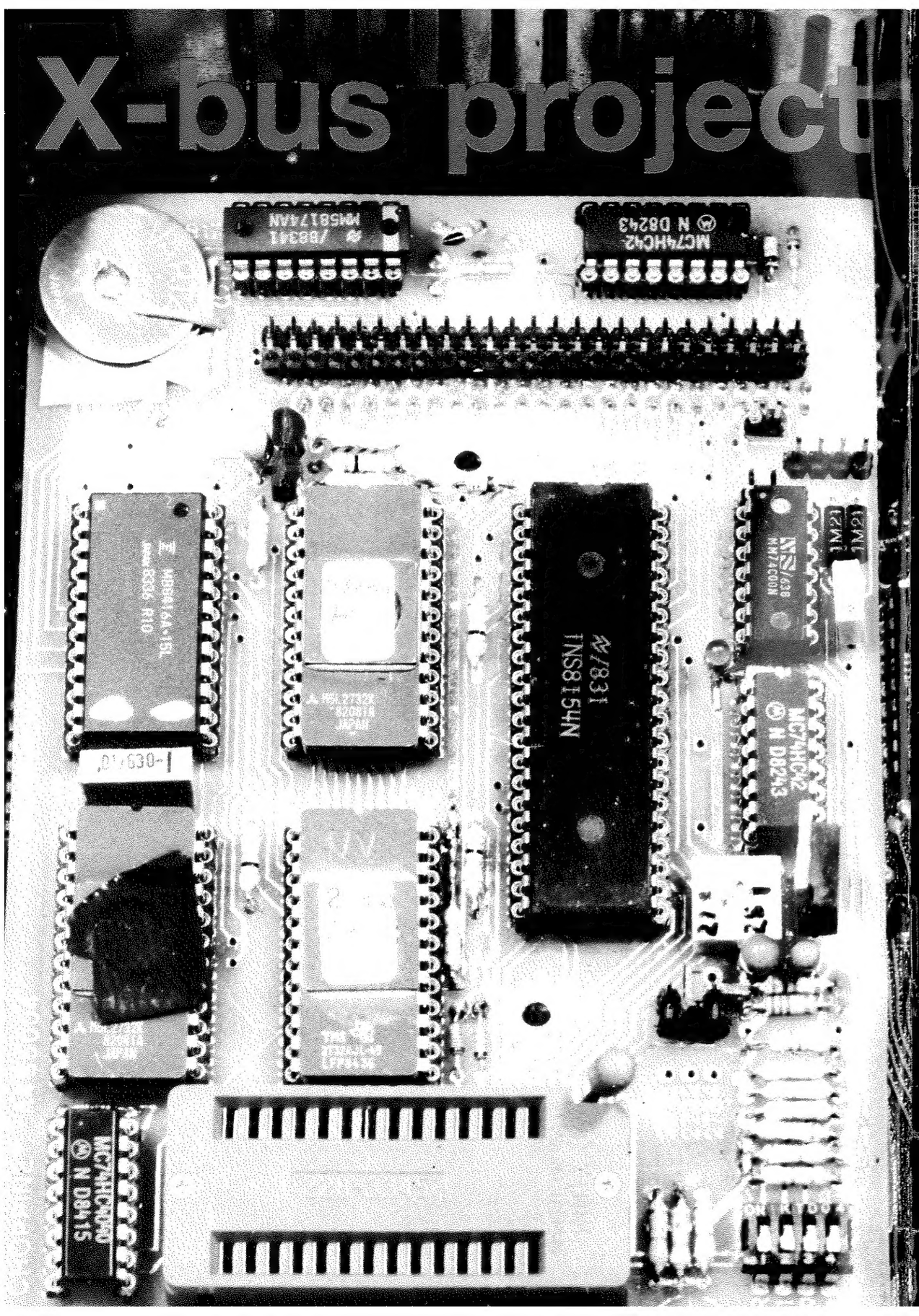
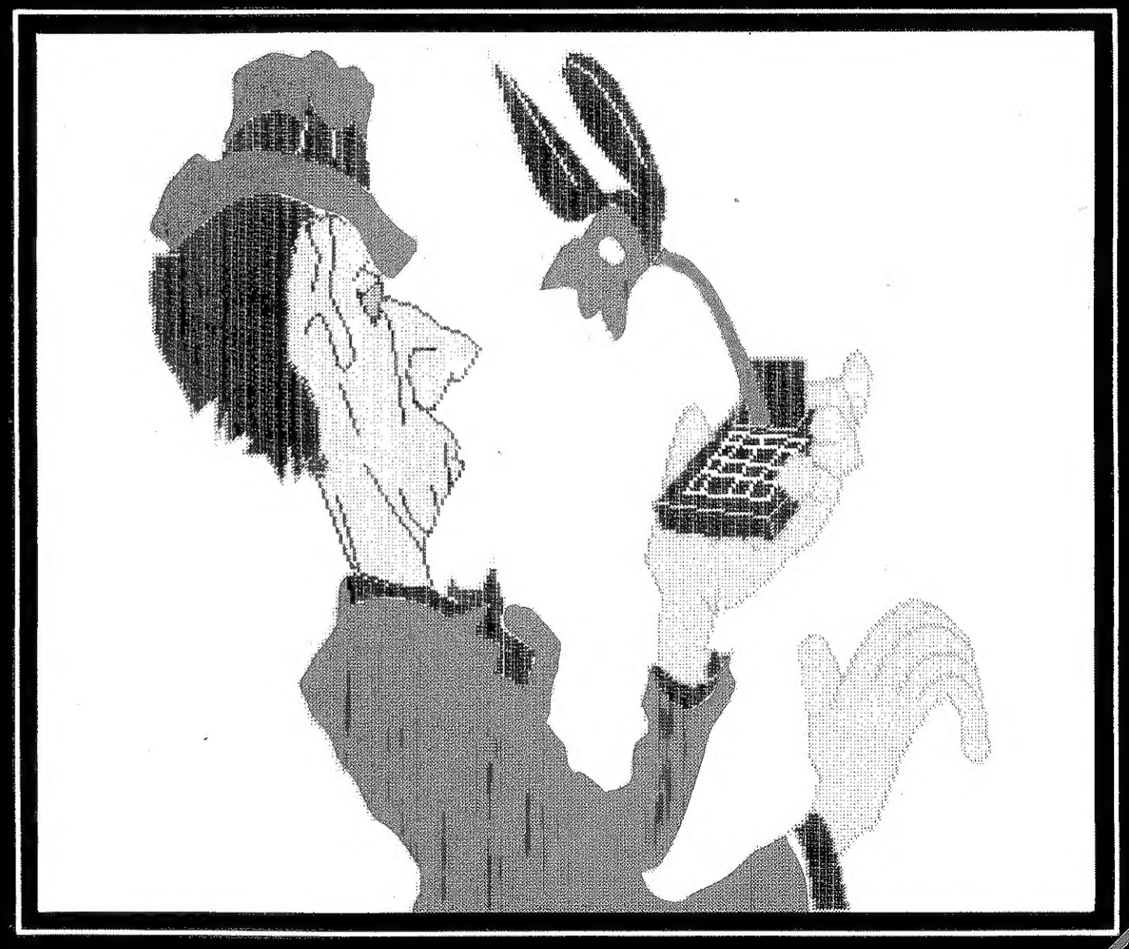
X-bus project



DAI NAMIC

28

tweemaandelijks tijdschrift mei - juni 1985



personal computer users club

een uitgave van dainamic v.z.w.
verantw. uitgever w. hermans, mottaart 20 - 3170 herselt

International

COLOFON

DAInamic verschijnt tweemaandelijks.
Abonnementsprijs is inbegrepen in de jaarlijkse contributie .
Bij toetreding worden de verschenen nummers van de jaargang toegezonden.

DAInamic redactie :

Dirk Bonné	wdw
Freddy De Raedt	Herman Bellekens
Wilfried Hermans	Frans Couwberghs
René Rens	Guido Govaerts
Bruno Van Rompaey	Daniël Govaerts
Jef Verwimp	Frank Druijff
Cedric Dufour	Willy Coremans

Vormgeving : Ludo Van Mechelen.

U wordt lid door storting van de contributie op het rekeningnr. **230-0045353-74** van de **Generale Bankmaatschappij, Leuven,** via bankinstelling of postgiro
Het abonnement loopt van januari tot december.

DAInamic verschijnt de pare maanden.
Bijdragen zijn steeds welkom.

CORRESPONDENTIE ADRESSEN.

Redactie en software bibliotheek

Wilfried Hermans
Mottaart 20
3170 Herselt
Tel. 014/54 59 74

Kredietbank Herselt
nr. 401-1009701-46
BTW : 420.840.834

Lidgelden / Subscriptions

Bruno Van Rompaey
Bovenbosstraat 4
B 3044 Haasrode
België
tel. : 016/46.10.85

Generale
Bankmaatschappij
Leuven
nr. 230-0045353-74

Voor Nederland :

GIRO : 4083817
t.n.v. J.F. van Dunne'
Hoflaan 70
3062 JJ ROTTERDAM
Tel. : (010) 144802

Pour la France :

DAInamic FRANCE
C. Dufour
Rue Lavoisier 9
59149 DUNKERQUE
Tel. 02866 3339

Inzendingen : Games & Strategy

Frank Druijff
's Gravendijkwal 5A
NL 3021 EA Rotterdam
Nederland
tel. : 010/25.42.75

DAInamic

PERSONAL COMPUTER USERS CLUB

4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	54344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

belangrijke ASCII-waarden in DAIpc

functie/symbool	HEX	DEC
back-space	8	8
TAB	9	9
linefeed	A	10
clear screen	C	12
CURSOR UP	10	16
CURSOR DOWN	11	17
CURSOR LEFT	12	18
CURSOR RIGHT	13	19
space-bar	20	32
Ø	30	48
A	41	65
a	61	97
pijltje rechts	89	137
pijltje links	88	136
pijltje boven	5E	94
pijltje onder	8C	140
volle blok	FF	255
verticale lijn	A	10
horizontale lijn	B	11
6 hor. lijnen	1D	29

ASCII - HEX - ASCII CONVERSION TABLE

LSD	MSD	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

VOORWOORD

Herselt, juni '85

Beste Leden,

We krijgen hier op de redactie nog dikwijls de vraag te horen :"Hoe is het met onze DAI ?"
Wat betreft verkoop van nieuwe toestellen is het hier in onze lage landen zeer rustig. Het aantal winkels in Belgie, Nederland en Duitsland waar de DAI te koop is, doet vermoeden dat er met de marketing toch wat mis gaat.
Wat betreft technische specificaties kan onze DAI nog steeds met de beste onder de micro's wedijveren, zelfs met zijn bejaarde 8080-brein. Het software-probleem, dat in de beginjaren zo nijpend was, hebben we met zijn allen opgelost. Zelfs in die mate, dat voor bepaalde markten, oa onderwijs, er een uitstekend aanbod is van goede programma's.
De negatieve factoren hebben we zelf niet in de hand : een ongunstig prijsetiket, een totaal gebrek aan publiciteit en een zeer onduidelijke verkoopspolitiek.
We hopen dat de recente bestellingen voor het onderwijs en de beschikbaaheid van ELAN een stootje in de goede richting geven.

Na een paar maanden van verbouwings werken is ons nieuw lokaal klaar voor gebruik : de verhuis staat voor de deur. Nu we heel wat meer ruimte hebben , kunnen we ons lokaal op bepaalde tijdstippen open stellen voor onze leden, meer nieuws hierover in een volgende uitgave.
In dit nummer geen nieuwe software-pakketten, na de vakantie zullen er zeker een aantal titels aan de bibliotheek toegevoegd worden.

Diegenen die de video-camera-interface wel interessant, maar erg duur vonden, kunnen zich alvast verheugen in het artikel van P.De Laet in de volgende editie. Een echte LOW-COST interface. We proberen hier een degelijk print-ontwerp klaar te stomen , zodat iedereen dit project kan nabouwen.

een prettige vakantie,
tot de volgende keer,

W.Hermans

Evenals de Thais prefereren de Dai-vrouwen kleurige gewaden in zacht rood, violet, azuurblauw en gouden tinten. Daarbij worden camelia's en kunstig bewerkte kammen in het zwarte haar gedragen. De vrouwen kwamen elegant gekleed te voorschijn uit de afgeschermde binnenste vertrekken van hun ruime, op palen gebouwde woningen, om zich te voet, per fiets en in vrachtwagens naar de feestelijkheden te laten vervoeren. Het festival viel dit jaar samen met het begin van het jaar 1347 op de boeddhistische kalender van de Dais.

GvA 1.5.85

INHOUDSTAFEL

28

RESET - RESTART

RESET - RESTART :

Dies ist ein verbessertes Programm, welches in DAInamic 1981 erschien (von Assink).
Bei einem versehentlichen RESET kann man mit diesem Programm das alte Basic-Programm wieder restaurieren.
Dazu verlegt man den HEAP in einen ungenutzten RAM-Bereich.
Danach läd man das Programm "Reset-Restart" und "RUN".
Jetzt gibt man die Position des HEAPs beim alten Programm ein.
Nach einiger Zeit meldet sich der DAI mit den BASIC-Pointern zurück, die jetzt automatisch beim Drücken von <SPACE> restauriert werden.

```
1     REM
2     REM RESET - RESTART
3     REM
4     REM Idee von Assink 1981
5     REM
6     REM Verbessert von Willi Herrmann / D-4320 Hattingen
7     REM
8     REM **** IMPINT ****
9     REM
10    INPUT "Start des HEAP's ";START:PRINT CHR$(12)
20    IF START=0 THEN START=#2EC
30    DEP=START
40    KOM=DEP:DIZ=0
50    OF=PEEK(KOM):NUM=(PEEK(KOM+1) SHL 8) IOR PEEK(KOM+2)
60    IF NUM>2000 OR NUM<1 THEN DEP=DEP+1:IF DEP<#A000 GOTO 40
70    IF FRAC(NUM/10)=0 THEN DIZ=DIZ+1
80    IF DIZ<11 THEN KOM=KOM+1+OF:GOTO 50
90    STEXT=DEP:LENH=DEP-START:KOM=DEP
100   OF=PEEK(KOM):NUM=(PEEK(KOM+1) SHL 8) IOR PEEK(KOM+2)
110   IF NUM<=#FFFF AND NUM<>0 AND OF<>0 THEN KOM=KOM+1+OF:GOTO 100
120   KOM=KOM+1:FINP=KOM
130   A=PEEK(KOM) IAND #F
140   IF A<>0 THEN KOM=KOM+A+1:B=PEEK(KOM) IAND #F:KOM=KOM+B+1:GOTO 130
150   ESY=KOM+1
160   PRINT "   Start HEAP  :  #";HEX$(START)
170   PRINT "  Laenge HEAP  :  #";HEX$(LENH)
180   PRINT "   Start TEXT  :  #";HEX$(STEXT)
190   PRINT "    Ende TEXT  :  #";HEX$(FINP)
200   PRINT "  Ende SYMBOL  :  #";HEX$(ESY)
210   CALLM #D6DA
220   POKE #F800,START IAND #FF:POKE #F801,START SHR 8
230   POKE #F802,LENH IAND #FF:POKE #F803,LENH SHR 8
240   POKE #F804,STEXT IAND #FF:POKE #F805,STEXT SHR 8
250   POKE #F806,FINP IAND #FF:POKE #F807,FINP SHR 8
260   POKE #F808,ESY IAND #FF:POKE #F809,ESY SHR 8
270   FOR N=#F80A TO #F818:READ A:POKE N,A:NEXT
280   CALLM #F80A
290   DATA #C5,#01,#9B,#02,#11,#00,#F8,#21,#0A,#F8
300   DATA #CD,#4F,#DE,#C1,#C9
```

by Raymond VANLATHEM
(club CAROLODAI)

RANDOM NUMBERS

Introduction

We all have already used a generator of random numbers by use of the instruction RND. These numbers are regular used for the resolution of certain problems, in order to make simulations (e.g. in an enterprise), during games, for didactic purposes (choosing exercises) and other ones. In any case, we want the events occurring in an unpredictable sequence, like if they were only depending on the laws of hasard. Well then by definition, random numbers are numbers 'taken by chance' and therefore not predictable for the user.

And what does the DAI do ?

As most computers, the DAI generates numbers called random, but in fact generates pseudo-randomnumbers (even if they might look as real). In reality, the numbers are obtained from a mathematical formula (and therefore determined) and are thus far from random.

To make such a series, one needs starting from a source-number (SEED) given by either the user or the computer itself, to give the series the desired random appereance.

Normally the DAI gives automatically the beginning of the series at the moment you turn your computer on, and generates afterwards the source-random-number. At each RND-instruction, the DAI will generate the following number of the random series.

Let's try the following routine, after power-supply of the DAI :

```
100  FOR I%=1 TO 5:PRINT RND(1),:
     NEXT
```

and execute it twice. We can see on the screen :

```
RUN
0.345451     0.917845     0.703101
 0.633126     0.13235
RUN
0.183837     0.527697     0.651674
 0.451856     0.531174
```

or two different rows of ramdom numbers, containing the first 10 numbers of the series.
A RESET doesn't allow the generation of the same two rows, and the routine above will give the following 5 numbers of the series. If we turn off the computer and wait at least 20 seconds, the DAI will generate again the random-sourcenumber 0.345451. If your DAI doesn't give the same number, this means it hasn't given the expected sourcenumber because of a 'cold' power-supply.

If we execute the routine with X as the argument of RND, the computer generates the same numbers, but now multiplied by X.
E.g. for RND(10), the first numbers displayed will be :

```
3.45451     9.17845     7.03101...
```

Writing A=RND(10) or A=10*RND(1) gives the same result.

And if X gets a negative value ?

What happens then ? The routine will always generate the beginning random number. Thus with X=-1, we'll obtain at each RUN five times the same number -0.571069, even without cutting the DAI's power-supply. This row is called 'repetitive series' in opposition with 'succeeding series', that is generated with a positive value for X.

Every negative value for X will result in a repetitive series : the generated number will be negative, different (according to the value of X), but will always be repeated.

If X=-10, we'll obtain always -3.21069. Notice that A=RND(-10) isn't equivalent to A=10*RND(-1). Indeed, 10*(-0.571069)=-5.71069. On the contrary, RND(-5) gives as generated number -1.60534 (multiplied by 2=-3.21068) : we can see that in this case A=RND(-10) is equivalent to A=2*RND(-5), if we neglect the rounding of the sixth digit. We'll understand later on why.

At first sight repetitive series don't seem very useful in programs. Only the initial number might help the progam-execution since the following numbers would always result in the same data in the same order.

Let's complete the initial routine with the following line :

```
10 Y=RND(-1)
```

and execute the routine twice. The DAI will display :

```
RUN
0.389234     0.359846     0.488753
0.388517     0.171452
RUN
0.389234     0.359846     0.488753
0.388517     0.171452
```

Now it isn't necessairy to cut off the power-supply and to wait 20 seconds. Try the following procedure : after a first RUN, execute RUN 100. The result is :

```
RUN
0.389234     0.359846     0.488753
0.388517     0.171452
RUN 100
0.969859     0.698164     0.996478
0.145397     0.145897
```

The second RUN generates 5 random numbers, coming from the succeeding series. Notice that the series generated after execution of RND(X), with negative value (in the example above with X=-1), is different from the normal series (generated with a positive value for X).

