
          WAITTIME(T);WRITE(12);MEM[#BA2D]:=#57;
          CURSOR(1,12);WRITE('BY THEO VERBERKT');
          WAITTIME(T)
       END;
      WAITTIME(125);DOE;WAITTIME(1ØØ)
    UNTIL XX=1
END.
```

***** DAI TINY PASCAL INTERPRETER *****

Voor DAI Tiny Pascal V3.1 is nu ook een eenvoudige interpreter beschikbaar. Een in Tiny Pascal geschreven interpreter, o.a. gepubliceerd in "the BYTE BOOK OF PASCAL", Peterborough, 1979, is door mij enigszins aangepast en met behulp van de compiler en de translator omgezet in een uitvoerbaar objectmodule, dat de adressen #09C5 - #1FF9 beslaat.
Met deze interpreter kunnen de P-codes van uw toepassingsprogramma, de stackpointer en de waarden van de stack bekeken worden. Er zijn maximaal 10 breakpoints mogelijk, terwijl de T(race)-optie u de laatste 22 (of minder) uitgevoerde P-codes toont.
Doordat de "runtime stack" van deze interpreter van #612B - #6FFF loopt, is het helaas niet mogelijk Pascal-programma's te interpreteren, die in grafische mode 8, 9, 10 of 11 (Basic: 5, 5A, 6 Of 6A) lopen. U zou dan bijv. met dummy procedures voor de tekenroutines kunnen werken.

********** door W.van Eck **********

```
##################################################
#                                                #
#              FORTH op de DAI                   #
#                                                #
#         Een nieuwe programeertaal.             #
#                                                #
##################################################
```

De tijden van programeren in het starre ongestructureerde basic kunnen binnenkort voorbij zijn. Voor de DAI komt er namelijk binnenkort een FORTH ter beschikking.

Wat is FORTH?

FORTH is een taal ontworpen om snel en gemakkelijk te kunnen programmeren, met een zo groot mogelijke controle over de computer. Het combineert geavanceerde structuren, zoals je die in ALGOL-68 tegenkomt met structuren die op het binnenste van de machine werken. Deze taal is daarom naar mijn mening tot nu toe de meest geschikte taal voor microcomputers zoals de DAI.
In de ideale situatie is FORTH het operating-system (bestuurssysteem), maar dan een die de mogelijkheden van PASCAL ver te boven gaat.
Maar ook wanneer je je niet met de machine zelf wilt bemoeien is FORTH de ideale taal op je computer. Snelheid wordt gecombineerd met compactheid. Je hebt een taal tot je beschikking waarin je met de meest geavanceerde structuren alles snel en effecient kunt programeren. Zijn de structuren die je wilt hebben er niet: je kunt ze er zelf bijmaken.
Samenvattend:voor FORTH geldt nog meer als in andere talen:-
IN FORTH KAN ALLES!!!!

Voordelen van FORTH

-Gestructureerd programeren.
-Uitbreidbaar. Alles, ook operatoren en controlestructuren.
-Grote controle over de machine. Forth kan zeer goed als operatingsystem fungeren.
-FORTH is zeer snel. Dit hangt o.a. samen met de "bedrade" manier van implementeren.
-FORTH compileert zeer compact.
-FORTH kent recursie.
-FORTH betekent snel programeren.
-Machinetaal kan gemengd worden met FORTH.
-Implementatie van andere hogere talen als PASCAL, LISP en SETL is in FORTH relatief eenvoudig (ook lagere talen als basic).

Nadelen van FORTH

-FORTH kan slecht leesbaar zijn door het stackgebruik (Wat doet: SWAP ROT SWAP ROT ?). Dit kan een kwestie van goed kommentaargebruik en lay-out zijn.
-RPN notatie. Gedeeltelijk een kwestie van wennen.
-Soms wat moeilijker te leren dan sommige andere talen. Dit hangt samen met de aparte manier van rekenen (RPN) en het

grote aantal woorden. Deze moeite is het resultaat zeker wel waard.
-Grote controle over de machine. Vooral bij multi-usersystemen en onervaren gebruikers kan dit een nadeel zijn.
-Geen strings en floating point. Door de uitbreidbaarheid is dit er gemakkelijk op te zetten. Er zijn hiervoor standaardpakketten.

De beschrijving die hieronder van FORTH gegeven wordt, is die van de nieuwe FORTH voor de DAI. In deze FORTH zitten veel dingen die in de 79-standaard niet kunnen. Standaard FORTH kan echter wel in deze FORTH gebruikt worden, op een paar uitzonderingen na (met een goede reden).

¥¥¥ Definities maken ¥¥¥

Een FORTH-systeem bevat een lijst van definities (subroutines) in de zgn. "Dictionary". Deze definities hebben een naam die uit alle characters, behalve de spatie mogen bestaan. Wanneer een naam wordt ingetyped (meerdere namen worden gescheiden door spaties), wordt de bijbehorende definitie na <RETURN> uitgevoerd.
Wanneer een reeks van definities uitgevoerd moet worden kan er een nieuwe definitie worden gemaakt die deze reeks uitvoert. Wanneer de naam van deze nieuwe definitie wordt aangeroepen, dan wordt deze reeks uitgevoerd.
Stel bijvoorbeeld dat het FORTH-systeem de volgende definities bevat:

```
JAN,PIET,SCHOOL,NAAR,GAAT
```

Als PIET naar school moet wordt de volgende zin ingetyped:

```
PIET GAAT NAAR SCHOOL
```

Voor JAN zou nu weer bijna dezelfde zin ingetyped moeten worden. Nu kan de volgende definitie gemaakt worden:

```
: NAARSCHOOL GAAT NAAR SCHOOL ;
```

De ":" geeft aan dat er een nieuwe definitie gemaakt gaat worden met als naam het woord erachter (spatie is scheider). De nieuwe definitie heet dus NAARSCHOOL.
De woorden daarna worden nu uitgevoerd als de nieuwe definitie wordt aangeroepen. De ";" geeft aan dat het maken van de definitie klaar is.
Als PIET en JAN naar school moeten kan nu worden gezegd:

```
PIET NAARSCHOOL ( De computer doet: PIET GAAT NAAR SCHOOL)
JAN NAARSCHOOL
```

De nieuwe definitie NAARSCHOOL kan nu zelf in een nieuwe definitie gebruikt worden. Dit geeft een van de voordelen van FORTH aan: Een ingewikkelde definitie kan gemakkelijk in een nieuwe ingewikelde definitie gebruikt worden, en die ook weer etc. Dit zorgt voor snel (exponentieel) programmeren. De uitvinder van FORTH, zegt dat hij in een jaar 1 goed Fortran (vergelijkbaar met basic) programma kan schrijven tegen 20 goede FORTH programma's.

*** De stack in FORTH ***

De definities in FORTH moeten op de een of andere manier met elkaar kunnen communiceren. Daarvoor is in FORTH gekozen voor het concept van de STACK.
Een stack kan worden voorgesteld als een stapel getallen. Het getal wat het laatst op de stack wordt gezet, komt er ook het eerste weer af (Last In First Out - LIFO).
De meeste FORTH-definities gebruiken getallen op de stack en/of laten getallen op de stack achter.

Een getal wat wordt ingetyped wordt op de stack gezet, de definitie . (punt) drukt het bovenste getal af. Kommentaar in FORTH staat tussen haakjes of achter \ (backslash):

```
1 2 3 4 5
          ( Op stack nu 5 getallen, 5 staat bovenop )
. . . . .
          ( Dit drukt af: 5 4 3 2 1 )
```

Wanneer nog een punt gegeven wordt geeft FORTH een foutmelding omdat er niets meer op de stack staat.

Rekenen in FORTH gaat met definities die als operatoren werken. De definitie + haalt de bovenste 2 getallen van de stack, telt ze op en zet de som weer op de stack.
Zo zijn er o.a. de volgende operatoren: + - * /
Een voorbeeldje van rekenen in FORTH:

2+3 wordt in FORTH:

```
2 3 +
```

(5*3+12)/2 wordt in FORTH:

```
5 3 * 12 + 2 /
```

Zoals je ziet werkt FORTH met het rekenen een beetje achterstevoren, de operatoren (zoals +) komen na de getallen waarop ze werken. Dit heet Postfix-notatie of Reversed-Polish-notation (RPN). Dit lijkt erg moeilijk, maar het werken ermee went snel. Bezitters van HP-rekenmachines zullen dit beamen: hierop word ook RPN gebruikt.

Voor het werken met de stack zijn de volgende woorden handig:

DUP : Dupliceert bovenste waarde op stack.
DROP : Verwijderd bovenste waarde van de stack.
SWAP : Verwisselt de twee bovenste getallen op de stack.
OVER : Copieert het op een na bovenste getal boven op de stack. a b c (c is stacktop) wordt na OVER: a b c b
ROT : Verplaatst het 3e getal naar de top van de stack: a b c wordt: b c a
n PICK: n is z'n inputparameter. Als over voor het nde getal. Na 4 PICK: a b c d e wordt: a b c d e b
n ROLL: Als ROT voor het nde getal.
Na 4 ROT: a b c d e wordt a c d e b

Een voordeel van FORTH, wat bijna geen enkele taal kan, is dat zelf operatoren gedefinieerd en geherdefinieerd kunnen worden.
Kommentaar kan erg verhelderend werken. Het is een goede gewoonte om na het maken van de definitie aan te geven wat de definitie op de stack verwacht en wat hij achterlaat. Dit kan op de volgende manier:

```
( op stack voor de def --- op stack na de def )
```

Ook een goede lay-out is erg belangrijk (zie volgende voorbeeld:

```
( Kwadrateren en optellen )
 : ^2 ( n1 --- n2 \ Kwadrateer )
      DUP *
   ;