What happens if we replace (at line 10) -1 by an other negative value ? The obtained series are different, because X is uneven.
For example for X=-2, -4, -8, -16,... we'll obtain the same series as with X=-1. But with X=-3, -6, -12, -24,... we'll obtain other series. X=-5, -10, -20, -40,... also will offer other series. But there still exists a relation between the correspondent random numbers of two series. E.g. : all the numbers from the series generated with X=-3 differ 0.5 from the numbers of the series from X=-1, but differ 0.25 or 0.75 from the numbers of the series from X=-5.

```
If X=-1, the DAI gives :
0.389234     0.359846     0.488753
0.388517     0.171452
If X=-3 :
0.889234     0.859846     0.988753
0.888517     0.671452
If X=-5 :
0.639234     0.109846     0.738753
0.138517     0.421452
```

This could be interesting for certain applications. Indeed, for example in the game HEAD [=1] or TAILS [=0] : in the beginning of the game with X=-1, we obtain 5 times HEAD, with X=-3, 5 times TAILS, and with X=-5, we obtain consecutive TAILS - HEAD - TAILS - HEAD - HEAD.

Notice also that a negative series of random numbers can be generated by multiplying the instruction RND (with a positif argument) with a negative number. E.g. : A=-1*RND(1) what isn't the same as A=RND(-1) at all !

Hints and tricks...

To avoid the automatic return to the random source-number when you turn your DAI on, you can use a routine that allows starting the series with the 'N'th number. Of course, the same routine allows skipping N numbers of the series between two program-executions.

```
20 INPUT "Give a number between 1
    and 100 ";A%
30 FOR J%=1 TO A%:Z=RND(1):NEXT
```

This routine, used by line 10, will give for A%=2 the number 0.488753 as random source-number; without line 10, the number 0.703101 will be used. We can choose the starting-number A% arbitrary. E.g. pointing blindly at a number of the telephone-directory and take the last two digits as value for A%, or giving the seconds your watch displayes,...

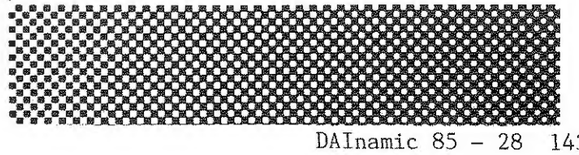
But this routine is rather slow, especially if we use larger INPUT-numbers (A%).

A faster and more varied routine is also possible : a number of 6 digits can be used (e.g. 6 digits of a telephone-number or the complete time (hour, minutes and seconds)).
Here printed the alternative routine :

```
20 INPUT "Give 2 numbers (between 0
    and 999) ";L1%,L2%
30 L%=L1%*1000+L2%:A%=L1%*0.127+128
40 B%=L2%*0.251:C%=L% MOD 997/4
50 POKE #12E,A%:POKE #12F,B%
    POKE #130,C%
```

At the end of this article, a hint that might help you make your programs : you can restart the series of random numbers (like a 'cold' power-supply) if A%=B%=C%=255 and you execute (only) line 50. This way, the resulting random-series will always be the same, starting from the initial random number (SEED).

We'll see later on how this routine works. But notice already that A% must be between 128 and 255, while B% and C% must be smaller than 256.

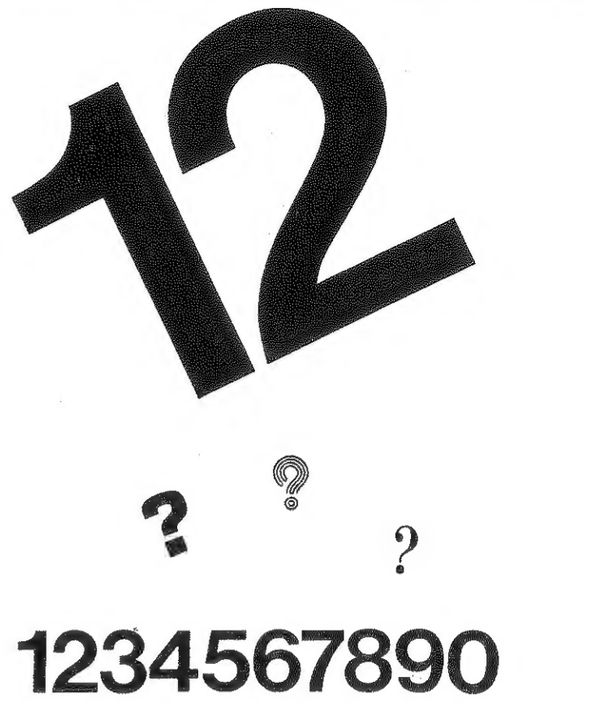


MASTERMIND IN D-BASIC

```
10   TITLE "MASTERMIND"
20   REM ****      Fet per R.BADIAS GIBERT
25   REM ****      Barcelona , Febrer de 1985
30   REM ** This program is a numerical ver-
          sion of the mastermind play
33   REM ** where DAI-pc will try to hit on
          your number
35   REM ** It allows to observe in grafic
          screen the process of number se-
40        lection .
45   REM ** The only information you can sup-
47   REM ** ply,when it shows you a number,is
          the total of common figures in
48        both numbers,specifying if they
          are occuping the same respective
          position (DEAD) or different
          (WOUNDED).
60   REM
105  DEF PROC ESCAPE
107  LOCAL I,J
110  MODE 0:PRINT CHR$(12):CURSOR 12,18
112  PRINT "DATA ENTRY ERROR !!!";PRINT :PRINT
115  FOR I=1 TO TOP:FOR J=1 TO B:PRINT TAB(7);
     T2(J,I);:NEXT
118  PRINT TAB(16+B);T2(B+1,I);"  D";TAB(26+B)
     ;T2(B+2,I);"  W"
120  NEXT:CURSOR 0,1
125  END PROC
300  DEF PROC RECOVER K
310  LOCAL I
320  IF K<=B THEN FOR I=0 TO 9
325  IF T1(K,I)=1 THEN T1(K,I)=0:END IF :NEXT
330  FOR I=1 TO K-1
333  IF T1(K,N(I))<>2 THEN T1(K,N(I))=1:END IF
335  NEXT
340  MAP K
345  END IF
350  END PROC
400  DEF FN GET(K)
401  LOCAL I,S,TEST
403  TEST=0:I=RND(10):S=I
404  REPEAT IF FIRST=1 AND T1(0,S)=0 THEN
     T1(0,S)=1:N(K)=S:TEST=1
405  ELSIF FIRST=0 AND T1(K,S)=0 THEN TEST=1:
     N(K)=S
407  ELSE S=(S+1) MOD 10:END IF
410  UNTIL S=I OR TEST=1
414  FN = TEST
420  END FN
430  DEF PROC COMPARE1 A,B VAR TEST
435  IF A>B THEN TEST=0:END IF
436  END PROC
442  DEF PROC COMPARE2 A,B VAR TEST
445  IF A<B THEN TEST=0:END IF
446  END PROC
450  DEF FN TESTER(K)
460  LOCAL I,J,T,M,F,F1,M1,TEST
500  TEST=1:T=0
508  IF FIRST<>1 THEN REPEAT T=T+1:M=0:F=0
509  FOR I=1 TO K:IF N(I)=T2(I,T) THEN M=M+1:
     END IF :NEXT
511  FOR J=1 TO B:FOR I=1 TO K
512  IF (J<>I) AND N(I)=T2(J,T) THEN F=F+1:
     END IF
513  NEXT:NEXT
514  M1=T2(B+1,T):F1=T2(B+2,T)
515  COMPARE1 M,M1,TEST
517  COMPARE1 F,F1,TEST
523  COMPARE2 M+B-K,M1,TEST
525  COMPARE2 F+B-K,F1,TEST
527  COMPARE2 M+F+B-K,M1+F1,TEST
535  UNTIL TEST=0 OR T=TOP:END IF
559  FN = TEST
560  END FN
705  DEF PROC CANCEL G
710  T1(G,N(G))=1
715  PLOT G,N(G),0
720  END PROC
2000 DEF PROC WRITE
2001 LOCAL I
2002 TOP=TOP+1
2004 ON ERROR GOTO "ER
2005 FOR I=1 TO B:T2(I,TOP)=N(I):NEXT
2011 "WR PRINT CHR$(12);:PRINT "
     PLAY  N.";TOP;"    ..... <<";
2013 FOR I=1 TO B:PRINT N(I);:NEXT:PRINT
     " >> ?"
2014 CURSOR 23,1:INPUT "DEADS    ";
     T2(B+1,TOP)
2015 CURSOR 23,0:INPUT "WOUNDEDS ";
     T2(B+2,TOP)
2018 GOTO "FI
2020 "ER RESUME "WR
2023 "FI END PROC
2100 DEF PROC MARKM
2105 LOCAL I
2110 FOR I=1 TO B:T1(I,N(I))=2:NEXT
2115 END PROC
2130 DEF PROC MARKF
2135 LOCAL I,J
2140 FOR I=1 TO B:FOR J=1 TO B
2145 IF I<>J THEN T1(J,N(I))=2:END IF
2150 NEXT:NEXT
2155 END PROC
2180 DEF PROC MARKMF
2185 LOCAL I,J
2190 FOR I=1 TO B:FOR J=1 TO B
2195 T1(J,N(I))=2:NEXT:NEXT
2200 END PROC
2230 DEF PROC MARKREST Q,P
2240 LOCAL I,J,K,TEST
2265 FOR I=0 TO 9:TEST=0
2268 FOR J=Q TO P:FOR K=1 TO B
2270 IF I=T2(K,J) THEN TEST=1:END IF
2280 NEXT:NEXT
2290 IF TEST=0 THEN FOR K=1 TO B:T1(K,I)=2:
     NEXT:END IF
2300 NEXT
2305 END PROC
```



```
2330 DEF PROC MARKREM
2340 LOCAL I,MF
2345 MF=0:FOR I=1 TO TOP:MF=MF+T2(B+1,I)+
     T2(B+2,I):NEXT
2350 IF MF=B THEN MARKREST 1,TOP:FIRST=0:
     END IF
2400 END PROC
3000 DEF PROC PROCESSINF
3270 LOCAL I,J,K,TEST,M,F,MF
3280 M=T2(B+1,TOP):F=T2(B+2,TOP)
3290 IF M=B THEN DONE=1
3300 ELSE IF M=0 AND F<>0 THEN MARKM
3320 ELSIF M<>0 AND F=0 THEN MARKF
3360 ELSIF M=0 AND F=0 THEN MARKMF
3390 END IF
3395 IF M+F=B THEN MARKREST TOP,TOP
3400 ELSIF M+F<>0 AND FIRST=1 THEN MARKREM
     :END IF
3450 IF TOP>10/B-1 OR M+F>=B THEN FIRST=0
     :END IF
3460 MAP 1
3470 END IF
3480 END PROC
3500 DEF PROC PROVE K
3530 IF K<=B THEN IF GET(K)=0 THEN IF K=1
     THEN ESCAPE
3535 ELSE CANCEL K-1:PROVE K-1:END IF
3540 ELSE IF TESTER(K)=0 THEN CANCEL K:
     PROVE K
3545 ELSE PLOT K,N(K),1:RECOVER K+1:
     PROVE K+1:END IF :END IF
3550 ELSE WRITE :PROCESSINF
3555 IF DONE<>1 THEN PROVE 1:END IF
3557 END IF
3560 END PROC
3602 DEF PROC PLOT I,J,FLAG
3605 LOCAL KOLOR
3608 IF FLAG<>1 THEN IF T1(I,J)=0 THEN
     KOLOR=10
3610 ELSIF T1(I,J)=1 THEN KOLOR=7
3612 ELSE KOLOR=8:END IF
3613 ELSE KOLOR=13:END IF
3615 FILL 3+J*6,6+I*5 5+6*J,7+5*I KOLOR
3630 END PROC
3650 DEF PROC MAP K
3655 LOCAL R
3660 FOR R=0 TO 9:PLOT K,R,0:NEXT
3685 END PROC
3700 DEF PROC NET
3710 LOCAL I
3720 FOR I=0 TO 10
3725 DRAW 1+6*I,9 1+6*I,5*B+9 7:NEXT
3730 FOR I=1 TO B+1
3740 DRAW 2,5*I+4 61,5*I+4 7:NEXT
3745 MAP 1
3750 END PROC
```




```
4000 REM
4001 REM ***   MAIN PROGRAM    ********
4002 ON BREAK GOTO 4500
4005 CLEAR 1000:COLORG 8 10 7 13:COLORT 8 0 0 0
4008 REPEAT MODE 0:POKE #75,32
4010 TOP=0:DONE=0:FIRST=1
4015 B=4:REM NUMBER SIZE
4020 DIM T1(B,9.0),T2(B+2.0,15.0),N(B)
4030 PRINT CHR$(12):CURSOR 0,17:PRINT SPC(22);
     "MASTERMIND"
4035 PRINT SPC(22);"----------"
4040 CURSOR 8,10:PRINT "THINK A NUMBER WITH
     ";B;"NOT REPEATED FIGURES ..... "
4045 REPEAT CURSOR 17,5:PRINT "<< PRESS ANY KEY >>"
4046 WAIT TIME 10:CURSOR 17,5:PRINT SPC(19):
     WAIT TIME 10:UNTIL GETC<>0
4050 MODE 2A:PRINT CHR$(12);:NET
4055 PROVE 1
4270 IF DONE=1 THEN PRINT CHR$(12);" TOTAL OF
     THROWS :";TOP:END IF
4350 PRINT " ANOTHER GAME  Y/N?";
4375 REPEAT H=GETC:UNTIL H<>0
4380 UNTIL H<>ASC("Y")
4500 POKE #75,95
4600 END
```

DCE RWC CONCEPT

Het DCE RWC concept

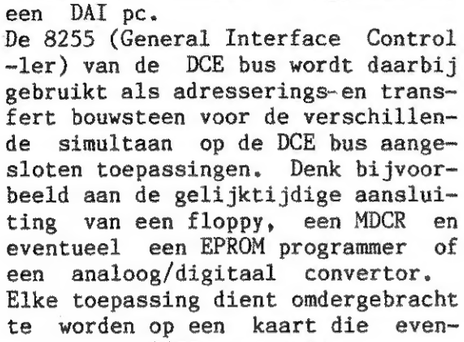
Het Data Communications Equipment-Real World Card concept is een systeem dat toelaat verschillende toepassingen zowel op ' hardware ' als ' software ' manier te koppelen met een DAI pc.
De 8255 (General Interface Control -ler) van de DCE bus wordt daarbij gebruikt als adresserings-en transfert bouwsteen voor de verschillende simultaan op de DCE bus aangesloten toepassingen. Denk bijvoorbeeld aan de gelijktijdige aansluiting van een floppy, een MDCR en eventueel een EPROM programmer of een analoog/digitaal convertor.
Elke toepassing dient omdergebracht te worden op een kaart die eveneens een 8255, of minstens een equivalente logika bevat. Deze bijkomende 8255 bouwstenen krijgen de naam van Real World Interface Controller.
Het is de koppeling tussen GIC en RIC, die het DCE-RWC concept uitmaakt. Dit concept wordt zowel op hardware als op software gebied ondersteund door de DAI firmware. Laat ons het probleem trapsgewijze ontleden.

Verbinding GIC-RIC

De kanalen A,B en C van de GIC worden buitengebracht op een busverbinding waarop alle RIC's eveneens aangesloten worden.
-Het kanaal A van de GIC wordt gebruikt als datatransfertkanaal en is verbonden met de databusbuffer van de RIC's.
-Kanaal B van de GIC wordt gebruikt als adresserings kanaal voor de Real World kaarten. De bits PB4 tot PB7 zijn op elke kaart verbonden via een decodeerschakeling met de $\overline{CS}$ ingang van de betreffende RIC. Deze bits bepalen het kaartadres. De bits PB0 en PB1 zijn op hun beurt verbonden met de registerselectiebits van alle RIC's. Deze bits bepalen de 'device' adressen. In het totaal kunnen we 16 kaarten met elk 4 device adressen aanspreken. Door gebruik te maken van de bit PC0 in de kaartadresdecoder kan, indien zulks gewenst is, het aantal te adresseren kaarten tot 32 opgevoerd worden. Het bit PC0 draagt de naam van ' bus expand ' bit.
-De bits PC1 en PC2 ten slotte zijn verbonden met de $\overline{RD}$ en $\overline{WR}$ klemmen van de RIC's. Het schema geeft een overzicht van de gebruikte techniek.
Op elke kaart kan een toepassing ondergebracht worden.

Kaartadresdecoder.

De kaartadresdecoder bestaat uit 4 exclusieve OF poorten met 2 ingangen. Een van de ingangen van de poorten is verbonden met een van de 4 meest beduidende bits van de B poort, de andere ingang kan met een schakelaartje op '0' of '1' gezet worden. De 5 ingangen van de NAND poort moeten hoog zijn, wil CS laag gaan. Met de 4 schakelaartjes kunnen 16 combinaties verkregen worden. Bij elk van deze combinaties kan CS maar laag gaan voor een wel bepaalde code van de bits PB4 tot PB7. Elke kaart kan maar geselecteerd worden voor een adres. Merken we op dat deze redenering slechts opgaat als PC0 eveneens hoog staat. Als in de PC0 lijn een invertor wordt opgenomen dan verkrijgen we weer 16 adressen maar nu met PC0 laag. We kunnen dan 32 kaarten adresseren.

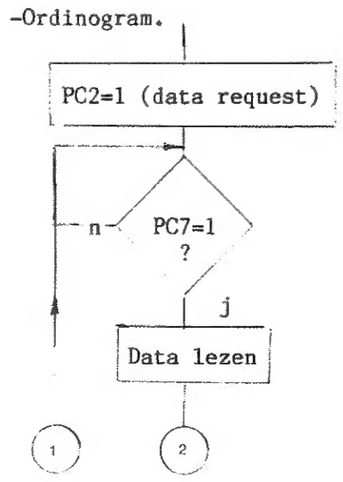


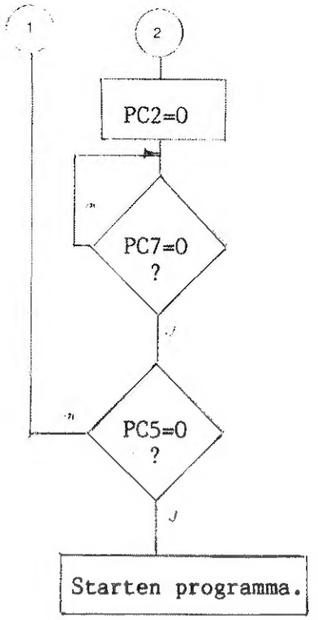
Firmware ter ondersteuning van RWC.

Bij resetten van de DAI gaat het opstartprogramma kijken of er apparaten op de DCE bus zijn aangesloten. Indien ja dan gaat de routine kijken of door een van de aangesloten kaarten, data aangeboden wordt Indien dit het geval is dan worden deze data als programma ingelezen en onmiddellijk daarna uitgevoerd. Worden geen data aangeboden dan wordt na een korte wachttijd de initialisatie verder afgewerkt.

Initialisatieroutine.

-Ordinogram.





-Bespreking initialisatieroutine.
Na initialisatie van de GIC (PA en PCh input - PB en PC1 output) worden de bits PC3, PB1 en PC0 hoog gezet. Een van deze bits kan bijvoorbeeld gebruikt worden om op de RWC kaart een latch aan te sturen. Deze latch kan dan de data bevatten die aan het kanaal A van de GIC aangeboden worden. Als deze data moeten gelezen worden moet PC5 laag gaan. Gaat PC5 niet laag, dan wordt, na een tijd bepaald door de inhoud van register B, de DCE initialisatieroutine verlaten en de opstartprocedure verder gezet. Gaat PC5 laag dan wordt naar de inleesroutine gesprongen. Transmissie van de data gebeurt onder controle van zogenoemde 'handshake signalen'. Door het hoogzetten van bit PC2 verwittigt de GIC het periferieapparaat dat data mag aangeboden worden (data request). DE data worden aangeboden (strobe) door het hoogzetten van PC7. Nadat de data ingelezen zijn wordt PC2 terug laag gezet (data acknowledge). Als tenslotte het periferieapparaat, PC7 terug laag zet, is het transfert van een byte afgelopen. Zolang PC5 laag blijft, worden verdere data ingelezen. Gaat PC5 hoog, dan wordt het

ingelezen programmatje gestart.