: Q+   ( n1 n2 --- n3 )
     ^2          ( n2 gekwadrateerd:     n1 n2*n2 )
     SWAP        ( getallen verwisseld:  n2*n2 n1 )
     ^2          ( n1 gekwadrateerd:     n2*n2 n1*n1 )
     +           ( Tel op,Klaar:         n2*n2+n1*n1 )
 ;
```

Gebruik: 2 3 Q+ .
geeft: 13
(=2*2+3*3=4+9)

Andere belangrijke operatoren en functies zijn:
*/ */MOD MOD /MOD NEGATE ABS MAX MIN OR AND XOR

*** Input/Output ***

De belangrijkste I/O routines zijn:

?TERMINAL: Scan het toetsebord. Wanneer geen toets is ingedrukt geef een 0 op stack, anders de ascii-waarde.

KEY : Haalt een character van het toetsenbord en zet de ascii waarde op stack. Wacht op indrukken van een toets.

EMIT: Stuurt het ascii-character op de stack naar de output.

Deze routines zijn gevectored. Alle I/O-routines maken gebruik van deze basisroutines. Omschakelen naar b.v. een terminal is dus vrij eenvoudig.

*** Structuur in FORTH / beslissingen,loops ***

Zoals alle talen kan FORTH beslissingen nemen die het programmaverloop bepalen. In basic gebeurt dit vnl. door middel van sprongen in het programma met GOTO en IF statements.
Een van de grote voordelen (je leest het goed VOORDELEN) van FORTH is, dat het geen GOTO of iets wat er op lijkt kent. Sprongen in het programma maken een programma meestal volstrekt onleesbaar. Soms zijn er uren puzzelen nodig voordat je erachter komt in welke situaties er waarheen gesprongen wordt, en wat er nou eigenlijk gebeurd. Bij lange programma's raak je dan ook vaak halverwege het programmeren de draad kwijt, tenzij het zeer overzichtelijk geschreven is (veel subroutines, allemaal bij elkaar en 95% kommentaar).
Van basicprogramma's zegt men terecht vaak dat het Spaghettiecode is (allemaal sprongen door elkaar).

In FORTH wordt het programmaverloop GESTRUCTUREERD gerregeld, zoals ook in PASCAL en ALGOL.

Beslissingen:
Een goed of fout conditie wordt aangegeven door een getal op de stack:

```
        0: niet waar
   niet 0: waar
```

Functies die booleans (=foutconities) afgeven zijn o.a.:
> < = NOT 0= 0< 0>

Een beslissing wordt als volgt uitgevoerd:

```
    --conditie_op_stack--
    IF
       -opdrachten-- ( Uitgevoerd als waar       )
    ELSE
       -opdrachten-- ( Uitgevoerd als niet waar )
    THEN             ( Dient als afsluiting     )
```

Het ELSE deel mag ook weggelaten worden.

Om tussen meerdere situaties te kunnen kiezen is een CASE-statement aanwezig. Afhankelijk van de waarde van het getal op de stack wordt een stukje uitgevoerd.

```
    DOCASE          ( Opening van CASE-statement              )
       -opdrach--   ( Het getal moet voor de eerste CASE op )
       -ten--       ( stack staan.                            )
       n1 CASE
          -opdrachten--
                    ( Als het getal gelijk is aan n1 wordt    )
                    ( dit deel uitgevoerd, anders wordt er    )
                    ( naar de statements na ELSE gesprongen.  )
                    ( Hierna wordt naar ENDCASE gesprongen.   )
       ELSE         ( Na ELSE komen statements die voor de    )
                    ( volgende waarde zorgen, hier: n2        )
       n2 CASE
          -opdrachten--
       ELSE
       n3 CASE
          -opdrachten--
       OTHERWISE    ( Geen waarde komt met het getal overeen  )
                    ( Het beslissingsgetal wordt weggegooid.  )
Inplaats van OTHERWISE kan ook ELSE   )
                    ( Wanneer inplaats van OTHERWISE ELSE     )
                    ( gebruikt wordt dan wordt het niet weg-  )
                    ( gegooid. Dit deel is niet verplicht     )
          -opdrachten--
    ENDCASE         ( Sluit statement af                      )
```

Loops (lussen) kunnen op de volgene wijze gemaakt worden:

```
    BEGIN           ( Begin de loop                           )
       -opdrachten--
    REPEAT          ( Spring naar bijbehorend BEGIN           )
                    ( In dit geval is de loop oneindig.       )


    BEGIN
       -opdrachten--
       --conditie--
    UNTIL           ( Spring naar BEGIN als de conditie       )
                    ( niet waar is, ga anders door met het    )
                    ( programma.                              )
```

```
--max+1--
--beginwaare--( Twee parameters op stack                    )
DO             ( Begin de DO-loop                           )
   --conditie--
LOOP           ( LOOP hoogt teller op en stopt wanneer      )
               ( deze gelijk is aan max+1. Anders wordt)
               ( naar het statement na DO gesprongen        )
```

Naast LOOP is er ook een +LOOP, die een getal op de stack nodig heeft om de teller op te hogen (LOOP is equivalent met 1 +LOOP). Hiermee kan ook achteruit geteld worden.

Om een loop te verlaten is er ook het WHILE statement. WHILE heeft een conditie nodig, is deze niet waar dan wordt de loop verlaten, anders gaat de loop door.

De definitie (en dus ook de loop) kan verlaten worden met de definitie EXIT .
EXIT heeft desastreuze gevolgen in een DO..(+)LOOP .

Al deze structuren kunnen vrijwel onbeperkt genest worden (in elkaar gebruikt). Vooral bij nesting is een goede layout erg belangrijk, b.v. bij elk nieuw niveau 3 spaties inspringen.

Wat bij vrijwel geen taal kan: Controlestructuren kunnen in FORTH ook bijgemaakt worden. Dit is echter iets voor de geoefende FORTH-programeur die zijn FORTH goed kent.
De hier getoonde structuren kunnen alleen tijdens compilatie (tussen : en ;) gebruikt worden.

*** Variabelen en constanten ***

FORTH kent variabelen en constanten:

```
4 CONSTANT VIER     ( VIER laat nu het getal 4 achter )
VALUE VAR1          ( Variabele VAR1 gemaakt.         )
VALUE VAR2
VIER TO VAR1        ( VAR1 bevat 4                    )
VAR1 TO VAR2        ( VAR2 bevat nu ook 4             )
VAR1 DUP * TO VAR2  ( VAR2 bevat nu 16                )
VAR1 2 / +TO VAR1   ( VAR1 bevat nu 6                 )
```

CONSTANT en VALUE zijn bouwers zoals : die een woord in de dictionary maken.

Een ander type variabele wordt gemaakt met VARIABLE:

```
VARIABLE PIET
```

Uitvoering van PIET laat het adres van de variable op de stack achter.
De definitie @ haalt de inhoud van een adres op.
De definitie ! stopt een waarde in een adres (n addr ---).
Voorbeeld:

```
5 PIET !            ( PIET bevat 5 )
PIET @              ( 5 op stack   )
3 PIET +!           ( PIET bevat 8 )
```

*** Talstelsels ***

Met FORTH kan in ieder willekeurig talstelsel gewerkt worden. De USER-variable (=systeemvariabele) BASE bevat de basis van het talstelsel. Wanneer bijvoorbeeld in het 2-tallig stelsel gewerkt moet worden, dan is het volgende voldoende (vanuit een hoger talstelsel als 2):

```
2 TO BASE
```

Het systeem kent dan alleen nog maar binaire getallen.
Is dit vaak nodig dan kan er ook een definitie van gemaakt worden:

```
: BINAIR 2 TO BASE ;
```

Let op dat inspecteren van BASE met:

```
BASE .
```

altijd 10 geeft (handiger is BASE 1 - .).
Handig, vooral wanneer je niet meer weet in welke basis je werkt zijn de definities:
DECIMAL HEX OCTAL

*** Diversen FORTH ***

-Zelf gemaakte definities kunnen worden verwijderd met:
FORGET <naam>
Alle definities die na <naam> gemaakt zijn zijn dan ook vergeten.

-De woorden in de dictionary zitten niet in een liniaire lijst maar in een soort boom. Met VOCABULARY <naam> kan een nieuwe tak worden gemaakt. Aanroepen van <naam> zorgt ervoor dat in de nieuwe tak wordt begonnen met zoeken.

-Het commando VLIST drukt alle definities af, beginnend met het laatstgemaakte woord in de huidige zoek-vocabulary. Indrukken van een toets stopt dit listen.

-FORTH programma's worden meestal met een editor in een file gemaakt. Dit wordt dan op FORTH geladen (vertaald).

-Woorden als VARIABLE VALUE : CONSTANT en VOCABULARY , de bouwers, kunnen ook zelf gemaakt (geherdefinieerd) worden.
CREATE <naam> maakt een nieuwe naam-header aan.

Na DOES> wordt aangegeven wat de nieuwe definitie <naam> moet doen. Op de stack staat dan een geheugen adres.
CONSTANT zou als volgt gedefinieerd kunnen zijn:

```
: CONSTANT
      CREATE    ( Maak header bij uitvoering van CONSTANT)
         ,      ( "komma": zet het getal op stack in het )
                ( geheugen (en reserveert dus 2 bytes.   )
      DOES>     ( Het gedeelte hierna wordt uitgevoerd   )
                ( bij het aanroepen van de constante.    )
                ( Het adres van het met "komma" gecompi- )
                ( leerde getal staat dan op stack.       )
         @      ( Haal de inhoud van het adres op.       )
;
```

-Het is zeer gemakkelijk om lowlevelroutines in FORTH te gebruiken, met behulp van een FORTH-assembler. De routines krijgen een naam en zijn niet te onderscheiden van de andere FORTH definities.

-Na goed uitvoeren van een interaktief commando antwoord FORTH met: "ok" en gaat de cursor naar een nieuwe regel.

-FORTH werkt standaard met getallen van 16 bits. Onderandere hierdoor is FORTH zeer snel. Ook zijn er definities voor 32 bits getallen (D+ D- etc.). Wanneer bv. floating point nodig is, kan er gemakkelijk een floatingpoint pakket op FORTH geladen worden voor het programmeren.

-Er zijn veel FORTH dialecten. Door de uitbreidbaarheid van FORTH is het echter zeer gemakkelijk om een conversiepakket voor een ander dialect te schrijven. Jouw FORTH gedraagt zich dan als het andere dialekt, en uitwisseling van programma's kan plaatsvinden.

-Met FORTH kan gemakkelijk recursief gewerkt worden. Omdat i.v.m. de herdefinieermogelijkheid een naam van een definitie pas gevonden kan worden na beeindiging van het maken van de definitie, is het woord: MYSELF ingevoerd.
Het bewaren van waarden tijdens recursie kan mooi op de stack.

*** FORTH op de DAI ***

Voor de DAI zal binnenkort een FORTH-versie beschikbaar zijn. Deze zal voor ongeveer f10- te koop zijn. Hierbij zit dan een FORTH-assembler en Editor.
Op de bijeenkomst zal een voorlopige versie aanwezig zijn, die gecopieerd mag worden. Hier zijn de assembler en editor al op geladen. De variabelen in deze FORTH (ook de Uservariabelen) zijn allemaal nog van het VARIABLE type: VALUE is nog niet geimplementeerd. Ook de grafische routines zijn nog niet geimplementeerd.

Bij deze FORTH kan het gehele ascii-alfabet gebruikt worden. De CTRL-toets wordt een echte Control-toets, waarmee controlcharacters gegenereerd kunnen worden door eerst deze toets in te drukken en dan andere. Het toetsenbord is als volgt ingedeeld:

Control-TAB	(ascii: 0)	NULL
Control-G	(ascii: 7)	BELL
Control-H	(ascii: 8)	BackSpace
	Dit doet ook	de toets pijl links.
Control-J	(ascii: A)	LineFeed
	Dit doet ook	de toets pijl beneden.
Control-L	(ascii: C)	FormFeed (Scherm schoon)
	Dit doen met	shift alle pijltoetsen.
Control-M	(ascii: D)	Cariage Return (Geen LineFeed)
	Dit doet ook	de RETURN-toets.
Control-X	(ascii: 18)	CANcel (De regel waar de cursor staat wordt gedelete)
TAB-toets	(ascii: 40)	@
Shift-0	(ascii: 5C)	\
Shift-space	(ascii: 5F)	_ (Underline)
Shift-TAB	(ascii: 60)	` (Tikkie terug)
CharDel	(ascii: 7B)	{
Shift-Return	(ascii: 7C)	|
Shift-ChDel	(ascii: 7D)	}

De toets: pijltje naar rechts haalt een character uit de inputbuffer op. Hiermee kunnen stukken uit de vorige zin herhaald worden.
Shift-Control is de Key-lock (Grote/Kleine letters).

De pijltjestoetsen werken alleen als boven beschreven bij normaal interactief werk. Met KEY geven ze de waarden die ze normaal bij de DAI horen te geven.

Voorbeeld1: machtsverheffen:
Voor positieve machten geldt:
-Als p=0: n^p=1
-Als p is even: n^p=(n^(p/2))^2
-Als p is oneven: n^p=n*n^(p-1)

Dit kan gemakkelijk in FORTH geprogrameerd worden:

```
: ^2  ( n1 --- n2 ! Kwadrateer )
      DUP *
  ;

: ?ODD  ( n --- flag ! Test of oneven )
      1 AND
  ;

: ^  ( n1 power --- n2 )
     ?DUP          ( Dupliceer bovenste getal als niet 0 )
     IF            ( Niet 0 )
        DUP ?ODD
        IF         ( Oneven )
           OVER SWAP 1-
                   ( Op stack nu: n1 n1 power-1 )
           MYSELF  ( n1 n1^(power-1)
           *       ( n1*n1^(power-1)
        ELSE       ( even )
           2/ MYSELF ^2
                   \ Op stack nu: (n1^(power/2))^2
        THEN
     ELSE          ( power=0 )
        DROP 1
     THEN
  ;
```

Deze routine berekent de machten met een ongelooflijke snelheid.

Voorbeeld2:Grootste gemene deler.
De GGD van 2 getallen, n1 en n2, wordt als volgt gevonden:
-Als n2=0 is, dan is n1 de GGD.
-Anders: n2 wordt de rest van n1/2 (= n1 MOD n2)
n1 wordt de oude n2.
Hiervan nu wordt weer de GGD bepaald.

```
: GGD  ( n1 n2 --- n3 )
     ?DUP
     IF SWAP OVER MOD MYSELF
     THEN
;
```

Voorbeeld:Complexe getallen.

```
( Opbouw: Reel deel, Imaginair deel, 2+2 bytes)

  : IVARIABLE
      CREATE
         4 ALLOT     ( Reserveer 4 bytes )
      DOES>          ( Laat alleen adres achter )
    ;

  : I!  ( complex addr --- )
      ROT OVER ! 2+ ! ( Eerst Re, dan Im )
    ;

  : I+! ( complex addr --- )
      ROT OVER +! 2+ +!
    ;

  : I@  ( addr --- complex )
      DUP @ SWAP 2+ @
    ;

  : I+  ( c1 c2 --- c3 )
      ROT +       ( IMM )
      ROT ROT +   ( RE  )
      SWAP
    ;

  : I-  ( c1 c2 --- c3 )
      ROT SWAP - ( IMM )
      ROT ROT  - ( RE  )
    ;

  : I* ( c1 c2 --- c3 )
      ( Om programma's te vervolgen is het handig om de )
      ( stack op papier bij te houden.                  )

      4 PICK 3 PICK *    ( Re1*Re2 )
      4 PICK 3 PICK *    ( Im1*Im2 )
      -                  ( Reel deel klaar )
      -5 ROLL            ( Inverse van 5 ROLL )
      4 ROLL *           ( Re1*Im2 )
      -3 ROLL *          ( Re2*Im1 )
      +                  ( Imaginair deel klaar )
    ;


  : I. ( c --- ! druk af)
      SWAP .
      ." + "      ( Druk "+" af.)
      .
      ." i "
    ;


  ( Gebruik: )

   3  4      (  3+ 4i )
  -5 22      ( -5+22i )

   2DUP      ( Doet: OVER OVER )
   I.        ( Dit geeft: -5 + 22 i )

   I+ I.     ( Dit geeft: -2 + 26 i )
```

APPENDIX

VLIST van het voorlopige FORTH-systeem.
Het adres is het Codefield-adres.
Een "i" achter dat adres geeft aan dat het woord immediate is.