Transfert ondersteuningsroutines

Naast de initialisatieroutine zijn in het softwarepakket vam de DAI pc een RWC lees- en een RWC schrijfroutine opgenomen . Deze routines worden benut door de BASIC commando's INPUT en OUT.

- input routine RWCIN (:D8E0).

Deze routine brengt 1 byte van een gegeven RWC adres naar register E. Vooraleer deze routine kan uitgevoerd worden, moet D geladen worden met het RWC adres (kaartadres,device adres)

```
RWCIN  PUSH PSW
       PUSH H
       LXI  H,:FE03 Initial. GIC
       MVI  A,:90   PA input
       DCX  H       Adrespoint. PC
       MVI  M,:FE   bus expand = 0
       MOV  A,D     busadres in A
       STA  :FE01   en naar PB.
       INR  M       bus expand = 1
       MVI  M,:FB   RD aktief
       LDA  :FE00   Data naar A
       MOV  E,A     en naar E
       MVI  M,:FF   desaktiveren RD
       DCR  M       deselect. kaart
       POP  H
       POP  PSW
       RET
```

- Outputroutine RWOUT (:D8C8)

Deze routine brengt een byte vanuit het register E naar een bepaald RWC adres. Vooraleer de routine uit te voeren, dient D geladen te worden met het kaartadres, en E met de data.

```
RWOUT  PUSH PSW
       PUSH H
       LXI  H,:FE01 Initial. GIC
       MVI  M,:80   alles output.
       DCX  H       adrespointer PC
       MVI  H,:FE   busexpand = 0
       XCHG         busadr H,data L
       SHLD :FE00   adr  B, data  A
       XCHG
       INR  M       busexpand = 1
       MVI  M,:FD   WR aktiveren
       MVI  M,:FF   WR desaktiveren
       DCR  M       busexpand = 0
       POP  H
       PUSH PSW
       RET
```

Voorbeelden DCE-RWC.

Om snel te kunnen werken worden veel gebruikte programma's soms opgeslagen in EPROM's. Deze programma's worden dan in geheugen geladen om ze te kunnen gebruiken.
Veronderstel dat op de DCE bus een kaart is aangesloten waarop acht EPROM's van het type 2732 (4Kbyte) kunnen ondergebracht worden. Uit het schema blijkt dat kanaal A van de RIC verbonden is met de datalijnen van de EPROM's, Kanaal B en C1 zijn verbonden met de adreslijnen en de bits PC4, PC5 en PC6 tenslotte zijn verbonden met de ingangen van een decoder 3 naar 8 (74LS138) De uitgangen van deze decoder worden gebruikt om de EPROM's te selecteren. Onze bedoeling is om een LOADER te schrijven dat een willekeurig adressenbereik uit deze EPROM's in geheugen kan lezen.
De adressen voor de verschillende EPROM's op de kaart zijn samengevat in volgende tabel.

EPROM 0	0000	tot	0FFF.
EPROM 1	1000	tot	1FFF.
EPROM 2	2000	tot	2FFF.
EPROM 3	3000	tot	3FFF.
EPROM 4	4000	tot	4FFF.
EPROM 5	5000	tot	5FFF.
EPROM 6	6000	tot	6FFF.
EPROM 7	7000	tot	7FFF.

We gaan de LOADER zo schrijven dat als U het programma start het zich meldt met LOAD. Er wordt dan van U verwacht dat U 3 adressen intikt. Het eerst adres geeft het begin aan van het data blok dat van de EPROM naar het geheugen dient gebracht te worden. Het tweede adres is het eindadres van dit blok. Het derde adres ten slotte is het beginadres van de plaats waar het programma in het geheugen van de DAI PC wordt ondergebracht. Als deze drie adressen ingetikt zijn, wordt het programma geladen en automatisch gestart.

-Listing van de LOADER

In het programma maken we gebruik van ROM routines uit de DAI-PC Vermits het programma dat we wensen te schrijven onder Utility gebruikt wordt,nemen we ons voor alleen routines te gebruiken uit dit zelfde Utility programma (Bank 3)
Uitzondering op deze regel zijn de input-en outputroutine voor de RIC.

Gebruikte routines.

ADARG :EADE
Deze routine leest een aantal argumenten (adressen) in, overeenkomstig het getal dat in het register C staat. Deze adressen worden in volgorde van inlezen op de STACK gezet

LIST$:ED2F
Deze routine list een karakterstring op het scherm. Het beginadres van deze string dient in het HL registerpaar te staan, en de string dient afgesloten te worden met 00.

Assemblerlisting van LOADER

```
;
; Programmaveranderlijken
;
AIN     EQU   :1390  GIC als input
AOUT    EQU   :1380  GIC als output
RICADH  EQU   :12    adres RIC PC
RICADL  EQU   :11    adres RIC PB
RICDAT  EQU   :10    adres RIC PA
RICIN   EQU   :D8E0  Leesrout. RIC
RICOUT  EQU   :D8C8  Schrijfr. RIC
LIST$   EQU   :ED2F
ADARG   EQU   :EADE
*
* Afdrukken LOAD
*
        LXI   H,MES  adr. MES in HL
       CALL   :ED2F  afdrukken  MES
*
* Inlezen 3 adressen
*
       MVI   C,3     wensen 3 adr.
       CALL  ADARG   te  lezen.
*
* Berekening lengte  te  lezen blok
* en naar BC brengen
*
       POP   H       Load adr. RAM
       POP   B       Eindadres blok
       POP   D       Beginadr. blok
       SHLD  STARTAD Save Startadr.
BERLEN MOV   A,C
       SUB   E       C-E
       MOV   C,A     naar C
       MOV   A,B
       SBB   D       B-D-carry
       MOV   B,A     naar B
*
* Lezen van blok uit EPROM
*
RDBLK  PUSH  D       Adr. stack
       CALL  OUTPUT
*
*  Uitgeven  LSB  van adres  EPROM.
*
ADLOUT POP   D
       MOV   A,D
       PUSH  D
       MOV   E,A
       MVI   RICADL
       CALL  RICOUT
*
*  Uitgeven MSB  van  adres  EPROM.
*
ADHOUT POP   D
       MOV   A,E
       PUSH  D
       MOV   E,A
       MVI   D,RICADH
       CALL   RICOUT
*
* Lezen van databyte.
*
DATIN  CALL  INPUT
       MVI   D,RICDAT
       CALL  RICIN
       MOV   M,A
*
* Aanpasen adrespointers
* Testen of laatste byte ?
*
PNTRS  POP   D
       INX   H
       INX   D
       PUSH  D
       MOV   A,C
       ORA   B
       JNZ   ADLOUT
       POP   D
*
*  Starten  gelezen  programma.
*
       LHLD  STARTAD
       PCHL
*
* Hulproutines
*
```

```
INPUT  LXI   H,AIN
       CALL  RICOUT
       RET
OUTPUT LXI   H,AOUT
       CALL  RICOUT
       RET
*
* Data en pointers
*
STARTAD RES  2
MES     DATA :0C
        ASC  'LOAD'
        DATA 00
        END
```

DNA SPACE COMPRESSING

Willi Herrmann
Richard-Dehmel-Str.4
D-4320 Hattingen

MAI 1985

Textkomprimierung beim DNA-Assembler :

In einem Source-Text eines DNA-Assembler-Files sind sehr viele überflüssige SPC (ca. 20-40%).
Dieses Programm löscht alle überflüssigen SPC.

Start : 1. Load Assembler
2. Load Assembler-Source
3. BASIC
4. Load 'TEXTKOMPR. DNA-Assembler'
5. Run (Programm nicht unterbrechen !)
6. <SPACE> drücken
7. Save Assembler-Source

```
1     REM
2     REM Textkomprimierer fuer DNA-Assembler
3     REM
4     REM Willi Herrmann
5     REM Richard-Dehmel-Str.4
6     REM D-4320 Hattingen
7     REM
8     REM **** IMP INT ****
9     REM
100   CLEAR 256
110   Z=PEEK(#123B)+256*PEEK(#123C)
120   N=PEEK(#1245)+256*PEEK(#1246)
130   X=2:M=N:FLAG=0
140   A=PEEK(M):B=PEEK(M+1)
150   IF PEEK(M-1)=#8D AND A=#AA THEN FLAG=1
160   IF A=#A7 THEN FLAG=-FLAG+1
170   IF FLAG=0 AND A=#A0 AND B=#A0 THEN M=M+1:GOTO 140
180   PRINT CHR$(A IAND #7F);
190   POKE N,PEEK(M):M=M+1:N=N+1
200   IF A=#8D THEN PRINT X;TAB(6);:FLAG=0:Z=Z-1:X=X+1
210   IF Z>0 GOTO 140
220   PRINT :PRINT "The Source is";M-N;" Bytes shorter"
230   CALLM #D6DA:CALLM 12000
```


X-BUS PROJECT

translation by Georges COCHIN

CHOPPINET Eric
27 rue Louis Pasteur
91310 LEUVILLE SUR ORGE
FRANCE
Tel : (6) 084 60 87

Leuville , April 20th 1985

Dear DAI friend

I thank you for taking interest in the EXTENSION-BUS card published in issue No 26 of DAInamic.

The delivery time of the unequipped card made with metallized holes is 6 weeks after receipt of your order.

The price in french francs is :

To fit up the card , you should get at hand :
a multimeter ,
if possible , an oscilloscope,
a scraper to cut off 3 tracks on the DAI main card,
a good soldering iron (with a very thin pane)
and wrapping wire .

To put on the card ,it is advisable to proceed carefully on a step by step basis as follows :

*** Soldering of all components
(resistors,capacitors ,diods,transistors...)
(connector X-BUS, socket of the integrated circuits.
*** Proceed to the following modifications on the main card :
1. Cut off the track which goes to track No 47 of the X-BUS
2. Cut off tracks going to tracks No 22 and 21 of the X-BUS and link up these strips by passing the X-BUS
3. Connect IC4 pin 9 to pin 47 on X-BUS (F0XX)
4. Connect IC45 pin 10 to pin 18 on X-BUS (F9XX)
5. Connect IC45 pin 11 to pin 22 on X-BUS (FAXX)
6. Connect IC107 pin 1 to pin 21 on X-BUS (reset)

*** Check all modifications with an ohmeter.
*** Switch power on the DAI with MEMOCOM deconnected
*** Check + 5V on each socket of IC
*** Put carefully on and on the right side the following IC :
- CI 5 74C00 !!! CMOS or HCMOS
- CI 6 74C42 !!! CMOS or HCMOS
- CI 3 your EPROM DCR.
!!!! (deconnect the main power) !!!!
*** Get into UT.Display adresses F000 F7FF
The ROM content is displayed, if such is not the case,you probably gave a wrong adress to the decoder (SWITCH A - B - A11 open)
If you have an oscilloscope ,you are able to display (F000-F7FF)
CI 44 pin 9 (main card)
pin 47 on X-BUS
CI 6 pin 12
CI 6 pin 14
*** Connect your MEMOCOM , and CALLM #F2F2

If you are using your DCR ROM , the compatibility is complete as far as software is concerned.
At this stage you cannot switch your EPROM with the dip switches.
POKE# 296,0 and now change the switches position and display adresses F000-F7FF.
Therefore you can change the memory bank but you loose the DCR commands.

It is now possible to introduce the IC 4 INS8154N.

The switches are useful only to initialize the good memory bank on a cold start , after that the 8154 is overriding and sets the pins of B port in a low impedance output state.
Conventionnaly ,always IC 3 with A11 = 5V
The DCR ROM has been completely written again with some extra commands

(RBP restore basic pointer)
(SBP save basic pointer)
(IMP printer initialization //)
(...)

Upon initialization 8154 overrides the commands
(A , B , A11 at low impedance)

The new DCR ROM allows all the functions

(LOOK) (LAST) (VERIFY) (CHECK) (LOAD) (SAVE) (CASx) (DCRx)

There is an eventuality to test if the DCR eprom is present in some programs with a Byte check in the memory bank F000-F7FF. This byte might be different now.

As far as the eprom programmer is concerned, I can provide you the corresponding software for EPROMs in machine language that you'll run with CALLM #7000

Answer the Menu :

L reading in #4000 - #4FFF
V checking
T blankness test
P programming with data in #4000
F end : back to monitoring

Be careful with the programming voltage (+21V or +25V) This programming voltage should be applied at the proper time (you'll be warn by the software) and should be cancelled as soon as the program gives you back the controls.You should use a +30V power unit, the LM137 (T1) controller and the (P1) potentiometer to allow you to get 21V or 25V.(A preliminary test without eprom is advisable)

Concerning the real time clock,the software is now under design. The IC 9 (MM58174A) is adressed from FAC0 to FAFF. Please refer to manufacturer's instruction leaflet.The QUARTZ is 32.768 kHz (watchmakers standard).
The clock goes on working even if the main power is switched off,I rely upon you to work out some new interesting applications.

A word about the 2K RAM use.
In order to write in the F000-F7FF memory space , you should POKE #10,#C9 ta avoid a STACK OVERFLOW and carefully check that SWITCH 'WR' be closed , then POKE #10,0.
When you open SWITCH 'WR' the RAM content is protected in case of power failure. This RAM is useful to run some programs you want to check before before writing them in ROM , not to store DATA.
To store variables the IN8154N RAM should be used,

All the software programs related to this circuit board were worked out by my friend Claude PICARD.

You'll find more informations about this circuit board in the next issues of DAInamic.

Friendly yours ,

CHOPPINET Eric

LISTE DES COMPOSANTS

Résistances

R1 ..... 100 Ω
R2 ..... 1,5 kΩ
R4 ..... 470 Ω
R5,R6,R7..... 4,7 kΩ
R8,R9 ..... 47 kΩ
R10 ..... 4,7 kΩ
R11,R12,R13 .. 47 kΩ
R14 ..... 1 kΩ
R15 ..... 4,7 kΩ
R16,R17 ..... 1 kΩ
R20 ..... 4,7 kΩ
R21 ..... 10 kΩ
R22 ..... 470 Ω
R23 ..... 10 kΩ
R24 ..... 100 Ω

Condensateurs

C1,C2...1 µF – 10 V (Ta)
C3,C4...10 nF
C5 .....1000 µF – 16 V
C6 .....100 nF
C8 .....10 pF ou 30 pF ajust.

Semi-conducteurs

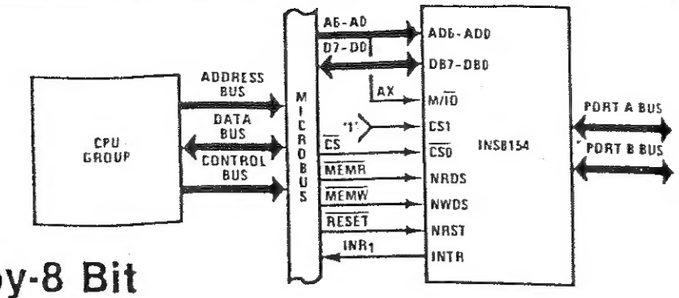
D1,D2,D3 .....1N 4148
D6,D7 .....BZX 3,9
D8 .....OA 81 (Ge)

T1 .....TDB 0117
T2,T3.....BC 548
IC0..... 6116
IC1,IC2,IC3 2716
....2732
....2764
IC4.....INS 8154 N
IC5.....74 C 00
IC6,IC10 .....74 C 42
IC7.....4040
IC8.....*EPROM vierge*
IC9.....MM 58174 A

Divers

Support à insertion nulle (28 br.)
Inter DIL (4 inters)
Connecteur 50 broches (2 x 25; pas 2,54)
Quartz 32,768 kHz

National Semiconductor



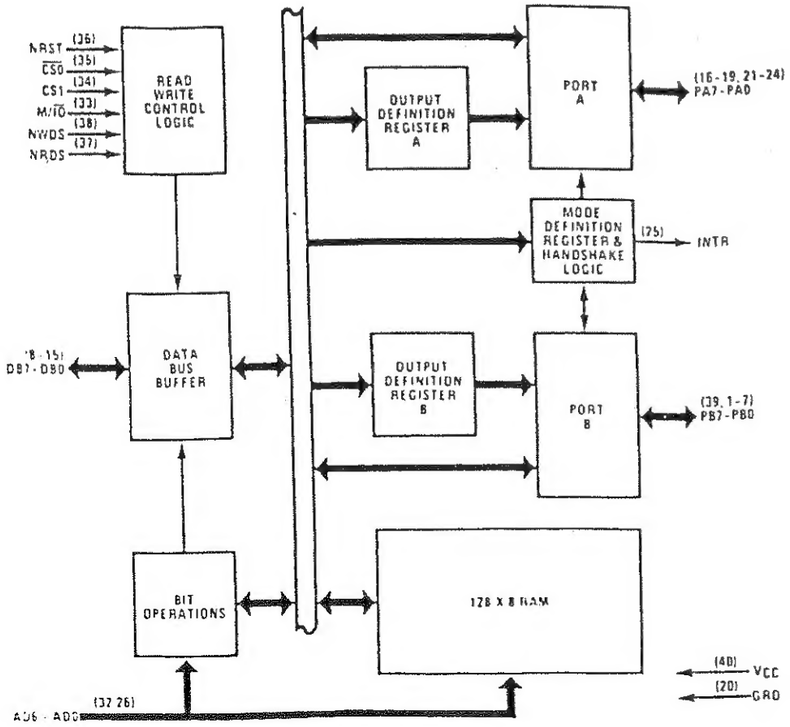
INS8154 N-Channel 128-by-8 Bit RAM Input/Output (RAM I/O)

General Description

The RAM Input/Output Chip is an LSI device which provides random access memory and peripheral interfacing for microcomputer systems. The RAM portion contains 1024 bits of static RAM organized as 128x8. The I/O portion consists of two peripheral ports of eight bits each. Each of the I/O pins in the two ports may be defined as an input or an output to provide maximum flexibility. Each port may be read from or written to in a parallel (8-bit byte) mode. To improve efficiency and simplify programming in control-based applications, a single bit of I/O in either port may be set, cleared or read with a single microprocessor instruction. In addition to basic I/O, one of the ports, port A, may be programmed to operate in several types of strobed mode with handshake. Strobed mode together with optional interrupt operation permit both high speed parallel data transfers and interface to a wide variety of peripherals with no external logic.

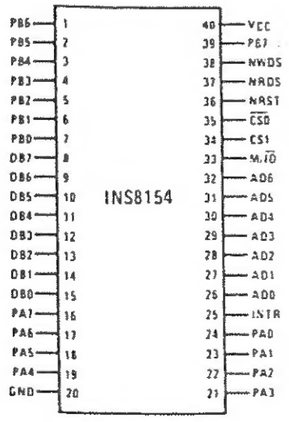
Features

- 128x8 RAM
- Single +5-volt power supply
- Low power dissipation
- Fully static operation
- Completely TTL compatible
- Two 8-bit programmable I/O ports
- I/O port A has TRI-STATE® capability
- Handshake controls for strobed mode of operation
- Single bit I/O operations with single instruction
- Reduces system package count
- Direct interface with SC/MP
- Independent operation of RAM and I/O
- MICROBUS™* Compatible



NOTE: APPLICABLE PINOUT NUMBERS ARE INCLUDED WITHIN PARENTHESES.

Pin Configuration



Pin Names

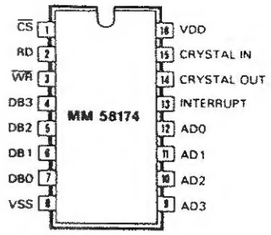
DB7 - DB0	DATA BUS
AD6 - AD0	ADDRESS INPUT
NRST	RESET INPUT
M/IO	MEMORY/IO SELECT
CS0, CS1	CHIP SELECTS
NWDS	WRITE STROBE
NRDS	READ STROBE
PA7 - PA0	PORT A
PB7 - PB0	PORT B
INTR	INTERRUPT REQUEST
VCC	+5 VOLTS
GND	0 VOLTS

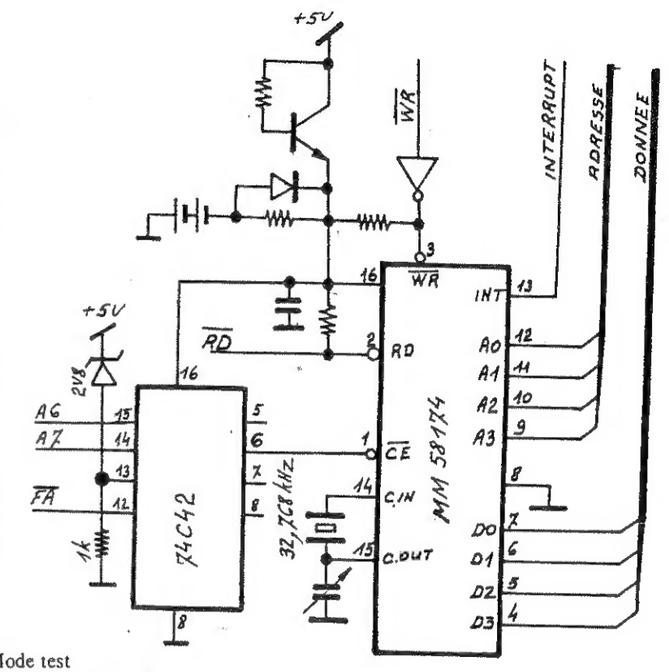
L'horloge « temps réel »

La fonction horloge « temps réel » est réalisée à l'aide d'un circuit spécialisé (58174A). Ce « compteur de temps » a déjà fait l'objet d'une description dans *Micro-Systèmes* (n° 21, p. 128). Rappelons toutefois, comme le montre la **figure 4,** que ce circuit contient, dans ses seize registres internes, tous les paramètres du « temps », des dixièmes de seconde jusqu'aux mois. Il ne permet cependant pas la lecture de l'année, bien que celle-ci figure dans un registre interne pour le calcul de la durée du mois de février.

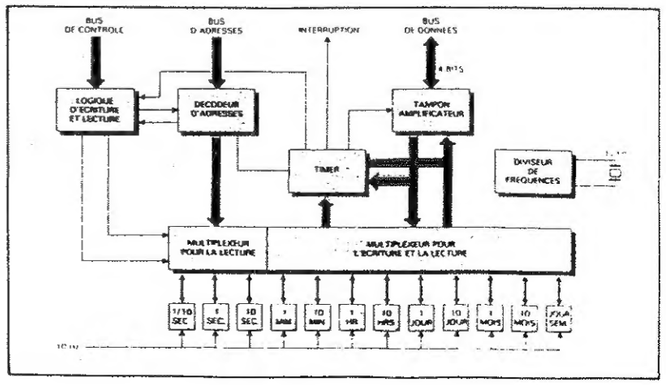
L'oscillateur interne, piloté par un quartz externe de 32,768 kHz (fréquence standard en horlogerie électronique : 2^{15}), synchronise une suite de diviseurs et de compteurs constituant l'ensemble des seize registres que nous venons d'évoquer. La fonction et l'adresse de chacun de ces registres sont détaillés **tableau 6.** Le brochage de ce circuit d'horloge est présenté **figure 5 a.**

La sauvegarde des données de ce boîtier peut être effectuée soit par une batterie rechargeable placée sur la carte mère, soit par une pile ordinaire de 4,5 V.





FA40	Horloge « temps réel »	Mode test
FA41		Dixièmes de seconde
FA42		Secondes (unités)
FA43		Dizaines de secondes
FA44		Minutes
FA45		Dizaines de minutes
FA46		Heures
FA47		Dizaines d'heures
FA48		Jours
FA49		Dizaines de jours
FA4A		Jours de la semaine
FA4B		Mois
FA4C		Dizaines de mois
FA4D		Année
FA4E		Marche/arrêt
FA4F		Interruption et indicateur d'états



PROGRAMMING IN ASSEMBLER

A course presented in parts, by UWE WIENKOP.

Original : German.
Translation : G.Doumont

Programs with small assembler subroutines turn up frequently in the Club journal. But also the programs which are offered for sale are very often either completely written in machine language or use subroutines in Assembler. And precisely, these routines are most of the time the most important part of the program : it is therefore a pity when one doesnot understand that kind of programming. Besides, programmers which are intensely engaged in machine language programming are often compassionately laughed at : their programming language being not up-to-date, etc....

In order to remove these prejudices, and also in order to provide some knowledge in the use of the many embodied functions of the DAI, I have decided to write this series. Besides, I also want to show that this kind of programming is not only very fast, but at the same time not so difficult as one always hears and, last but not least, that it can be very exciting.

But before a description of the instructions can be initiated, a few theoretical notions have to be dealt with : it is necessary to go a little into number systems and into conversion methods.

0. GENERAL THEORETICAL NOTIONS.

0.1. Number systems and conversion methods.

In DAI BASIC there exist numbers in two different numbersystems : on the one hand the "normal" number with base 10 and on the other hand the hexadecimal numbers. The latter ones are identified by means of a "#" placed before them. Hexadecimal means that the number is represented in base 16.

The base of a number system indicates how many relevant figures are at disposal in order to represent a given number. It is well known that in the decimal system, with which we usually calculate, there are indeed 10 possible figures : 0,1,.....9. In the hexadecimal system there must accordingly exist 16 figures. No figures being known other than 0,...,9, letters A,...,F have been borrowed, so that the figures :

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

are at disposal.

In the computer there exists in addition another important number system : one could even say that it is the most important one, after all. It is the binary system. Here there exist only two figures : 0 and 1.

In BASIC one has mostly to deal with number representation in the decimal system. Computers, however, are not generally very concerned with that number system. Internally, they can only differentiate between two states for instance voltage high and voltage low. From these two states, greater numbers can be formed by means of series of figures placed behind one another, while signs and figures can be formed by more abstract considerations.

Consequently, it is necessary to convert every number into the corresponding binary equivalent. Sequences of commands such as for instance a PRINT statement are for a computer only an abstract concept with which he cannot start anything at all. It is only after the proper conversion and a later processing by an operation system that the statement becomes understandable by the computer. In order to understand better this conversion and the later work with numbers, it is meaningful now to discuss the conversions.

In the decimal system, every place corresponds to a definite "valence", which is to be multiplied by the figure found at that place. The sum of all products so obtained gives finally the value of the number. This valence gives the other significance of the base : the base, when raised to the power of the place, is equal to the valence. This looks very complicated, but is in fact very simple. Let us give an example :

Take the number 1243. The indexation of the figures starts at the last figure to the right (3), which index is taken as 0. Proceeding to the left with increasing index, (4) gets index 1, (2) gets index 2, and (1)gets index 3. As indicated above, the valences are obtained by raising the base to the powers of the indexes,

that is :	10↑3	10↑2	10↑1	10↑0
or	1000	100	10	1

We then multiply these valences with the corresponding figures and sum up the products :

```
1*1000  +  2*100  + 4*10  +   3*1 =  1243
```

At this point, you will certainly ask yourself : nothing special in there, I knew that before... Nevertheless, this method has also to be applied in another number system, for instance in order to find out which value a number expressed in the hexadecimal system will have in the decimal system.

Take as an example the number : #BFEF which represents he highest screen RAM address. If we calculate this number according to the above method, we obtain :

```
 B*16↑3    +   F*16↑2  +   E*16↑1   +   F*16↑0 =
 B*4096    +   F* 256  +   E*  16   +   F*   1 =
11*4096    +  15* 256  +  14*  16   +  15*   1 = 49135
```

You can easily check this from BASIC by doing

```
PRINT#BFEF
```

as PRINT always expresses numbers in decimal, and therefore converts and gives the number in the decimal system.

Now let us consider the opposite case : to convert a decimal number into another number system. The method used is also very easy. One divides the decimal number by the required base(integer division), and one puts the rest aside. The process is continued using the quotient until the result of the division gives zero. If one then writes the sucesive rests in opposite sequence, one obtains the number converted to the desired base.

Let us also here take an example : converting back the above number 49135

into the hexadecimal system.

```
 49135  / 16 = 3070  + 15        15 = #F
  3070  / 16 =  191  + 14        14 = #E
   191  / 16 =   11  + 15        15 = #F
    11  / 16 =    0  + 11        11 = #B
Result : #BFEF as expected.
```

The conversion into the binary system works out the same way, except that the base is here equal to 2. Let us take 97 as an example :

```
  97  /  2 =   48  +  1
  48  /  2 =   24  +  0
  24  /  2 =   12  +  0
  12  /  2 =    6  +  0
   6  /  2 =    3  +  0
   3  /  2 =    1  +  1
   1  /  2 =    0  +  1
Result :  (97) decimal = (1100001) binary
```

Now, one question can be asked : it was said above that the binary system was the most important one for the computer : why then have we got to be burdened by another number system, that is, the hexadecimal system? The answer to this is very simple. As one has noted from the quoted examples, numbers expressed in the binary system are unsuitable for human beings : rather long binary numbers already appear when dealing with rather small decimal numbers. In order to remove this shortcoming, the binary figures have been grouped four by four into hexadecimal figures. The number becomes four times shorter and easier to survey. For instance, the above number becomes :

```
      110 0001
       #6   #1          so that  (97) decimal = #61
```

For this reason, in the DAI Utility, the contents of registers and memories are given in hexadecimal form. Computing in hexdecimal (or in binary) has another advantage. Through the development of the memory in the computer and its dependance on the binary system, computing in the hexadecimal and binary system gives rise to values which are better rounded off than in the decimal system. Consider for instance the following example : the ammount of memory is always expressed in kilobytes, or in K's. One K =1024 Bytes. 1024=2↑10=#400 etc...

0.2 Units of information.

As was already briefly announced above, a binary digit is the smallest unit of information; she is designated by the word "bit". As one can do very little with one bit, 8 bits were put together. This gives rise to one "byte". According to the above mentionned computing method, the greatest number to be represented by means of 8 bits is 255 :

1*2↑7+1*2↑6+1*2↑5+1*2↑4+1*2↑3+1*2↑2+1*2↑1+1*2↑0 =255

There are in fact in total 256 possibilities, as the zero also has to be considered.

The next greater unit is a 16 bit word. Here we have 16 bits at disposal in order to represent a number, and the greatest is correspondingly 2↑16-1=65535.

There are of course in machine language also negative values. These numbers will be considered in the section corresponding to substraction.

0.3 Assemblers and their functions.

As you probably have already noted from the software library of the Club, DAInamic Germany offers an Assembler of its own (AHT). Similar products can also be obtained from DAInamic Belgium under the names DNA and SFL. These products distinguish themselves in fact by their user friendliness for input, expressions, etc. Anyway,they all fulfill equally well their duty.

As was already explained above, all instructions must be converted in binary form. As writing an instruction under that form makes it very uneasy to survey - one must identify the number with an instruction - an "assembler"language for machine instructions has been introduced. The assembler has the task of conveying an understandable and readable instruction to these binary figures. Because the instruction sets of the current microprocessors vary very much from one to another, and because the manufacturers introduce different assembler languages, there exists for each known processor a proper assembler language with very different instructions.

This was one of the reasons to introduce the so-called high level languages. They not only facilitate the processing of programs, but they also are to a great extent portable. As result, a BASIC program that was for instance written on an APPLE without going into the specific characteristics of that computer can relatively easily be transfered on another computer.

Now back to the assembler language. One distinguishes here two kinds of programs :

1)-SOURCE code : the source of a program contains the readable text that you have produced when working out the program. That text can as in the BASIC Editor be corrected, etc... However, that text cannot be run directly. It must first be assembled.

2)-OBJECT code : this is then the assembled source text and it can be run directly. The object code has on the source code the advantage of being 3 to 4 times shorter, but it can only be corrected with difficulty, as the instructions now appear under their translated form. You can, for fun, call the Utility of the DAI and from there, have a look on the range #C000 to #EFFF. A 24 kilobytes object code is to be found here.

The instructions of the source code are also called "mnemonics" as they serve as "supports of the memory" for the corresponding machine instructions.

An assembler has also, next to the translation function, other important tasks, but these will be dealt with at the proper place.

(to be continued)

TEKSTVERWERKER IN BASIC

```
1       REM ,TEKSTVERWERKINGSPROGRAMMA
2       REM ,
3       REM ,Hierbij de listing van een tekstverwerkings-
4       REM ,programma ter plaatsing in DAINAMIC.De toelich-
5       REM ,ting op de regels 63024 en volgende samen met
6       REM ,de keuzemogelijkheden op de regels 60006 t/m 60019
7       REM ,spreken voor zichzelf.De printer (in mijn geval
8       REM ,de microline 80) is aangesloten op de RS 232 uit-
9       REM ,gang van de DAI.Deze tekst is geproduceerd door
10      REM ,achtereenvolgens de programma's 1,2 en 7 te ge-
11      REM ,bruiken.De listing demonstreert hoe de tekst is
12      REM ,opgeborgen in REM-statments.Statement 16 bevat de
13      REM ,commandoregel,die ten behoeve van de listing de
14      REM ,printer heeft geplaatst in de toestand 16.5 charac-
15      REM ,ters per inch.
16      REM ,
17      REM ,R.P.Offereins, Wilmskamp 13, Hengelo (O), Nederland
18      REM ,:CS
60000   REM ************** TEKSTVERWERKING  30/12/84 **************
60001   POKE #29B,236:PRINT CHR$(12):POKE #29B,255
60005   POKE #131,1:COLORT 15 0 0 0:NL=0:BG=59999:GOSUB 61816
60006   PRINT "Aantal aanwezige tekstregels = ";NL
60007   PRINT "Type een van de volgende commandonummers."
60011   PRINT "1  EDIT.Tekstregels naar editbuffer."
60012   PRINT "2  STORE.Editbufferregels naar tekstgeheugen."
60013   PRINT "3  DELETE.Verwijder regels uit tekstgeheugen."
60014   PRINT "4  DELETE & STORE."
60015   PRINT "5  REPLACE.Verplaats tekstregels."
60016   PRINT "6  PRINT tekstgeheugen MET regelnummers."
60017   PRINT "7  PRINT tekstgeheugen ZONDER regelnummers."
60018   PRINT "8  PRINT rechtstreeks regels."
60019   PRINT "9  Toelichting."
60020   INPUT PRGNR!:PRINT
60025   ON PRGNR! GOTO 60060,61100,61900,62300,62100,61200,61205,62500,63020
60030   PRINT "Verkeerd commandonummer ,opnieuw typen.":GOTO 60020

60059   REM ------------------------------------EDIT
60060   POKE #29B,236:CLEAR 6000:POKE #29B,255:CALLM #E280:POKE #131,1
60061   ADE=2+PEEK(#29B)+256*PEEK(#29C)
60062   INPUT "Type beginnummer,eindnummer";BG,ED
60063   IF ED>59999 OR BG>ED THEN PRINT "Fout nummer":GOTO 60062
60064   ADT=PEEK(#29F)+256*PEEK(#2A0)
60065   GOSUB 60066:CALLM #E1FB:REM --------END
60066   NR=256*PEEK(ADT+1)+PEEK(ADT+2)
60068   IF NR>ED THEN RETURN
60070   IF NR<BG THEN GOTO 60076
60072   FOR I=2 TO PEEK(ADT+4):POKE ADE+I-2,PEEK(ADT+4+I):NEXT I
60074   ADE=ADE+PEEK(ADT+4)-1:POKE ADE,13:ADE=ADE+1
60075   POKE #A4,PEEK(VARPTR(ADE)+3):POKE #A5,PEEK(VARPTR(ADE)+2)
60076   ADT=ADT+1+PEEK(ADT):GOTO 60066

60499   REM ----------------SUB PRINTERSTART
60500   PRINT CHR$(30);
60510   PRINT CHR$(27);:PRINT CHR$(65);:PRINT CHR$(27);
60511   PRINT CHR$(54);:RETURN

60899   REM -----------------SUB NUMMERS TOEVOEGEN
60900   A=LEN(STR$(NR))
60905   FOR I=1 TO A-3
60910   CR!=ASC(RIGHT$(STR$(NR),A-I)):POKE ADE+I,CR!:NEXT I
```