```
 CFA
3C86  TASK       3C78  FX
3C61  FILE       3C47  EDIT       3BF9  EDINT      3B46  EDKEY
3B19  $FIND      3AE5  GET$       3AA1  BLDEL      3A5C  BLMOV
3A1F  COPY       3A05  SETEND     39DB  SETPOS     39C1  CLEAN
39B4  pc         39AA  PCHAR      3967  REMOVE     394B  >END
3927  M^         38EF  MY         38BB  UPDATE     387F  NEWSCR
3865  LPAGE      383A  LCOUNT     37F0  COOR       37DA  CURPOS
378A  POS        3757  LINEADR    371B  $SEARCH    3708  EDCHR
36F9  EDSCR      36EC  cy         36E2  CX         36D8  wy
36CE  WX         36C4  flim       36B8  fend       36AC  fstart
369E  TASK       363Ci LABEL      3627i CODE       360Fi ;CODE
2D80i ASSEMBLER                   2D6F  TASK       2D5Bi ONERR>
2D4Di . (=CR)    2D40i . (=NULL)  2D13i ENDCASE    2CF0i OTHERWISE
2CBBi CLOSE-IF   2C99i THEN       2C69i ELSE       2C55i +LOOP
2C40i LOOP       2C19i UNTIL      2C02i REPEAT     2BA9i WHILE
2B73i DO         2B61i BEGIN      2B3Ai CASE       2B28i DOCASE
2B0Ai IF         2AF0  ?FAST      2AD5  USER       2ABB  CONSTANT
2AA5  VARIABLE   2A8Bi ;          2A68  :          2A4Fi DOES>
2A3Ai ;CODE      2A21  (;CODE)    2A0A  IMMEDIATE
29EBi '          29D6i MYSELF     2986i [COMPILE]
2972  VOCABULARY                  2948  CREATE     28C6  HEADER
28B0i \          2881i (          2826  FORGET     2777  VLIST
2718  ID.        26F5  DUMPST     2657  DUMP       2643  TAB
2616  .EMIT      25D7  EXTEND     259E  COLD       254F  ABORT
24BD  QUIT       24A9  !TIB       2499  LOAD       246F  -->
2432  LD         23EA  INTERPRET                  23CBi DLITERAL
23ADi LITERAL    226F  ]          2260i [          2239  CHAIN
2224i FORTH      220F  DEFINITIONS                21EC  SMUDGE
21CC  LAST       21BA  !CSP       21A6  PFA        2197  LFA
2186  NFA        2179  CFA        2156  TRAVERSE   212E  FIND
20F8  -FIND      20E5  LATEST     20B1i ."         2089  WORD
2069  QUERY      1F28  EXPECT     1EB4  NUMBER     1E78  CONVERT
1E4B  MESS"      1E30  COMPILE    1E19  C,         1E07  ,
1DE0  ALLOT      1DC9  ?EXEC      1DB2  ?COMP      1D97  ?CSP
1D81  ?PAIRS     1D65  ?ERROR     1D02  MESSAGE    1C86  ERROR
1C6C  S0         1C5E  HERE       1C48  TWRITE     1C2E  TREAD
1C1D  C?         1C0F  ?          1BFE  H.         1BDC  B.
1BCE  U.         1BC0  .          1BB1  D.         1B9F  .R
1B7B  D.R        1B64  #S         1B3A  #          1B23  SIGN
1B0B  #>         1AFB  <#         1AE5  HOLD       1AD5  PAD
1AC2  HEX        1AAF  DECIMAL    1A98  OCTAL      1A5C  TYPE
1A3E  SPACES     1A2C  SPACE      1A11  CR         1A01  PAGE
19EF  LF         19DF  BS         19CF  BELL       19B7  EMIT
19A1  TERMINAL   1957  SEMIT      18D0  KEY        18BB  KEYBOARD
18A5  ?TERMINAL                   188A  !?T        187B  MOVE
186B  BLANKS     1859  ERASE      182A  FILL       1818  */
1806  */MOD      17E1  M*         17C5  M/MOD      178E  M/
177A  DABS       1762  MAX        174B  MIN        173C  D>
1710  D<         16FE  2SWAP      16E5  2OVER      16D8  BL
16CE  TRUE       16C2  FALSE      16B5  4          16AC  3
16A3  2          169A  1          1691  0          1688  -1
167E  -2         1674  -3         166A  -4         164D  FILEIN
1628  FILEOUT    15F8  F$         15EC  PUTC       15CF  GETC
```

15AF	SKEY	1589	CURY	1579	CURX	1568	CURSOR
154E	BYE	1534	SWITCHB	1508	DIGIT	14EB	CHECKSUM
14C2	-TRAILING			14AF	><	1497	FINDBYTE
1484	COUNT	1451	IMOVE	142A	CMOVE	141A	TOGGLE
1405	+!	13FA	C!	13ED	!	13E2	C@
13D6	@	13C7	MOD	13B6	/MOD	13A3	/
133C	(U/M)	12F9	U/MOD	12D2	*	1297	U*
1275	D-	1257	D+	1247	-	123D	+
122E	XOR	121D	OR	120D	AND	11F9	UMIN
11E4	UMAX	11C9	DNEGATE	11B4	NEGATE	119E	S->D
1188	ABS	116B	2/	1161	2*	1154	4-
1147	4+	113B	3-	112F	3+	1124	2-
1119	2+	110F	1-	1105	1+	10F1	U>
10DD	U<	10D3	>	10B8	<	10A2	=
108D	0>	107C	0<	106B	NOT	1059	0=
104E	RP@	1037	RP!	1028	R@	1016	R>
1003	>R	0FF4	?DUP	0FE6	2DUP	0FDA	DUP
0FCF	SWAP	0FC3	2DROP	0FB7	DROP	0F4D	ROLL
0F40	ROT	0F1D	PICK	0EFB	DEPTH	0EED	OVER
0ED7	SP!	0ECA	SP@	0E8C	ENCL	0E46	(FIND)
0E3C	EXECUTE	0E29	LIT	0E16	LEAVE	0E01	J
0DF4	I	0DE9	CALL	0DD1	EXIT	0D0B	0BR
0CFB	BR	0CF0	?STACK	0CA9	?S	0C8E	(ERR)
0C81	STATE	0C74	>IN	0C69	DPL	0C5E	OUT
0C53	HLD	0C48	ERR	0C3D	CSP	0C32	TIB
0C27	CURRENT	0C18	DUPLEX	0C0A	(LAST)	0BFC	NOCH
0BF0	BASE	0BE4	COL	0BD9	C/L	0BCE	WIDTH
0BC1	(NUM)	0BB4	(EXPECT)	0BA4	(?T)	0B98	(KEY)
0B88	(EMIT)	0B7D	CHVECT	0B6F	CONTEXT	0B60	MESS