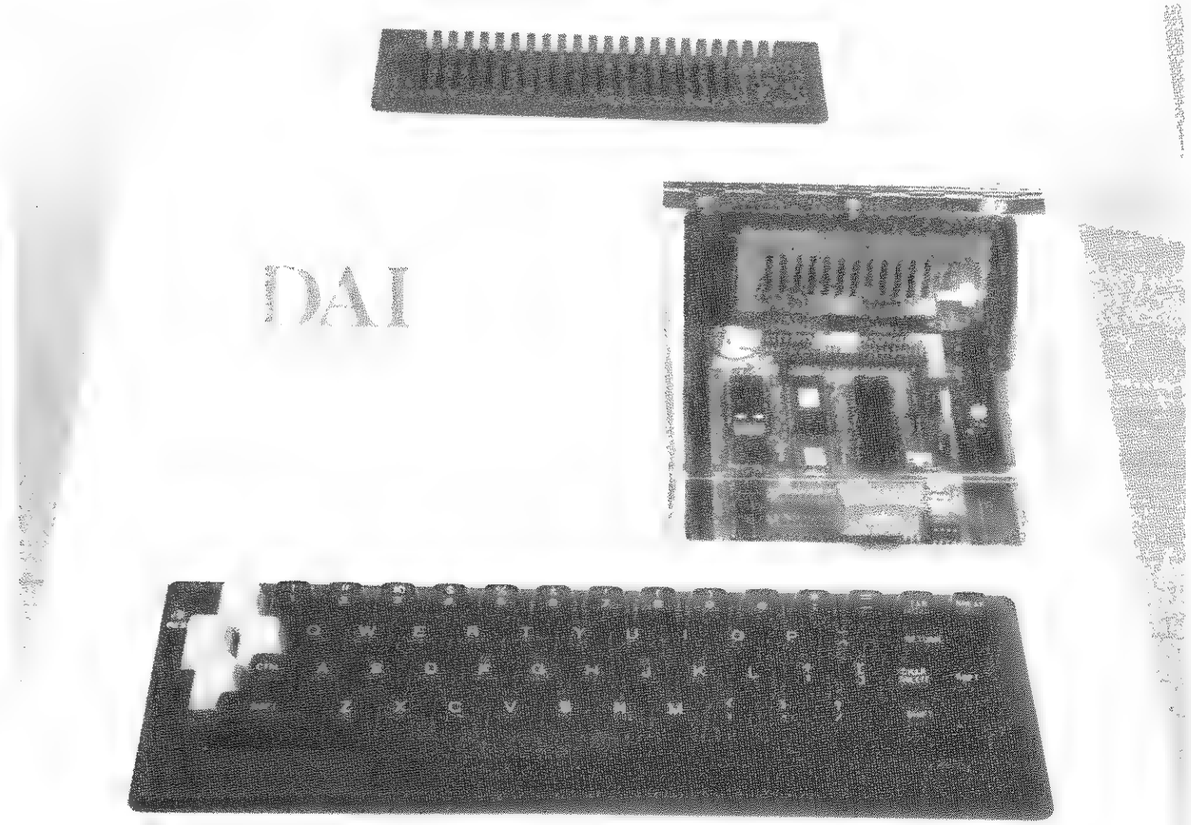


```
60920   POKE ADE+A-2,82:POKE ADE+A-1,69
60930   POKE ADE+A,77:POKE ADE+A+1,44
60940   NR=NR+1:ADE=ADE+A+1:RETURN

61099   REM ------------------------------------STORE
61100   POKE #29B,236:INPUT "Type  beginnummer ";NR:BG=NR:PRINT
61106   IF NL<NR THEN NR=NL+1:BG=NR
61107   GOSUB 61110:NR=NR-1:GOSUB 61816
61108   PRINT "Starten met RUN":POKE #135,2:END
61110   N1=PEEK(#29B)+256*PEEK(#29C)+21
61120   ED=PEEK(#A4)+256*PEEK(#A5)-1:ADE=ED:GOSUB 60900
61130   FOR AD=N1 TO ED:CR!=PEEK(AD):ADE=ADE+1:POKE ADE,CR!:POKE AD,65
61140 1  IF CR!=13.0 AND AD<ED THEN GOSUB 60900
61150   NEXT AD:POKE ED,13:ADE=ADE+1:RETURN

61199   REM ---------------------------PRINT MET/ZONDER NUMMERS
61200   FLG1!=1.0
61205   INPUT "Type beginnummer,eindnummer ";BG,ED
61206   IF ED<BG OR ED>59999.0 THEN PRINT "Fout nummer,type opnieuw":GOTO 61205
61208   PRINT :PRINT :POKE #131,0:GOSUB 60500:C1=123:C2=125
61210   ADT=PEEK(#29F)+256*PEEK(#2A0)
61215   NR=256*PEEK(ADT+1)+PEEK(ADT+2)
61220   IF NR>ED THEN FLG1!=0.0:POKE #131,1:END
61225   IF NR<BG THEN GOTO 61260
61231   IF FLG1!=0.0 THEN FOR I=0 TO 7:PRINT CHR$(32);:NEXT I
61232   IF FLG1!=0.0 AND PEEK(ADT+6)=58 THEN GOSUB 61600:GOTO 61260
61233   IF FLG1!=1.0 THEN GOSUB 61700
61235   IF PEEK(ADT+4)=1 THEN GOTO 61255
61240   FOR I=2 TO PEEK(ADT+4):CR!=PEEK(ADT+4+I)
61243 1  IF CR!=91 THEN CR!=C1
61245 1  IF CR!=93.0 THEN CR!=C2
61250   PRINT CHR$(CR!);:NEXT I
61255   PRINT CHR$(13);:NRA=0
61260   ADT=ADT+1+PEEK(ADT):GOTO 61215

61599   REM ----------------------SUB COMMANDODETECTIE
61600   A=PEEK(ADT+7):B=PEEK(ADT+8)
61605   C=100*(B-48)+10*(PEEK(ADT+9)-48)+PEEK(ADT+10)-48
61610   IF A=67 AND B=76 THEN PRINT CHR$(31);
61620   IF A=67.0 AND B=77.0 THEN PRINT CHR$(30);
61630   IF A=67.0 AND B=83.0 THEN PRINT CHR$(29);
61640   IF A=76.0 AND B=76.0 THEN PRINT CHR$(27);:PRINT CHR$(65);
61650   IF A=76.0 AND B=83.0 THEN PRINT CHR$(27);:PRINT CHR$(66);
61660   IF A=83.0 AND B=76.0 THEN PRINT CHR$(27);:PRINT CHR$(54);
61670   IF A=83.0 AND B=83.0 THEN PRINT CHR$(27);:PRINT CHR$(56);
61680   IF A=91.0 AND C<256.0 THEN C1=C
61685   IF A=93 AND C<256 THEN C2=C
61690   RETURN

61699   REM ----------------------SUB NUMMERPRINT
61700   FOR I=LEN(STR$(NR))-1.0 TO 3 STEP -1
61710   PRINT CHR$(ASC(RIGHT$(STR$(NR),I)));:NEXT I
61720   FOR I=LEN(STR$(NR)) TO 10:PRINT CHR$(32);:NEXT I
61730   RETURN

61816   REM ----------SUB NUMMEREN IN TEKSTGEHEUGEN
61820   ADT=PEEK(#29F)+256*PEEK(#2A0)
61830   FOR I=1 TO 10000:A=256*PEEK(ADT+1)+PEEK(ADT+2)
61840   IF A<BG THEN NL=A:ADT=ADT+1+PEEK(ADT):NEXT I
61850   IF A=60000 THEN RETURN
```

```
61860   POKE ADT+2,NR IAND #FF
61870   POKE ADT+1,NR SHR 8:NR=NR+1
61880   ADT=ADT+1+PEEK(ADT):NEXT I

61899   REM ------------------------------DELETE
61900   CLEAR 6000:POKE #29B,236
61905   BG=59999:GOSUB 61816
61910   INPUT "Type beginnummer,eindnummer";NA,NB:PRINT
61915   IF NB>NL THEN NB=NL+1
61920   IF NA>NB THEN PRINT "Fout nummer":GOTO 61910
61925   NR=NA:BG=NB+1:GOSUB 61816
61930   ADE=PEEK(#29B)+256*PEEK(#29C)+21
61940   FOR NR=NA TO NB:B=LEN(STR$(NR))
61950 1  FOR I=B-1 TO 3 STEP -1:POKE ADE,ASC(RIGHT$(STR$(NR),I))
61980   ADE=ADE+1:NEXT I:POKE ADE,13:ADE=ADE+1:NEXT NR
61985   PRINT "Starten met RUN"
61990   POKE #29B,255:CALLM #E280:POKE #135,2:POKE #131,1:END

62099   REM ------------------------------------REPLACE
62100   POKE #29B,236:CLEAR 6000:POKE #29B,255:CALLM #E280:POKE #29B,236
62101   POKE #131,1:ADE=21+PEEK(#29B)+256*PEEK(#29C)
62102   ADT=PEEK(#29F)+256*PEEK(#2A0)
62105   BG=59999:GOSUB 61816
62110   INPUT "Type beginnummer,eindnummer,nieuw beginnummer";NA,NB,NC
62130   IF NA>NB OR NB>NL OR NC>NL THEN PRINT " Nummer te groot":GOTO 62110
62135   IF NC<2*NB-NA+2 AND NC>2*NA-NB-1 THEN PRINT "Verkeerd nummer":GOTO 62110
62145   ADT=PEEK(#29F)+256*PEEK(#2A0)
62150   BG=NA:ED=NB:GOSUB 60066
62153   IF NC<NA THEN BG=NC:NR=NC+NB-NA+1:GOSUB 61816
62154   IF NC<NA THEN NR=NB+1:BG=2*NB-NA+2:GOSUB 61816:NR=NC
62155   IF NC>NB THEN BG=NB+1:NR=NA:GOSUB 61816
62156   IF NC>NB THEN BG=NC-NB+NA-1:NR=NC:GOSUB 61816:NR=NC-NB+NA-1
62160   GOSUB 61110
62170   IF NC<NA THEN BG=NB+1:ED=2*NB-NA+1:NA=BG:NB=ED
62180   GOTO 61940

62299   REM --------------------------DELETE & STORE
62300   POKE #29B,236:INPUT "Type beginnummer,eindnummer";NA,NB
62310   IF NL<NA OR NB<NA THEN PRINT "Fout nummer,type opnieuw":GOTO 62300
62320   IF NB>NL THEN NB=NL
62330   NR=NA:GOSUB 61110:NR=NR-1:NC=NR:BG=NB+1:GOSUB 61816
62340   IF NC>=NB THEN GOTO 61990
62350   NA=NC:GOTO 61930

62500   REM ----------------EEN REGEL TYPEN
62510   PRINT "TYPE GEWENSTE REGEL; /// BETEKENT STOP"
62515   PRINT "IS 'T EERSTE TEKEN EEN SPATIE DAN TUSSEN AANHALINGSTEKENS!"
62520   INPUT A$:IF A$="///" THEN END
62530   PRINT :POKE #131,0:PRINT A$:POKE #131,1:GOTO 62510

63019   REM --------------------------TOELICHTING
63020   PRINT CHR$(12);
63024   PRINT "Na programma 1 krijgt men de EDIT-toestand,waarin men tekst"
63026   PRINT "kan redigeren.Programma 2 (STORE) zorgt dat deze tekst op-"
63028   PRINT "eenvolgend genummerd in het tekstgeheugen wordt opgeslagen"
63030   PRINT "in z.g.REM-statements tesamen met het programma,dat op de"
63032   PRINT "regels 60000 en hoger staat.  Ook na de programma's"
63033   PRINT "3 (DELETE),4 (DELETE & STORE) en 5 (REPLACE)"
63034   PRINT "staan de regels opeenvolgend genummerd in het geheugen"
63035   PRINT "Er zijn tekstregels en commandoregels voor printerbesturing."
```



```
63036   PRINT "Commandoregels  worden in programma 6 (PRINT MET NUMMERS)"
63040   PRINT "wel en in programma 7 (PRINT ZONDER NUMMERS) niet afgedrukt."
63044   PRINT "Voorbeelden van opgeslagen regels zijn dus:"
63048   PRINT "     20REM, tekstregel     "
63049   PRINT "     30REM, commandoregel  "
63050   PRINT "De commandoregels zijn"
63055   PRINT " :CL :CM  :CS   voor brede,normale en smalle letters."
63060   PRINT " :LL :LS      voor lange(20cm) en korte(16cm) regels."
63065   PRINT " :SL :SS    voor grote(4.2mm) en kleine(3.2mm) regelafstand."
63068   PRINT " :]xyz :[abc  teken ] c.q. [ wordt CHR$(xyz) c.q. CHR$(abc)."
63069   PRINT "                                (alleen  in de volgende regel)"
63075   PRINT "De printer pas inschakelen als men wil afdrukken!"
63080   PRINT "De EDIT-toestand van programma 1 altijd beeindigen met"
63085   PRINT " BREAKtoets/BREAKtoets !!!!   Na * starten met RUN."
63095   END
```



DAI WITH X-BUS CARD

CONVERSION PDL

— C O N V E R S I O N —

Le principal défaut de la routine PDL du DAI est le temps de conversion à vide. En effet quand le paddle n'est pas branché le DAI met un temps tout a fait fantaisiste pour lire la valeur présente sur le port. Vous trouverez ci dessous la routine du DAI revue et corrigé afin d'en éliminer ce défaut. Cette routine interessera tout les programmes faisant appel au paddle pour y eviter les résultats désagreables d'un freinage (Cf Acrobates...).

```
SPL V1.1  PAGE 1

0000                    PRT       1BH,43H,1H,1BH,45H,0DH

0000            ;
0000            ;
0000            ; Definition des differentes adresses du DAI :
0000            ;
0000            ;
0000  à=0040 POROM    EQU       40H             ;Copie de FD06
0000  à=FC00 PDLCH    EQU       0FC00H          ;Ad. compteur pdl
0000  à=FC06 SNDC     EQU       0FC06H          ;Ad. commande compteur
0000  à=FD01 PDLST    EQU       0FD01H          ;Ad. Lancer compteur
0000  à=FD06 PORO     EQU       0FD06H          ;Ad. Info. pdl
0000            ;
0000  à=0000 PDL      EQU       0H              ;N0 du paddle (0-5)
0000            ;
0000                    ORG       400H
0400            ;
0400            ;
0400            ; Autorisation fonctionnement compteur paddle
0400            ;  - PORO Bit 3 = 1
0400            ; Selection du N0 de paddle
0400            ;  - PORO Bit 0,1,2 = N0 de paddle
0400            ;
0400            ;
0400  F3      RPDL    DI                        ;Acces au E/S
0401  3A4000          LDA       POROM
0404  E6F8            ANI       0F8H            ;N0 pdl=0
0406  F608            ORI       PDL#8H          ;Autorise pdl+n0
0408  3206FD          STA       PORO            ;Actualise E/S
040B  324000          STA       POROM           ;Actualise copie E/S.
040E  FB              EI
```


CONVERSION PDL

```
040F            ;
040F            ;
040F            ; Initialisation du 8253 (Compteur programmable)
040F            ;  - SNDC Bit 0     = 0  : Compteur binaire 16 bits
040F            ;  -      Bit 1,2,3 = 0  : Arret du cptr sur impulsion pdl
040F            ;  -      Bit 4,5   = 1  : Charge deux octets
040F            ;  -      bit 6,7   = 0  : Compteur pdl (0) selectionne
040F            ;
040F            ;
040F  3E30            MVI A     30H             ;Init. 8253
0411  3206FC          STA       SNDC
0414            ;
0414            ; Chargement du compteur 0 avec 255
0414            ; Lance le compteur 0 en accedant a l'adresse PDLST
0414            ;
0414  21FF00          LXI H     0FFH            ;Charge compteur 0
0417  2200FC          SHLD      PDLCH
041A  3A01FD          LDA       PDLST           ;Lance compteur 0
041D            ;
041D            ; Boucle d'attente.
041D            ;
041D  212301          LXI H     123H            ;Attente conversion
0420  2B      WAIT    DCX H
0421  7D              MOV A,L
0422  B4              ORA H
0423  C22004          JNZ       WAIT
```

```
SPL V1.1  PAGE 2

0426            ; La valeur lue sur le port est maintenant disponible
0426            ; pour la connaitre reellement on effectue le calcul
0426            ;   Valeur effective = - (Valeur lue + 50)
0426            ;
0426  2A00FC          LHLD      PDLCH           ;Lecture conversion
0429  7D              MOV A,L
042A  2F              CMA                       ;Complemente
042B  DE31            SBI       31H             ;soustrait 49
042D  6F              MOV L,A
042E            ;
042E            ; La valeur convertie est maintenant disponible dans L
042E            ;
042E            ; On interdit maintenant l'acces au paddle
042E            ; PORO Bit 3 = 0
042E            ;
042E  F3              DI
042F  3A4000          LDA       POROM
0432  E6F7            ANI       0F7H
0434  3206FD          STA       PORO
0437  324000          STA       POROM
043A  FB              EI
043B            ;
043B                  END
```

CP/M terminal

Peter Jongen
Zeemanhof 25
2871JW Schoonhoven.

Dear Dai-namic friends,

As I alraedy mentioned in one of my earlier letters I am using the DAI as a (full grafic) terminal connected to my CPM system (Bigboard II).
Enclosed I send you a new copy of this program and a modified copy of the program's "facts and figures"

I have extracted the alternative inkey routine use in this program and assembled it as a seperate ML module, loadable in the DAI.
Set the start of heap to an address above the routine, read the tape and do >G 400.
The comments in the listing are selfexplanatory (I hope).

Furthermore as I prommissed at the HCC-days in Utrecht I send you coppies of my letters to Peter Noyez from Belguim who asked my help in developping a disk-contoller and DOS for his DAI.
As you can read I asked him to make an article for dainamic about his experiences. In his latest letter he wrote to do his best in finding time for it. Sinds I am not using my own 8-inch DAI-disk anny more and my d.o.s. was not yet completely ready for publication, (read: not yet idiot proof) I am not in a position anny more to write a full detailed article myself. If however Peter is not in a position to do a full article about his findings I wil try to write something.
I do not send you the listings etc of my dos and controler that accompanied my letters to Peter if somebody is interested let me know or wait for Peter article.

One week ago I have changed the DAI hardware to get the 80-column mode develloped by Anton Doornenbal. The change is realy easy to make for someone who has experience in this field.
Ofcause I adapted the change in my terminal program.
Now I have a terminal with 80X24 character screen, and all grafic features of the DAI!!.

And! I can still use the DAI as a DAI with disks. Both, in my CPM system and the DAI I have written programs that simmulate a DAI-disk-operating-system!. I can LOAD, SAVE, and do LOADA and SAVEA from 'disk'. The files are than loaded and stored on a diskette under CPM. Furthermore the programs supports: printing on the CPM list device from out the DAI, and viewing the disk directory.

Assembly listing of the alternative inkey routine.
The assembling and linking is done on the CPM system. The program is than down-line-loaded into the DAI's memory.

```
       MACRO-80 3.43   27-Jul-81       PAGE    1


                        ;
                        ; ALTERNATIVE INKEY ROUTINE. CALLED PER RST-1 BY THE
                        ; SCAN
                        ; KEYBOARD ROUTINE. NORMAL ROUTINE DID NOT ALLOW CTRL
                        ; CHARACTERS TO BE GENERATED.
                        ; - UPPER CASE LOCK/UNLOCK IS DONE WITH CTRL-SHIFT.
                        ; - ESCAPE CHAR IS CONTROL-[
                        ; SOME CHARACTERS PRESENT IN DAI-CHARACTER ROM AND
                        ; NOT ON KEYBOARD
                        ; ARE GENERATED USUNG CONTROL-KEY.
                        ; - CONTROL :  !  ,  7  ,  8  ,  9  ,  -  ,  /
                        ;   GENERATE :  !  ,  '  ,  {  ,  }  ,  _  ,  \
                        ;
                        ;
                        ;           Peter Jongen
                        ;           Zeemanhof 25
                        ;           2871jw Schoonhoven.     tel:  01823 5170
                        ;
                        ;
0080                    BIT7    EQU     080H
0064                    I1USA   EQU     064H
                        ;
                                ORG     0400H
                        ;
0400' E5                START:  PUSH    H
0401' F3                        DI
0402' 21 040B'                  LXI     H,RST1
0405' 22 0064                   SHLD    I1USA   ;SET NEW INTERUPT VECTOR
0408' E1                        POP     H
0409' FB                        EI
040A' C9                        RET
040B' E1                RST1:   POP     H       ;FROM STACK
040C' E3                        XTHL            ;RET ADR IN HL
040D' F5                        PUSH    PSW
040E' 7E                        MOV     A,M     ;DATA BYTE
040F' FE 12                     CPI     012H    ;IS IT INKEY?
0411' CA 041B'                  JZ      INKEY   ;YES
0414' F1                        POP     PSW
0415' E3                        XTHL
0416' E5                        PUSH    H
0417' 21 C70E                   LXI     H,0C70EH
041A' E9                        PCHL            ;CONTINUE NORMAL RST1
041B' F1                INKEY:  POP     PSW
041C' 23                        INX     H       ;ADDRESS AFTER RST-DATA
041D' E3                        XTHL            ;TO STACK
041E' F5                        PUSH    PSW
041F' C5                        PUSH    B
```

```
0420'   E5
0421'   3A 0040
0424'   F5
0425'   F6 C0
0427'   32 0040
042A'   32 FD06
042D'   FB
042E'   3E 07
0430'   90
0431'   B7
0432'   87
0433'   87
0434'   87
0435'   81
0436'   4F
0439'   2A 02A7
043B'   06 00
043D'   FE 11
043F'   DA 0449'
0440'   FE 2B
0442'   D2 0449'
0445'   3A 02C3
0448'   47
0449'   3A 02B0
044C'   A8
044D'   E6 40
044F'   CA 0458'
0452'   D5
0453'   11 0038
0456'   19
0457'   D1
0458'   06 00
045A'   09
045B'   7E
045C'   B7
045D'   CA 04B6'
0460'   FE 80
0462'   CA 04B6'
0465'   FE 7E
0467'   C2 046C'
046A'   3E 40
046C'   47
046D'   3A 02AE
0470'   E6 40
0472'   CA 0491'
0475'   78
0476'   FE 40
0478'   D2 048D'
047B'   21 04CD'
047E'   4E
047F'   23
0480'   7E
0481'   B8
0482'   23
0483'   CA 04B2'
0486'   0D
```

```
        PUSH    H
        LDA     040H    ;OLD BANK SELECT
        PUSH    PSW     ;SAVE IT
        ORI     0C0H    ;SELECT BANK 3
        STA     040H
        STA     0FD06H  ;DO SELECTION
        EI
        MVI     A,07H   ;)
        SUB     B       ;)
        ADD     A       ;)CALCULATE OFFSET
        ADD     A       ;)FOR KEY PRESSED
        ADD     A       ;)STORE IN C
        ADD     C       ;)
        MOV     C,A     ;)
        LHLD    02A7H   ;ADRR OF ASCii TABEL
        MVI     B,0
        CPI     011H    ;)
        JC      L3E145  ;) CHECK IF
        CPI     02BH    ;) A-Z
        JNC     L3E145
        LDA     02C3H   ;SHIFT LOCK
        MOV     B,A     ;IN B
L3E145: LDA     02B0H   ;GET SHIFT BYTE
        XRA     B       ;TAKE CTRL INTO ACCOUNT
        ANI     040H
        JZ      L3E146  ;NO SHIFT
        PUSH    D
        LXI     D,038H  ;OFFSET L.C.
        DAD     D
        POP     D
L3E146: MVI     B,0
        DAD     B
        MOV     A,M     ;GET ASC FROM TABLE
        ORA     A       ;CHECK BREAK,REPT,SHIFT
        JZ      L3E149  ;CHECK FOR CTRL
        CPI     BIT7    ;CHECK CNTRL ONLY
        JZ      L3E149  ;UPDATE CNTRL FLAG
        CPI     07EH    ;~ CHAR
        JNZ     NOTRAN  ;DIRECT:  TILDE TO MASTERSPACE
        MVI     A,040H  ;MAKE @
NOTRAN: MOV     B,A     ;ASCII IN B
        LDA     02AEH
        ANI     040H    ;CTRL BIT
        JZ      L3E     ;NOT SET
        MOV     A,B
        CPI     040H    ;CHAR ALPHA A-z ?
        JNC     MKCNTR  ;YES, MAKE CONTROL CHAR
        LXI     H,XLTAB ;POINT TO TRANSLATE TABLE
        MOV     C,M     ;GET TABE COUNT
XLLOOP: INX     H       ;POINT TO CHAR. TO TRANSLATE
        MOV     A,M     ;GET IT
        CMP     B
        INX     H       ;POINT TO TRANSLATION
        JZ      CFOUND  ;FOUND IT, DO TRANSLATIONA
        DCR     C
```



```
0487'   C2 047F'
048A'   C3 0491'
048D'   78
048E'   E6 9F
0490'   47
0491'   2A 02BE
0494'   E5
0495'   CD D69C
0498'   3A 02C0
049B'   BD
049C'   CA 04A4'
049F'   22 02BE
04A2'   E3
04A3'   70
04A4'   E1
04A5'   F3
04A6'   F1
04A7'   32 0040
04AA'   32 FD06
04AD'   FB
04AE'   E1
04AF'   C1
04B0'   F1
04B1'   C9

04B2'   46
04B3'   C3 0491'


04B6'   3A 02AE
04B9'   4F
04BA'   3A 02B0
04BD'   A1
04BE'   E6 40
04C0'   CA 04A5'
04C3'   3A 02C3
04C6'   2F
04C7'   32 02C3
04CA'   C3 04A5'


04CD'   06
04CE'   38
04CF'   7B
04D0'   39
04D1'   7D
04D2'   2F
04D3'   5C
04D4'   2D
04D5'   5F
04D6'   37
04D7'   60
04D8'   21
04D9'   7C
```

```
        JNZ     XLLOOP  ;TRANSLATE LOOP
        JMP     L3E     ;NO TRANSLATION, END OF TABLE
MKCNTR: MOV     A,B
        ANI     9FH     ;MAKE CONTOL BUT LEAVE
        MOV     B,A     ;BIT 7 SET
L3E:    LHLD    02BEH   ;NEXT POS IN KLIND
        PUSH    H
        CALL    0D69CH  ;UPDATE POINTER
        LDA     02C0H   ;GET L.S. BYTE
        CMP     L
        JZ      L3E147
        SHLD    02BEH   ;UPDATE KLIND
        XTHL    ;GET OLD KLIND
        MOV     M,B     ;STORE ASC
L3E147: POP     H
L3E148: DI
        POP     PSW     ;OLD BANK-SELECT
        STA     040H    ;RESTORE
        STA     0FD06H  ;RESET
        EI
        POP     H
        POP     B
        POP     PSW
        RET             ;RETURN FROM RST1
;
CFOUND: MOV     B,M     ;GET NEW CHAR,
        JMP     L3E
;
;* CNTRL FLAG
L3E149: LDA     02AEH   ;CTRL
        MOV     C,A
        LDA     02B0H   ;SHIFT
        ANA     C
        ANI     040H    ;BOTH BITS SET
        JZ      L3E148  ;NO
        LDA     02C3H   ;YES
        CMA
        STA     02C3H
        JMP     L3E148
;
; TRANSLATE TABLE FOR CHARACTERS NOT ON DAI-KEYBOARD
;
XLTAB:  DB      (XLTABE-XLTAB)/2
        DB      '8'
        DB      123
        DB      '9'
        DB      125
        DB      '/'
        DB      92
        DB      '-'
        DB      95
        DB      '7'
        DB      96
        DB      '!'
        DB      124
```

```
04DA'                           XLTABE:
                                        END


Macros:

Symbols:
0080    BIT7            04B2'   CFOUND          0064    I1USA
041B'   INKEY           0491'   L3E             0449'   L3E145
0458'   L3E146          04A4'   L3E147          04A5'   L3E148
04B6'   L3E149          048D'   MKCNTR          046C'   NOTRAN
040B'   RST1            0400'   START           047F'   XLLOOP
04CD'   XLTAB           04DA'   XLTABE
```

8080 to Z80 EQUIVALENCE

8080	Z80	8080	Z80	8080	Z80
ACI [B2]	ADC A, n	IN [B2]	IN A, (n)	POP H	POP HL
ADC M	ADC A, (HL)	INR M	INC (HL)	POP PSW	POP AF
ADC r	ADC A, r	INR r	INC r	PUSH B	PUSH BC
ADD M	ADD A, (HL)	INX B	INC BC	PUSH D	PUSH DE
ADD r	ADD A, r	INX D	INC DE	PUSH H	PUSH HL
ADI [B2]	ADD A, n	INX H	INC HL	PUSH PSW	PUSH AF
ANA M	AND (HL)	INX SP	INC SP	RAL	RLA
ANA r	AND r	JC [B2] [B3]	JP C, nn	RAR	RRA
ANI [B2]	AND n	JM [B2] [B3]	JP M, nn	RC	RET C
CALL	CALL nn	JMP [B2] [B3]	JP nn	RET	RET
CC [B2] [B3]	CALL C, nn	JNC [B2] [B3]	JP NC, nn	RLC	RLCA
CM [B2] [B3]	CALL M, nn	JNZ [B2] [B3]	JP NZ, nn	RM	RET M
CMA	CPL	JP [B2] [B3]	JP P, nn	RNC	RET NC
CMC	CCF	JPE [B2] [B3]	JP PE, nn	RNZ	RET NZ
CMP M	CP (HL)	JPO [B2] [B3]	JP PO, nn	RP	RET P
CMP r	CP r	JZ [B2] [B3]	JP Z, nn	RPE	RET PE
CNC [B2] [B3]	CALL NC, nn	LDA [B2] [B3]	LD A, (nn)	RPO	RET PO
CNZ [B2] [B3]	CALL NZ, nn	LDAX B	LD A, (BC)	RRC	RRCA
CP [B2] [B3]	CALL P, nn	LDAX D	LD A, (DE)	RST	RST P
CPE [B2] [B3]	CALL PE, nn	LH LD [B2] [B3]	LD HL, (nn)	RZ	RET Z
CPI [B2]	CP n	LXI B [B2] [B3]	LD BC, nn	SBB M	SBC A, (HL)
CPO [B2] [B3]	CALL PO, nn	LDID [B2] [B3]	LD DE, nn	SBB r	SBC A, r
CZ [B2] [B3]	CALL Z, nn	LXI H [B2] [B3]	LD HL, nn	SBI [B2]	SBC A, n
DAA	DAA	LXI SP [B2] [B3]	LD SP, nn	SHLD [B2] [B3]	LD (nn), HL
DAD B	ADD HL, BC	MOV M, r	LD (HL), r	SPHL	LD SP, HL
DAD D	ADD HL DE	MOV r, M	LD r, (HL)	STA [B2] [B3]	LD (nn), A
DAD H	ADD HL, HL	MOV r1, r2	LD r, r'	STAX B	LD (BC), A
DAD SP	ADD HL, SP	MVI M	LD (HL), n	STAX D	LD (DE), A
DCR M	DEC (HL)	MVI r [B2]	LD r, n	STC	SCF
DCR r	DEC r	NOP	NOP	SUB M	SUB (HL)
DCX B	DEC BC	ORA M	OR (HL)	SUB r	SUB r
DCX D	DEC DE	ORA r	OR r	SUI [B2]	SUB n
DCX H	DEC HL	ORI [B2]	OR n	XCHG	EX DE, HL
DCX SP	DEC SP	OUT [B2]	OUT (n), A	XRA M	XOR (HL)
DI	DI	PCHL	JP (HL)	XRA r	XOR r
EI	EI	POP B	POP BC	XRI [B2]	XOR n
HALT	HLT	POP D	POP DE	XTHL	EX (SP), HL

Facts and figures about the DAI-terminal program.

How to load the program:

Set the start of heap pointer to :D00
The program starts at :400
The tape reads into :400-BC1, address BC1-D00 are initalised by the program.

GENERAL:

The program works at 9600 baud, but that can be changed before starting the program.

Interupt is used for receiving the characters from the host system. These characters are then stored into a buffer. As a buffer the program uses all available RAM above End-of-symbol table onto Screen bottom. The main loop then takes the characters from the buffer and puts them on the screen. Especialy when the DAI has to scroll up the screen it is not fast enough to keep up with the 9600 baud line speed, but because of the buffering no characters are lost.

If very large and long (several pages +/- 40k!!) messages are send to the terminal at full speed, the posibility exists that the buffer will overoll. If that is expected a small hardware change will allow an inactive "CLear-to-send" sinal to be generated at pin 5 of the RS232 interface. The DAITERM program support this. As soon as enough buffer space is available the CTS signal will turn active again etc. By this methode no characters are lost no matter what happens. Ofcause the host system has to be able to see this CTS signal.

THE KEYBOARD:

The DAITERM uses a slightly modified keyboard routine. The contol key will now operate as it should. Pressing it alone will do nothing, but together with any other key this will generate the correct control-code. The shift key works as normal however to shift/loc from upper to lower case and back one needs to press control-shift. Furthermore the ' ' character generates a '@'.

Some extra characters are generated using the contol-key with some none alpha keys.

contol	:	!	7	8	9	-	/
generate	:	!	'	{	}	_	\

The cursor control keys do not generate any character to the host system but are used for terminal control.(see below)

THE SCREEN:

The screen format is 24 lines by 80! characters. Sinds the DAI hardware only allows 60 chars to be displayed, the DAITERM program allows horizontal scrolling. By pressing the left cursor control key (gray keys) the screen will scroll 10 character positions to the left. Doing this twice will schow char. pos 80 on the right side of the screen. etc etc for the right arrow key.
If the cursor moves to, or is at, a character position not currently displayed on the screen, it will stay at the left or right side of the screen and change into an arrow pointing in the direction of the currently not displayed part.
Auto cr/lf is generated if more than 80 characters are send to a line.

TERMINAL CONTROL:

To stop the DAITERM program; fist press break and then press the shift and up-arrow key. The program will restore all interupt, pointers etc and terminate.
If the terminal is in grafic mode and you want to return to character mode press Break and then the down-arrow key. This also clears the screen when in normal character mode.

PROGRAMMING THE TERMINAL:

Appart from being able to display received text the program exepts several control sequences to control cursor positioning, mode control, grafics etc.
Here is the full list:

>	cntr-L = 012h	
or	cntr-Z = 01Ah	
or	esc *	
or	esc :	Cursor home and erase screen
>	esc = l c	Cursor positioning, l=line number biased 020h c=column number biased 020h to position the cursor at top left of screen send: 01B3D2020 (all hex)
>	esc T	
or	cntr-X	Erase to end of line. Leave cursor at current position and clear rest of line.
>	esc Y	
or	cntr-Q	Erase to end of display. Leave cursor at current position ans erase rest of screen after and below cursor.
>	esc M x	Dai grafic controls, see below.

>	esc m	Load dai memory from host,see below.
>	esc c	Call routine in Dai memory,see below.

USING DAI-GRAFICS:

All grafis are controled with the "esc,M,x...." sequence.

x stands for:	# of parameters
s - set screen mode	1
g - set grafic colours (colorg)	4
t - set text colours (colort)	4
l - draw a line	5
d - set a dot	3
f - fill a block	5

attn. The case is important so "esc M G..." will do nothing and sound the terminal allarm shortly (1000Hz at sound channel 1).

After the above sequence up to 5 parameters must to be specified. Parameters are given with normal !integer! ascii numbers separates by one or more spaces.
Best I explain this with an example using basic:

```
10 E$=CHR$(27)
20 M%=10
30 C1%=0:C2%=5:C3%=10:C4%=13
40 X1%=0:Y1%=0:X2%=100:Y2%=120
50 PRINT E$;"Mg";C1%;C2%;C3%;C4%
60 PRINT E$;"Ms";M%            :REM SET SCREEN MODE 6
70 FOR N=1 TO 1000:NEXT        :REM WAIT LOOP EXPLAINED BELOW
80 PRINT E$;"Ml";C3%;X1%;Y1%;X2%;Y2%
```

Above program will set COLORG initiate the screen in MODE 6 and DRAW one line 0,0 100,120 in colour 5

Screen mode parameter:

MODE	PARAMETER	MODE	PARAMETER
0	255	3A	5
1	0	4	6
1A	1	4A	7
2	2	5	8
2A	3	5A	9
3	4	6	10
		6A	11

Note:!
The colour number is given as the first parameter, not as in the DAI as the last parameter!.

If the line, dot or fill parameters cause off screen, nothing happens, the line is not drawn.

Special care has to be taken if a mode set command is done. As already mentioned the program assigns all available memory as input buffer and screen-save (characters scrolled off the screen). If the screen mode changes, the buffer has to be rearranged. After a mode change the program looks if the available memory has been changed and if so, resets all input, output buffer pointers. By doing so all data send to the terminal after the mode change command is lost. During the time it takes the DAI to set up the screen memory no data should be send to the terminal.

DOWN-LINE-LOAD:

Esc,m,aa,cc ....

aa = 2 bytes address to store date in
cc = 2 bytes count value for the number of bytes
both numbers in 8080 hexadecimal format (l.s. byte first)

All data received after this sequence are stored into the DAI memory starting at the requested adress. Before you can use this, be sure to make enough room by changing the DAI-pointers.
I use this to down-line-load a new versions of the terminal program, that I assemble with the micro-soft assembler under CPM, or load the screen-copy program.

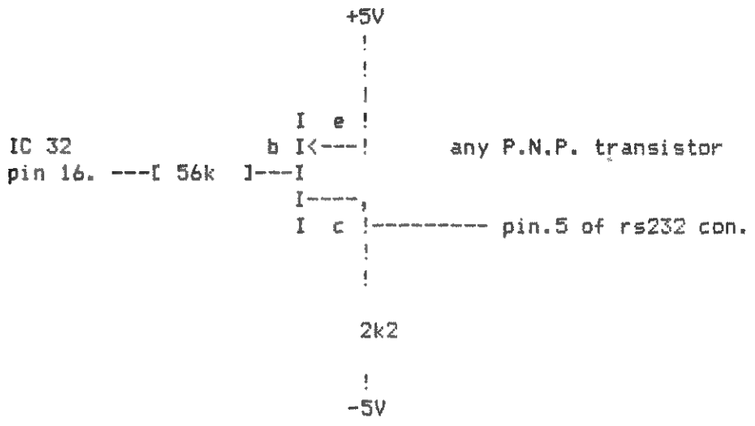
CALLING THE DAI:

The host system can load a program in the DAI memory and then start it by means of the following sequence:
esc,c,m,ACBEDLHaa
c and m are the lower case ascii c and m
A,C,B,E,D,L,H are the contents off all 8080 registers, they are loaded before calling the routine at adress "aa". Again "aa" is in 8080 format l.s. byte first.
After return from the called routine the DAITERM program checks if the memory pointers have changed and if so reassigns the input buffers.
With the two sequences from above the host system can e.g. load the START-OF-HEAP pointer, call :DE05, down-line-load the screen copy program and call it. In my case the screen copy program then sends the screen data to the host-system, where the data is stored in memory and printed or saved in a disk file.

CLEAR TO SEND:

As already mentioned, if the terminal is connected to a system that send pages of data or a lot of grafic commands the 'slow' DAI is not fast enough to allow 9600 baud so the input-buffer will overflow. The DAITERM program periodicaly chechs for this condition and if the buffer is nearly full it will set the CTS output inactive. As CTS I use the signal that controls the power on relay of the 2nd cassette interface. One transistor and 2 resistors are used to make the signal CCITT compatible (<-3V = inactive, > +3V = active).

CIRCUIT FOR CTS:



You could test the program by connecting 2 DAI pc with the rs232 connected:

ground	pin 1 -------	pin 1	ground
out	pin 2 -------	pin 3	in
in	pin 3 -------	pin 2	out
dtr	pin 4 -------	pin 5	clear to send