0B54	VLINK	0B47	DP	0B3D	FENCE	0B30	FIRST
0B23	(TIB)	0B16	(PAD)	0B09	RL	0AFF	R0
0AF5	SL	0AEB	(S0)				

***** FORTH *****

FORTH, een computertaal ontworpen door Charles H. Moore, is nu ook beschikbaar voor de DAI. Een voorlopige versie is door Adri Bos gemaakt en kan op de bijeenkomsten worden gecopieerd. In deze versie zijn de grafische routines nog niet geïmplementeerd. Een assembler en een full-screen-editor met "zoek-string" en copieer- en deletemogelijkheden zijn al wel aanwezig.
Over deze FORTH-versie (die ik een kleine week voor het schrijven van dit stukje ontving) ben ik zeer enthousiast. Weliswaar is mijn DAI al enige keren gecrasht, maar dit was tot nu toe in alle gevallen aan mijn eigen programmeeronkunde te wijten.
Er is een nuttig gebruik gemaakt van de CTRL-toets. Zo bewerkstelligt bijv. ↑H het deleten van een karakter, ↑L is "scherm schoon", ↑X is "cancel de regel waar de cursor staat" etc.. Voor het overgaan van grote naar kleine letters en omgekeerd wordt nu SHIFT CTRL gebruikt.
Uiteraard heb ik in deze korte tijd lang niet alle mogelijkheden kunnen uittesten, maar een eerste indruk is dat alles naar behoren werkt.
Diegenen, die weinig of nog niets van FORTH af weten, kan ik aanraden het augustusnummer (1980) van BYTE te lezen, dat aan deze taal is gewijd. Het aprilnummer van BYTE (1982) bevat een artikel over een DOS voor FORTH, dat met enige voorkennis een goed inzicht in de mogelijkheden kan geven.
FORTH op de DAI-PC, van harte aanbevolen!

***** Wim van Eck *****

***** DE PROGRAMMEERBARE GENERATOR VOOR GRAFISCHE TEKENS *****
(zie DAI-manual blz. 14-16)

De karakter- en grafische tekensgenerator maakt gebruik van een stelsel van variabele lengten om een efficient gebruik van het geheugen te maken.
Voordat een verdere beschrijving volgt, eerst een uitleg van enige uitdrukkingen:

1. Een "SCAN" is: 1(één) horizintaal over het TV-scherm bewegende electronenstraal welke zich met elke herhaling een stukje naar beneden beweegt. Op onze TV beweegt deze SCAN 625 keer heen en weer, maar omdat de afhankelijkheid van de besturing van de TV-scan wordt vermeden, is een display van minimaal 2 en maximaal 32 scan's per line mogelijk.
2. Een "LINE" is: Een aantal scan's (zie 1), welke alle worden gestuurd door de zelfde informatie in de RAM.
3. Een "BLOB" is: Het kleinste op het scherm te maken oppervlak. Deze oppervlakte is wat afmeting betreft afhankelijk van de SCREEN-modes.
4. Een "FIELD" is: Een set van 8 blobs waarvan de kleur wordt bepaald door een paar (2) byte's in het geheugen.
5. Een "MODE" is: Een van de verschillende manieren waarop informatie op het scherm zichtbaar wordt gemaakt.
6. Een "PICTURE" is: Het beeld of informatie dat door een aantal van boven naar beneden bewegende line's op het scherm zichtbaar wordt gemaakt. Elke line kan onafhankelijk van de vorige, in één der andere, modes geprogrammeerd zijn.

Bij het begin van de display van elke line bepalen twee byte's, ofwel het "CONTROL WORD", uit het geheugen de mode van die line en geven aan of al of niet een kleur geprogrammeerd is en herhaald moet worden.
De eerste byte van dit paar is het "HIGH ADRESS"-byte (deze is na het starten van de PC altijd ≠# 7A en dat zijn dus 8 bits van links naar rechts genummerd 7 t/m Ø, waarvan de waarde binair geprogrammeerd is als Ø111 1Ø1Ø.

De programmering van de display per regel is als volgt:

MODE CONTROL

bits 7 en 6 bepalen als:

- ØØ grafisch 4-kleuren display;
- Ø1 4-kleuren karakter display;
- 1Ø grafisch 16-kleuren display;
- 11 16-kleuren karakter dispay.

RESOLUTION CONTROL

bits 5 en 4 bepalen als:

- ØØ lage grafische resolutie (grote display)
- Ø1 medium grafische resolutie (middengr. display)
- 1Ø hoge grafische resolutie (kleine display)
- 11 extra lage grafische resolutie (tekst display)

LINE REPEAT COUNTING

De bits 3 t/m Ø bepalen het aantal keren waarop de informatie (DATA), bestemd voor de regel, herhaald moet worden.
Met het programmeren van bits 5 en 4 zijn er dus 4 mogelijkheden voor karakter display.

In het nu volgende programma wordt, ter verduidelijking, het high adress byte achtereenvolgens ≠#4A = (Ø1ØØ 1Ø1Ø); 5A = (Ø1Ø1 1Ø1Ø); 6A = (Ø11Ø 1Ø1Ø) en weer 7A = (ØØ11 1Ø1Ø). Dus de enige veranderende data zijn in de bits 5 en 6. Het aantal mogelijke karakters is uiteraard ook afhankelijk van de grootte.