Peter Jongen
Zeemanhof 25
2871JW Schoonhoven tel: 01823 5170

BLEUNOSE

```
1     REM :
2     REM :     ***********************************
3     REM :     *                                 *
4     REM :     *     R.MARCEL/FEVRIER 1982       *
5     REM :     *                                 *
6     REM :     *              BLUENOSE           *
7     REM :     *                                 *
8     REM :     ***********************************
9     REM :
100   MODE 0:COLORT 5 13 5 5
110   PRINT CHR$(12):CURSOR 0,23:PRINT "SCHOONER                  'BLUENOSE'
'A NADA"
115   PRINT "de 43 m.                                            (1920)"
120   POKE #BEE2,#DF:POKE #BEE3,#5F:CURSOR 4,21:PRINT "GOELETTE"
130   MODE 6A:COLORG 8 14 10 0
140   REM : Au mouillage
150   FILL 75,0 246,10 0:GOSUB 3000:GOSUB 3000
160   DRAW 290,15 244,12 0:DRAW 290,14 244,11 0
170   DRAW 270,0 260,11 0
180   GOSUB 4000
190   DRAW 58,24 152,205 0
200   DRAW 153,200 208,120 0
210   DRAW 100,24 150,140 0:DRAW 164,21 206,120 0
220   DRAW 151,20 90,24 0:DRAW 151,21 90,25 0
230   DRAW 155,23 206,17 0:DRAW 155,24 206,18 0
240   DRAW 288,18 210,112 0:DRAW 254,12 210,112 0
250   WAIT TIME 200
300   REM :Appareillage
310   GOSUB 3100
320   DRAW 152,116 110,164 0:DRAW 152,117 110,165 0
330   GOSUB 3000:GOSUB 3000
340   GOSUB 3100
350   DRAW 206,92 166,122 0:DRAW 206,93 166,123 0
360   WAIT TIME 150:DRAW 270,0 260,11 8
370   GOSUB 3000:GOSUB 3000:GOSUB 3000:GOSUB 3100
380   GOSUB 4000
390   WAIT TIME 200
400   REM : Changement de couleurs
410   DIM CC(15.0),CB(15.0)
420   CC(1.0)=0.0:CC(2.0)=1.0:CC(3.0)=8.0:CC(4.0)=10.0:CC(5.0)=9.0:CC(6.0)=12.0:CC(7.0)=14.0
430   CB(1.0)=0.0:CB(2.0)=1.0:CB(3.0)=3.0:CB(4.0)=4.0:CB(5.0)=6.0:CB(6.0)=7.0:CB(7.0)=9.0
440   CB(8.0)=10.0:CB(9.0)=15.0:CB(10.0)=2.0
450   FOR N=1.0 TO 10.0
460   A%=RND(8.0):B%=RND(16.0):C%=RND(16.0):D%=RND(11.0)
470   IF B%=CC(A%) OR C%=CC(A%) OR B%=CB(D%) OR C%=CB(D%) THEN 460
480   IF CC(A%)=CB(D%) THEN 460
490   COLORG CC(A%) B% C% CB(D%):WAIT TIME 200:NEXT
500   COLORG 8 10 3 0
1000  GOTO 1000
3000  REM :S/P  FILL TRIANGLE
3010  READ X1,Y1,X2,Y2,X3,Y3,CD
3020  XJ=X1:YJ=Y1:XK=X2:YK=Y2:GOSUB 3500:A=R1:B=R2
3030  XJ=X2:YJ=Y2:XK=X3:YK=Y3:GOSUB 3500:C=R1:D=R2
3040  XJ=X3:YJ=Y3:XK=X1:YK=Y1:GOSUB 3500:E=R1:F=R2
3050  FOR Y=Y1 TO Y2:XD=(Y-B)/A:XF=(Y-F)/E:DRAW XD,Y XF,Y CD:NEXT
3060  FOR Y=Y2 TO Y3:XD=(Y-D)/C:XF=(Y-F)/E:DRAW XD,Y XF,Y CD:NEXT
3070  RETURN
3100  REM :S/P  FILL QUADRILATERE
3110  READ X1,Y1,X2,Y2,X3,Y3,X4,Y4,CD
3111  REM
3112  REM : Les Yi doivent etre dans l'ordre croissant
3113  REM
3120  XJ=X1:YJ=Y1:XK=X2:YK=Y2:GOSUB 3500:A=R1:B=R2
3125  IF SGN(X2-X1)=SGN(X4-X3) THEN 3145
3130  XJ=X2:YJ=Y2:XK=X3:YK=Y3:GOSUB 3500:C=R1:D=R2
3135  XJ=X3:YJ=Y3:XK=X4:YK=Y4:GOSUB 3500:E=R1:F=R2
3140  XJ=X4:YJ=Y4:XK=X1:YK=Y1:GOSUB 3500:G=R1:H=R2:GOTO 3160
3145  XJ=X2:YJ=Y2:XK=X4:YK=Y4:GOSUB 3500:C=R1:D=R2
3150  XJ=X4:YJ=Y4:XK=X3:YK=Y3:GOSUB 3500:E=R1:F=R2
3155  XJ=X3:YJ=Y3:XK=X1:YK=Y1:GOSUB 3500:G=R1:H=R2
3160  A1=A:B1=B:A2=G:B2=H:FOR Y=Y1 TO Y2:GOSUB 3300:NEXT
3170  A1=C:B1=D:FOR Y=Y2 TO Y3:GOSUB 3300:NEXT
3180  IF SGN(X2-X1)=SGN(X4-X3) THEN 3200
3190  A1=E:B1=F:FOR Y=Y3 TO Y4:GOSUB 3300:NEXT:RETURN
3200  A1=C:B1=D:A2=E:B2=F:FOR Y=Y3 TO Y4:GOSUB 3300:NEXT:RETURN
3300  XD=(Y-B1)/A1:XF=(Y-B2)/A2:DRAW XD,Y XF,Y CD:RETURN
3500  IF XJ=XK THEN XK=XK+0.1
3510  R1=(YJ-YK)/(XJ-XK):R2=((XJ*YK)-(YJ*XK))/(XJ-XK):RETURN
4000  REM S/P MATURE
4010  FILL 150,10 152,140 0:FILL 152,120 153,205 0
4020  FILL 206,10 208,120 0:FILL 208,104 209,170 0
4030  DRAW 209,165 290,16 0
4060  DRAW 153,205 209,170 0
4070  DRAW 208,14 155,20 0:DRAW 208,15 155,21 0
4080  DRAW 151,17 50,23 0:DRAW 151,18 50,24 0
4090  DRAW 65,10 70,23 0:DRAW 65,10 60,23 0:DRAW 160,10 160,20 0
4100  DRAW 150,140 135,9 0:DRAW 206,120 195,9 0
4110  FILL 85,9 130,12 0:DRAW 85,13 130,12 10:DRAW 60,10 260,11 10
4120  FILL 83,10 85,13 10:DOT 130,11 10
4130  FOR J=0.0 TO 13.0 STEP 13.0:DOT 90+J,11 8:NEXT
4200  RETURN
5000  DATA 246,0,  150,9,264,13,0
5010  DATA 75,0,104,10,60,9,0
5020  DATA 149,19,55,25,149,118,112,159,14
5030  DATA 252,16,210,20,216,96,14
5040  DATA 288,18,234,30,212,109,10
5050  DATA 205,16,157,22,205,91,168,120,14
5060  DATA 204,97,172,122,208,167 ,14
5070  DATA 149,123,117,163,152,201,14
5080  DATA 265,62,218,88,212,158,14
5090  DATA 155,103,206,110,205,128,162,188,10
```

PROGRAMMEERTECHNIEKEN

De vorige keer ben ik begonnen met de functies van het handboek in groepen onder te verdelen. In de indeling die ik toen maakte kleefden nog wel enige 'slordigheidjes'. Niet dat er nu onzin stond, maar het een en ander behoeft toch nog wel enige toelichting.

Zoals reeds de vorige maal betoogd werd, beschouwen we een functie hier als een of ander voorschrift.
Dit voorschrift koppelt dan aan elk origineel (of argument) dat aan de functie kan meegegeven worden een zogenaamd beeld.
De mogelijke originelen zitten met zijn allen in een verzameling en die verzameling geven we aan met het woord 'Domein'. Ook de verzameling van alle beelden bij elkaar geven we met een eigen naam aan; we noemen die verzameling het 'Bereik'.

Indeling

Er bleken echter functies te bestaan, die toch geen argument meekregen en dus leek het er op dat er geen origineel aanwezig was. De vorige keer ben ik hier uitvoerig op ingegaan en ik wilde dan nu ook een andere groep wat nader bekijken. De grootste groep nog onbesproken functies (=langste kolom) wilde ik tot het laatst bewaren en onze aandacht nu eerst richten op de groepen waar tekstvariabelen een rol spelen.

In de tabel stonden zij als volgt :

Functies met tekstvariabele als argument en getalsvariabele als beeld	Functies met getalsvariabele als argument en tekstvariabele als beeld
ASC	CHR$
LEN	HEX$
VAL	LEFT$
VARPTR	MID$
	RIGHT$
	STR$
	SPC
	TAB

Velen zullen (hopelijk) bij deze indeling al hun wenkbrouwen hebben gefronst. In de linker kolom is VARPTR duidelijk een buitenbeentje en in de rechter kolom is de TAB en eventueel SPC een functie, die enigzins afwijkt.
Ik wilde deze kolommen niet afzonderlijk behandelen, daar er in de linkerkolom nogal wat functies bij zitten, waarvan in de rechterkolom de inverse functie staat.

Inverse functie

Bij elke functie moet gelden dat er bij elk origineel een en ook precies een beeld is. De argumenten die we aan een functie kunnen meegeven waar geen beeld bijhoort zitten dus niet in het domein en mogen dan ook niet als argument meegegeven worden.
Het hoeft echter niet zo te zijn dat bij elk origineel een ander beeld hoort. Het is best mogelijk dat er meerdere originelen zijn die allen hetzelfde beeld krijgen toegevoegd.
Is de functie echter zo, dat - of door het voorschrift - of door het gekozen domein (of beide) er bij elk origineel precies een beeld hoort, dat aan geen enkel ander origineel wordt toegevoegd spreken we van een een bijectie (uitspreken als bie-jek-sie !) of een-een afbeelding.
In dit geval hoort niet alleen bij elk origineel een beeld (dit moet zo zijn bij elke functie), maar bij elk beeld hoort een origineel en niet meerdere.
Het is dus mogelijk in dit geval een functie te definieren waarbij het origineel en het beeld van plaats zijn verwisseld.
Deze functie wordt de inverse functie van onze oorspronkelijke functie genoemd.

Simpele voorbeelden van functies die een inverse hebben zijn:

1 - Aan elk getal wordt het getal toegevoegd dat twee groter is.
De inverse functie is dan vanzelfsprekend de functie, die aan elk getal het getal toevoegt dat twee kleiner is.

2 - Aan elk getal wordt het getal toegevoegd dat drie maal zo groot is als het origineel.
De inverse functie is nu de functie, die aan elk getal het getal toevoegt dat drie maal zo klein is.

3 - Aan elke ASCII-code wordt een karakter toegevoegd.
De inverse functie voegt dan aan elk karakter een ASCII-code toe.

N.B.: Bij onjuiste interpretering van het begrip ASCII-code is dit onjuist !

Theorie en praktijk

Bij een nadere beschouwing blijkt er echter vrijwel geen functie te zijn op de DAI waarvan ook de inverse aanwezig is.

Ik zal dit illustreren aan de hand van het eerste voorbeeld. Het zal iedereen duidelijk zijn dat de functie zoals die hier gedefinieerd is een inverse heeft. Maar op de DAI (en elke andere computer trouwens ook) is dat niet het geval.
Ik beschouw twee gevallen apart :

a) We rekenen in gehele getallen (zgn. integers). Het is dan denkbaar een zo groot getal als argument mee te geven dat het beeld voor de DAI te groot wordt. De functie (en de bijbehorende inverse) bestaan dan alleen als het domein beperkt wordt.

b) We rekenen in getallen, die in de floating point notatie zijn genoteerd. We zien nu als we het origineel maar groot genoeg nemen dat er geen verschil meer zal zijn tussen de beelden die twee verschillende originelen geven. Er is in dit geval dus geeneens meer sprake van een functie laat staan een inverse.

Het voorafgaande komt omdat een computer nu eenmaal discreet werkt. Het is daarom ook belangrijk er bij bepaalde problemen aan te denken dat - alhoewel onze theorie goed is - het best eens zo zou kunnen zijn, dat de fout die we vaststellen door dat discrete werken wordt veroorzaakt.
In de 'normale' praktijk valt het gelukkig nogal mee.

Een test

Laat uw DAI maar eens een willekeurig getal door twaalf delen. Het resultaat weer door twaalf delen en zo door tot we in totaal honderd maal door twaalf gedeeld hebben.
Vervolgens gaan we dat resultaat met twaalf vermenigvuldigen enzovoorts tot we honderd maal met twaalf vermenigvuldigd hebben.
Iedereen ziet dat het uiteindelijke resultaat gelijk zou moeten zijn aan het oorspronkelijke uitgangsgetal.
Maar niet de DAI want die zal 'braaf' nul als resultaat oplepelen.

```
10      INPUT WAARDE
20      FOR I=1 TO 100
30      WAARDE=WAARDE/12
40      NEXT
50      FOR I=1 TO 100
60      WAARDE=WAARDE*12
70      NEXT
80      PRINT WAARDE
90      GOTO 10
```

Heeft U de beschikking over een andere computer die een grotere nauwkeurigheid heeft dan de DAI zodat het bovenstaande programma bij 'normale' waarden toch terugkomt bij de opgegeven waarde vervangen we honderd door een groter getal (bv duizend) en dan gaat het toch weer fout.

De Inversen

Welke zijn nu de functie op de DAI die een inverse hebben ? Uit de getoonde kolommen zijn dat de volgende paren :

ASC - CHR$
VAL - STR$

en eventueel kunnen we nog denken aan

\# - HEX$

ASC en CHR$

ASC is een afkorting van ASCII dat op zijn beurt weer een afkorting is voor American Standard Code for Information Interchange. Een standaard code dus om informatie uit te kunnen wisselen, die in Amerika is afgesproken. De afspraak betreft in tegenstelling tot wat velen menen niet de 256 mogelijkheden om de vulling van een byte als karakter weer

te geven. Op zijn hoogst kunnen we zeggen dat de 128 mogelijkheden die de eerste zeven bits bieden redelijk gestandaardiseerd zijn.
Zijn we echter streng aan het controleren dan blijkt in de praktijk dat we alleen de codes van de hoofdletters, de kleine letters, de cijfers en de meest voorkomende leestekens als gestandaardiseerd kunnen beschouwen.
Zelfs vrij algemene codes als 8 voor backspace en 12 voor schoon beeld blijken niet altijd te kloppen.
Zoals bekend wordt bij de DAI de set karakters van de eerste 128 mogelijkheden vrijwel identiek herhaald in de tweede 128 mogelijkheden.

Als origineel voor de functie ASC kunnen we elke gewenste tekstvariabele meegeven. Het beeld zal echter altijd slechts de ASCII-waarde van het eerste karakter van die tekstvariabele zijn.
Hieruit kunnen we al direct zien dat hier dus duidelijk geen sprake kan zijn van een inverse daar de functie geen bijectie is. Er is zelfs nog iets wat in tegenspraak met met de een-een gedachte : Als de DAI gevraagd wordt naar ASC(A$) en we krijgen als resultaat 0 weten we niet eens zeker dat de tekstvariabele A$ met een karakter met code nul begint. We krijgen namelijk hetzelfde resultaat als A$ leeg is !

CHR$ komt van CHaracteR string.
Het dollarteken wordt uitgesproken als string en geeft aan dat het resultaat een tekstvariabele is. In dit geval bestaat deze tekstvariabele uit maar een karakter.
Het domein van deze functie is de verzameling getallen 0 tot en met 255.

VAL en STR$

Ook bij VAL (van VALue=waarde) en STR$ (van STRing) lijkt de simpele uitleg erop, dat we hier met twee inversen te maken hebben.
De VAL heeft als argument een tekstvariabele die een getal voorstelt. Het beeld is de getalswaarde van het in tekst opgegeven getal.

voorbeelden :

```
PRINT VAL("4.6")              geeft  4.6
A$="3.2":PRINT VAL(A$)          ,,   3.2
B$="1.3":C=VAL(B$):PRINT C      ,,   1.3
```

De STR$ heeft als argument een getalswaarde en als beeld een tekstvariabele die in karakters het getal voorstelt.

voorbeelden :

```
PRINT STR$(2.3)              geeft ' 2.3'
A=5.6:PRINT STR$(A)            ,,  ' 5.6'
```

N.B. De aanhalingstekens (quotes) zijn geplaatst om de spatie voor het getal zichtbaar te maken.

We krijgen, nu we dus geen moeilijkheden hebben gezien, het idee dat VAL en STR$ elkaars inverse zijn. Maar niets is minder waar.
De eerste problemen verschijnen als we met STR$ een integer getal opgeven als argument. De STR$ blijkt dan als beeld niet de getalswaarde te geven maar altijd de floating point vorm van deze waarde. Overigens mogen de tekstvariabelen best met + beginnen of een zeker aantal spaties aan het begin hebben.

Maar dan de VAL. Ook hier blijkt het resultaat opgegeven te worden in floating point notatie. Maar ernstiger is het feit dat getalsteksten die met een - of een spatie begint niet wordt geaccepteerd.
De foutmelding die we bij toch pogen krijgen is zelf fout: 'INVALID NUMBER' het had bv 'INVALID TEXT' moeten zijn.

Een laatste fout uit de ROM's die ik moet melden is het gebruik van + en ; Tik maar eens PRINT STR$(6)+STR$(8) in Je zou nu verwachten ' 6.0 8.0' te zien te krijgen. Maar nee, de DAI verrast ons met ' 8.0 8.0'. Gebruiken we ; in plaats van + is het resultaat wel zoals verwacht.

en HEX$

Dit zijn nu wel echte inverse functies maar bij DAI heet # geen functie. Het # - teken (uitspraak nummer, hekje, hex of hexadecimaal; mijn voorkeur gaat uit naar hex) geeft aan dat het er achter volgende getal opgegeven wordt in de hexadecimale of zestientallige notatie. Dit geldt alleen bij intikken, daar de interne notatie in integervorm is.
De HEX$ waar de haakjes wel nodig zijn is de inverse van # en accepteert dus getalwaarden als origineel en levert teksten af waarin de gewenste getallen in hexadecimale code staan.
Dezelfde fout die we net bij STR$ hadden geconstateerd ten opzichte van het gebruik van de + geldt ook voor HEX$.

SPC en TAB

Deze beide functies hebben eigenlijk een verkeerde naam. SPC$ en TAB$ zou beter zijn daar het beeld een tekstvariabele is.
SPC is een samentrekking van het woord SPaCes (=spaties) en TAB een afkorting van TABulator.

Doen we A$=SPC(23) dan is A$ een tekst geworden die uit 23 spaties bestaat.
Doen we A$=TAB(23) dan is A$ een tekst geworden die uit 23 spaties bestaat.
Er is zo te zien geen enkel verschil.
We kunnen het verschil echter zichtbaar maken door voor de toewijzing de opdracht CURSOR 20,10 te geven.
Let er op dat dat dan wel op dezelfde regel gebeurt.
Beide functies hebben als beeld een tekst bestaande uit een aantal spaties. Het aantal is bij SPC het aantal dat opgegeven wordt als origineel.
Bij TAB is het aantal spaties gelijk aan het verschil tussen het opgegeven aantal en de waarde die de horizontale positie van de cursor aangeeft.
In feite heeft de TAB dus niet een argument, maar twee argumenten. Een van die twee namelijk de cursorpositie wordt door de DAI gefourneerd.
Is deze laatste waarde groter dan de opgegeven TAB-waarde zal het resultaat de lege string zijn.

Grappig is nog te vermelden dat we P$ wel als 255 spaties kunnen definieren met SPC(255) maar nooit op het scherm kunnen afdrukken. Probeer maar !

LEN

LEN komt van LENgth.
We geven een tekstvariabele als origineel op en krijgen de lengte ervan uitgedrukt in een getal dat het aantal karakters vertegenwoordigt.

LEFT$ en RIGHT$

Streng wiskundig gezien staan deze functies ook niet in de juiste kolom. Het klopt dat ze een tekstvariabele als beeld hebben maar het origineel is niet alleen een getalsvariabele. Er is nog een tweede argument, net als bij TAB. Dit tweede argument is een tekstvariabele.
We kunnen dit als volgt interpreteren: Als we de tekst als vast gegeven beschouwen, kunnen we vanaf dat moment het getal als origineel nemen. De plaats in de rechter kolom klopt dan. Fixeren we echter het getal; kunnen we daarna de tekst gaan varieren.
Ze zouden dan in een aparte kolom moeten komen te staan.
De syntax is simpel: LEFT$(A$,I) resp. RIGHT$(A$,I) met A$ de tekstvariabele, waarvan de linker resp. de rechter I karakters als beeld toegevoegd moeten worden. De waarde van I is mimimaal 0 en maximaal het aantal karakters van de tekstvariabele die argument is.

MID$

Een nog vreemdere eend in de bijt is de MID$. Hier dienen namelijk drie argumenten bij te worden opgegeven.
Te weten de tekstvariabele waaruit het beeld gedestileerd moet worden, en een tweetal getalsvariabelen die aangeven hoeveel karakters er moeten worden overgeslagen en hoeveel karakters er in het beeld moeten zitten.
De syntax is dus MID$(A$,I,J) met I karakters overslaan en het beeld is J karakters lang. Voor I en J geldt hier dat ze samen maximaal het aantal karakters mogen zijn dat A$ lang is.

Een ezelsbruggetje om te onthouden dat I het aantal karakters overslaan is en J het aantal karakters , dat we in het beeld wensen te hebben.
Bij alle drie (LEFT$, RIGHT$ en MID$) is de tekst als eerste genoemd en de lengte van het beeld als laatste. De tussenstaande waarde I moet dus wel het aantal plaatsen overslaan zijn.

VARPTR

VARPTR is een samentrekking van de woorden VARiable PoinTeR.
Om echter nu alle zaken die er mee te maken hebben op te sommen rest mij te weinig plaats. Ik zal er de volgende keer mee beginnen, temeer daar het een mooie brug is naar de laatste kolom.

Frank H. Druijff

EPROM-PROGRAMMER

EBES
2400 MOL

Aanpassing en uitbreiding van de EPROM-programmer
v. G. Knoops in dainamic 13.

De hierin beschreven programmer heeft volgens mij twee onvolkomenheden nl :

1) Hij werkt niet volgens D C E - concept wat betekent dat hij niet te gebruiken is door DAI-gebruikers die met een floppy-disk werken.

2) Bij het inladen met de "load" en "Next load" instructie gaan de origines verloren. Dit geeft problemen bij grote programma's met veel subroutines die elk een eigen "origine" hebben.