```
LIST
15Ø     PRINT CHR$(12)
17Ø     PRINT
2ØØ     COLORT 8 Ø 8 8
25Ø     PRINT
3ØØ     POKE #BE5D,#4A:PRINT "HET BE-"
3Ø5     PRINT
31Ø     POKE #BD51,#5A:PRINT "INVLOEDEN van het"
32Ø     PRINT
35Ø     POKE #BC45,#6A:PRINT "SCREEN DATA FORMAT door BITS 5 en 4 in"
36Ø     PRINT
37Ø     PRINT "HIGH ADDRESS BYTE van START ADDRES LINE Nr. te veranderen"
38Ø     PRINT "in 4 (=Ø1ØØ); 5 (=Ø1Ø1); 6(=Ø11Ø) of 7
(=Ø111 of 6Ø karak-"
39Ø     PRINT "ters. Het veranderen van de BITS 7 en 6 heeft 'n andere
functie,"
4ØØ     PRINT "o.a. de display in kleuren, samen met LOW ADDRESS BYTE, te"
41Ø     PRINT "realiseren indien de DAI-PC is gemodificeerd."
42Ø     PRINT"Zie blz. 13 DAInamic jan/febr. '82"
```

***** door H. Wanders *****

***** INPUT$(N) *****

In microsoft basic komt de functie INPUT$(N) voor. Deze functie werkt als volgt: Er wordt gewacht tot er N keer op een willekeurige toets gedrukt is.
Een microsoft voorbeeld:

```
1ØØ     A$ = INPUT$(4) : PRINT A$
```

Dit voorbeeld wacht in regel 1ØØ totdat er vier willekeurige toetsenzijn ingedrukt en geeft de waarde van de variabele A$ in ASCII.
Als we achtereenvolgens R, E, P en T intoetsen wordt A$ dus "REPT".
Het voordeel hiervan is dat zo input gepleegd kan worden zonder dat het op het scherm verschijnt en men het zonder de RETURN-toets kan afsluiten.
Hierna een voorbeeld om het op de DAI te simuleren.
Aanroep: N=4 : GOSUB 1Ø

```
1Ø      TELLER=0
2Ø      G=GETC : G=GETC : G=GETC
3Ø      G=GETC : IF G=Ø OR (G=8 AND TELLER=Ø) or G=13 GOTO 3Ø
4Ø      IF G=8 THEN KAR$=LEFT$(KAR$,LEN(KAR$)-1) : TELLER=TELLER-1
5Ø      IF G<>8 THEN KAR$=KAR$+CHR$(G) : TELLER=TELLER+1
6Ø      IF TELLER<>N GOTO 3Ø
7Ø      RETURN
```

********** door W.Nijland **********

D42E ON
D44S OFF CAS MOTOR REMOTE.