Als oplossing voor het tweede probleem werd afgestapt van de assembler - subroutina "Load" en "Next load" en vervangen door het basic-programma "Loader-dainamic", waaraan als supplement een verschuiving werd toegevoegd.

Deze is interessant als de EPROM niet op adres #Ø Ø Ø Ø begint.

Om aan het eerste probleem te voldoen, werd aan het schema een "8255" en een"adres-decoder" toegevoegd. (zie bijgevoegd schema)

Dit bracht ook enkele kleine wijzigingen mee in zowel de basic - als - assembler-programma's.


```
10    REM EPROM-PROGRAMMER
11    REM /////////////////
12    REM
14    REM
16    REM NAAM V.D. SCHIJF : EPROM
18    REM AUTEUR: L. VANTOURNHOUT / G.CALUWAERTS
20    REM VERSIE 24 JULI 1984
40    CLEAR &4000:COLORT 0 15 0 0
45    DIM T@(5,0),MEMRO/0(5,0):FOR XO/0=1 TO 5:READ T@(XO/0),MEMRO/0(XO/0):NEXT
46    IF PEEK(&6000)=&F5 THEN 60
47    POKE &131,3:PRINT "LOAD OEPROM.BIN":POKE &131,1
60    POKE &B1F4,1:POKE &B1FD,&A8:POKE &B1FA,0:POKE &B1FB,&AO
80    WAIT TIME 50:OUT &E3,&80:OUT &E2,0:OUT &E0,0:OUT &E1,0
90    AO/0=GETC:WAIT TIME 5
100   REM
102   REM PROGRAMMA-MENU
103   REM --------------
104   PRINT CHR@(12):TYPE@=T@(PEEK(&B1F4))
105   CURSOR 30,20:PRINT "TYPE:":TYPE@
108   CURSOR 15,18:PRINT "C   CHECK"
109   PRINT "                L   LOAD"
111   PRINT "                P   PROGRAM"
112   PRINT "                R   READ"
113   PRINT "                T   TYPE EPROM"
114   PRINT "                U   UTILITY"
115   PRINT "                V   VERIFY"
120   PRINT :INPUT "    WAT WENST U ":A@
140   IF A@="C" GOTO 200
142   IF A@="L" GOTO 20000
146   IF A@="P" GOTO 500
148   IF A@="R" GOTO 600
150   IF A@="T" GOTO 700
152   IF A@="U" GOTO 800
154   IF A@="V" GOTO 900
170   GOTO 100
200   REM
201   REM CHECK IF THE EPROM IS ERASED
203   REM ----------------------------
210   PRINT CHR@(12):CURSOR 15,20:PRINT "CHECK"
220   GOSUB 1000
230   CALLM &B000
240   AO/0=PEEK(&B1FF):IF AO/0=0 THEN PRINT CHR@(12):PRINT "OK":GOTO 105
250   PRINT CHR@(12):PRINT "NOT ERASED":GOTO 105
500   REM
501   REM PROGRAMMING THE EPROM
505   REM ---------------------
510   PRINT CHR@(12):CURSOR 15,15:PRINT "PROGRAM":CURSOR 15,13:PRINT "BUSY    "
520   GOSUB 1050
530   CALLM &B0F4
540   GOTO 90
600   REM
601   REM READING THE EPROM
602   REM -----------------
610   PRINT CHR@(12):CURSOR 15,15:PRINT "READ"
620   GOSUB 1000
630   CALLM &B15B
640   GOTO 100
700   REM
701   REM TYPE EPROM
```

```
702     REM ----------
710     PRINT CHR@(12):CURSOR 15,15:PRINT "TYPE"
715     FOR X0/0=1 TO 5:CURSOR 20,13-X0/0:PRINT T@(X0/0):NEXT
720     CURSOR 15,5:INPUT "KIES EEN TYPE":IN@:PRINT
730     Y0/0=0:FOR X0/0=1 TO 5
735     IF IN@=T@(X0/0) THEN Y0/0=X0/0
740     NEXT
750     IF Y0/0=0 THEN PRINT CHR@(12):PRINT "      TYPE ONBEKEND":GOTO 108
760     POKE #81F4,Y0/0:POKE #81FD,(MEMRO/0(Y0/0) IOR #A000)/255.0
761     REM INSTELLEN PARAMETERS VOOR GEWENSTE TYPE
770     PRINT CHR@(12):GOTO 80
800     REM
801     REM TERUG GAAN NAAR UTILITY
802     REM ----------------------
810     CALLM #8105
900     REM
901     REM KONTROLE OF DE EPROM JUIST IS INGELEZEN
902     REM ---------------------------------
910     PRINT CHR@(12):CURSOR 15,15:PRINT "VERIFY"
920     GOSUB 1000
930     CALLM #8190
940     A0/0=PEEK(#81FF):IF A0/0=0 THEN PRINT CHR@(12):PRINT "OK":GOTO 105
950     PRINT CHR@(12):PRINT "BAD":GOTO 105
1000    REM
1001    REM INITIATE 8255 (GIC) FOR CHECK,READ,VERIFY
1002    REM ---------------------------------
1003    REM GICB=GIC CONTROL BYTE; ECB=EPROM CONTROL BYTE
1004    REM RB= RELAIS BYTE
1010    GICB0/0=#90
1020    IF TYPE@="2732" THEN ECB0/0=#80:RB0/0=#B0:GOTO 1100
1030    ECB0/0=#20:RB0/0=#38:GOTO 1100
1050    REM INITIATE 8255 FOR PROGRAMMING
1060    GICB0/0=#80
1070    IF TYPE@="2732" THEN ECB0/0=#C0:RB0/0=#D0:GOTO 1100
1080    IF TYPE@="2532" THEN ECB0/0=#40:RB0/0=#50:GOTO 1100
1090    ECB0/0=#58:RB0/0=#50
1100    OUT #E3,GICB0/0:OUT #E2,RB0/0:POKE #81FE,ECB0/0:WAIT TIME 25:RETURN
1600    DATA 2716,2048,2516,2048,2732,4096,2532,4096,2758,1024
10001   REM TYPE, MEMORY RANGE
20000   REM
20001   REM LOADER
20002   REM ------
20010   PRINT CHR@(12):DPFLO/0=1:RECO/0=0
20020   PRINT "BEGINADRES EPROM (HEX): #":INPUT B0/0
20144   FADRO/0=#3002:LIMLO/0=#9FFF:LIMHO/0=#AFFF
20199   PRINT CHR@(12)
20200   FLEO/0=FADRO/0+(PEEK(FADRO/0-2) SHL 8.0)+PEEK(FADRO/0-1):REM FILE LENGTE
20219   REM
20220   REM INTER RECORD ROUTINE
20221   REM --------------------
20230   GOSUB 21000:IF BYTO/0.LT..GT.0 GOTO 20950
20240   GOSUB 21000:IF BYTO/0.LT..GT.0 GOTO 20950
20250   GOSUB 21000:IF FADRO/0.GT.=FLEO/0 THEN PRINT CHR@(12):GOTO 20960
20260   IF BYTO/0.LT..GT.#FF THEN PRINT CHR@(12):GOTO 20960
20281   REM
20290   REM LEES ORIGIN EN SOORT RECORD
20291   REM --------------------------
20300   SUMO/0=#FF:RECO/0=RECO/0+1.0
20310   PRINT :PRINT "RECORD ":RECO/0;
20320   GOSUB 21000:ORGO/0=BYTO/0
20330   GOSUB 21000:ORGO/0=ORGO/0+(BYTO/0 SHL 8)
```



```
20340   GOSUB 21000:IF BYTO/0.LT..GT.0.0 GOTO 20600:REM NAAR RES-TYP
20349   REM
20350   REM PROGRAM TYP
20351   REM ------------
20360   PRINT " PROGRAM TYPE"
20370   REM PADRO/0=RADRO/0
20400   GOSUB 21000:LNGO/0=BYTO/0-1
20410   FOR RADRO/0=ORGO/0+#A000-B0/0 TO ORGO/0+LNGO/0+#A000-B0/0
20420   GOSUB 21000:GOSUB 22000
20430   IF CURX.GT.0 GOTO 20450
20440   PRINT HEX@(RADRO/0):"  ":
20450   PR@=" ":IF BYTO/0.LT.=#F THEN PR@=PR@+"0"
20460   PR@=PR@+HEX@(BYTO/0):PRINT PR@:
20470   IF RADRO/0 IAND #F=#F THEN PRINT
20480   NEXT
20490   GOTO 20800
20584   REM
20590   REM RESERVE TYP
20595   REM -----------
20600   GOSUB 21000:FLSO/0=BYTO/0
20610   GOSUB 21000:CHARO/0=BYTO/0
20620   GOSUB 21000:AREAO/0=BYTO/0
20630   GOSUB 21000:AREAO/0=AREAO/0+(BYTO/0 SHL 8.0)-1
20635   PRINT " RESERVE TYPE"
20640   IF FLSO/0.LT..GT.0.0 GOTO 20700
20650   RADRO/0=ORGO/0+AREAO/0
20670   GOTO 20740
20700   FOR RADRO/0=ORGO/0+A000-B0/0 TO ORGO/0+AREAO/0+A000-B0/0
20710   BYTO/0=CHARO/0:GOSUB 22000
20720   NEXT
20740   PRINT "AREA ":HEX@(ORGO/0):" ":HEX@(ORGO/0+AREAO/0):
20750   IF FLGO/0=0 THEN PRINT " SKIPPED OVER":GOTO 20800
20760   PRINT " FILLED WITH ":
20770   IF CHARO/0.LT.=#F THEN PRINT "0":
20780   PRINT HEX@(CHARO/0):
20790   REM KONTROLE CHECKSUM
20791   REM ------------------
20800   CHSO/0=SUMO/0 IAND #FF:GOSUB 21000
20820   IF CHSO/0.LT..GT.BYTO/0 THEN PRINT CHR@(12):PRINT "CHECKSUM ERROR":GOTO 20960
20910   PRINT
20920   GOTO 20220:REM VOLGEND RECORD
20930   GOTO 20920
20950   PRINT CHR@(12):PRINT "GAP ERROR"
20960   GOTO 105
21000   BYTO/0=PEEK(FADRO/0):FADRO/0=FADRO/0+1.0
21010   SUMO/0=SUMO/0+BYTO/0
21020   RETURN
21900   GO/0=GETC:IF GO/0=0 GOTO 21900
21910   PRINT :IF GO/0=ASC("Y") THEN AO/0=0:RETURN
21920   IF GO/0=ASC(" ") THEN AO/0=1:RETURN
21930   AO/0=2:RETURN
22000   IF RADRO/0.LT.=LIMLO/0 OR RADRO/0.GT.=LIMHO/0 THEN PRINT :PRINT "VIOLATION AT ADRES ":HEX@
C  (RADRO/0):END
22010   IF DPFLO/0.LT..GT.0 THEN POKE RADRO/0,BYTO/0
22020   RETURN
```

GDAIWJTK ASSEMBLER, V2.3

S% .

PAGE 01

```
                ; KONTROLEPROGRAMMA VOOR HET VERIFIEREN
                ; VAN HET UITGEWIST ZIJN VAN EEN EPROM
                        ORG    :B000
                ;
                EPROM   EQU    :E0         ADRES EPROM KAART
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                KONTR   EQU    EPROM+3     PROGRAM REG 8255
                ;
                ;
                LEES    EQU    :90         STURING POORTEN
                SCHRYF  EQU    :80
                ; MODE=0 BIT 2=0 NORMALE POORT
                ; POORT C=OUTPUT BIT 0=0
                ; POORT B=OUTPUT BIT 1=0
                ;
                ; MODE =00 BIT5=BIT6=0
                ; NORMALE POORT
                ; POORT C= OUTPUT BIT3 =0
                ; POORT A= INPUT VOOR LEES BIT4=1
                ; POORT A= OUTPUT VOOR SCHRYF BIT4=0
                ;
                ; BIT7=1
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                PRTA    EQU    EPROM+0     ADRES POORT A
                ; BIT 0-7 ZIJN DATA LIJNEN
                ;
                PRTB    EQU    EPROM+1     ADRES POORT B
                ; BIT 0-7 ZIJN LAAGSTE ADRESLIJNEN
                ;
                PRTC    EQU    EPROM+2     ADRES POORT C
                ; BIT 0-2 HOOGSTE ADRESLIJNEN A8-A10
                ; BIT 3 NOTCE/PROG VOOR 2716 2516 2758
                ;       A11 VOOR 2732 EN 2532
                ; BIT 4 NOTOE VOOR 2716 2516 2758
                ;       NOTCE/NOTPROG VOOR 2732
                ;       NOTOE/NOT PROG VOOR 2532
                ; BIT 5 '1' VOOR 2716 2516 2758 2532
                ;       NOTOE/VPP VOOR 2732
                ; BIT 6 PROGRAMMEERRELAIS
                ; BIT 7 KEUZERELAIS
                ;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                ;
                CHB     EQU    :B1FF       CHECKBYTE
                ;       OP 0 VOOR ALLE BYTES OP FF
                ECB     EQU    :B1FE       KONTROLE BYTE
                ;                          (STUREN RELAIS)
                SAWA    EQU    :A000       STARTADRES WERKRUIMTE
                ;
                MEMR    EQU    :B1FD       MEMORY RANGE
                ;                          RELATIEF TOV SAWA
```


PAGE 02

```
                FADR    EQU    :3002       STARTADDRES OBJ.FILE
                ;
                DSAOF   EQU    :B1FA       DSAOF
                ; DESTINATION START ADDRES OBJ. FILE
                ; OBJ. FILE IS LOADED FROM THIS ADDRES
                ; IN WORK AREA
                EXOR    EQU    :B1FC       EXOR BYTE USED IN PROGRAM
                FLEA    EQU    :B1F8       FILE LENGTH ADDRES
                ;
                EXREG   EQU    :B1F5       EXTRA REGISTERS
                ;
                RWOP    EQU    :D8C8       REAL WORLD OUTPUT
                RWIP    EQU    :D8E0       REAL WORLD INPUT
                ; REGISTER D =ADRES OP DCE
                ; REGISTER E =DATA OP DCE
                ;
                ;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
B000 F5         CHECK   PUSH   PSW
B001 C5                 PUSH   B
B002 D5                 PUSH   D
B003 E5                 PUSH   H           PUSH ALL
B004 16E3               MVI    D,KONTR
B006 1E90               MVI    E,LEES
B008 CDC8D8             CALL   RWOP        8255 GEPROG OM TE
                ;                          LEZEN
B00B 3AFEB1             LDA    ECB         KONTROLELIJNEN
B00E 47                 MOV    B,A
B00F 0E00               MVI    C,0
B011 2100A0             LXI    H,SAWA      TELLER EINDE EPROM
                ;
B014 16E1       NEXTC   MVI    D,PRTB      STUREN EPROM POORT B
B016 59                 MOV    E,C         ADRES A0 TOT A7
B017 CDC8D8             CALL   RWOP        DCE-OUTPUT
                ;
B01A 16E2               MVI    D,PRTC      STUREN POORT C
B01C 58                 MOV    E,B         ADRES A8 TOT A11 EN
                ;                          KONTROLELIJNEN
B01D CDC8D8             CALL   RWOP        DCE OUTPUT
                ;
B020 16E0               MVI    D,PRTA      LEZEN EPROM POORT A
B022 CDE0D8             CALL   RWIP        DCE INPUT
B025 7B                 MOV    A,E
B026 FEFF               CPI    :FF
B028 C23CB0             JNZ    NOTER       NOT ERASED
B02B 23                 INX    H
B02C 03                 INX    B
B02D 3AFDB1             LDA    MEMR
B030 BC                 CMP    H           ?EINDE EPROM
B031 C214B0             JNZ    NEXTC
B034 3E00               MVI    A,0         VLAG ALLES OK
B036 32FFB1             STA    CHB
```

```
107 B039 C341B0          JMP   EINDE
108 B03C 3EFF     NOTER  MVI   A,:FF
109 B03E 32FFB1          STA   CHB       VLAG NIET GEWIST
110 B041 E1       EINDE  POP   H
111 B042 D1              POP   D
112 B043 C1              POP   B
113 B044 F1              POP   PSW
114 B045 C9              RET
115 B046 F5       LOAD   PUSH  PSW       LOAD OBJ.FILE IN
116                ;                      WORKAREA
117 B047 C5              PUSH  B
118 B048 D5              PUSH  D
119 B049 E5              PUSH  H
120 B04A 010230          LXI   B,FADR
121 B04D 3A0030          LDA   FADR-2    HIGH ORDER FILE
122                ;                      LENGTH
123 B050 67              MOV   H,A
124 B051 3A0130          LDA   FADR-1    LOW ORDER FILE
125                ;                      LENGTH
126 B054 6F              MOV   L,A
127 B055 09              DAD   B
128 B056 2B              DCX   H
129 B057 2B              DCX   H
130 B058 22F8B1          SHLD  FLEA
131 B05B 2AFAB1          LHLD  DSAOF
132 B05E EB              XCHG
133 B05F 0A       NEXTL  LDAX  B
134 B060 FE00            CPI   :0
135 B062 C2DAB0          JNZ   GAPE      GAP ERROR
136 B065 03              INX   B
137 B066 0A              LDAX  B
138 B067 FE00            CPI   :0
139 B069 C2DAB0          JNZ   GAPE
140 B06C 03              INX   B
141 B06D 0A              LDAX  B
142 B06E FEFF            CPI   :FF
143 B070 C2D2B0          JNZ   FL        FILE LOADED
144                ; CONTROL IF END FILE IS REACHED
145                ;
146 B073 2AF8B1          LHLD  FLEA
147 B076 7C              MOV   A,H
148 B077 B8              CMP   B
149 B078 DAD2B0          JC    FL        FILE LOADED
150 B07B C283B0          JNZ   NOTJET
151 B07E 79              MOV   A,C
152 B07F B0              CMP   L
153 B080 D2D2B0          JNC   FL
154 B083 21F5B1   NOTJET LXI   H,EXREG
155                ; EXREG CHECKSUM/EXR.+1 COUNTER /EXR.+2
156                ;                      LENGTH RECORD
157 B086 36FF            MVI   M,:FF     CHECKSUM BYTE
158 B088 03              INX   B
159 B089 0A              LDAX  B
```



```
160 B08A 86              ADD   M
161 B08B 77              MOV   M,A
162 B08C 03              INX   B
163 B08D 0A              LDAX  B
164 B08E 86              ADD   M
165 B08F 77              MOV   M,A
166 B090 03              INX   B
167 B091 0A              LDAX  B
168 B092 FE00            CPI   0
169 B094 CAA0B0          JZ    NORES     RESERVE TYPE NOT USED
170 B097 03              INX   B
171 B098 03              INX   B
172 B099 03              INX   B         RESERVE TYPE IS IGNORED
173 B09A 03              INX   B
174 B09B 03              INX   B
175 B09C 03              INX   B
176 B09D C35FB0          JMP   NEXTL
177 B0A0 03       NORES  INX   B
178 B0A1 0A              LDAX  B
179 B0A2 86              ADD   M
180 B0A3 77              MOV   M,A
181 B0A4 23              INX   H
182 B0A5 3600            MVI   M,0       CLEAR COUNTER
183 B0A7 0A              LDAX  B         LENGTH RECORD
184 B0A8 23              INX   H
185 B0A9 77              MOV   M,A
186 B0AA 2B       NEXTR  DCX   H
187 B0AB 2B              DCX   H
188 B0AC 03              INX   B
189 B0AD 0A              LDAX  B
190 B0AE 12              STAX  D
191 B0AF 13              INX   D
192 B0B0 86              ADD   M
193 B0B1 77              MOV   M,A
194 B0B2 3AFDB1          LDA   MEMR
195 B0B5 BA              CMP   D
196 B0B6 CAE2B0          JZ    MEMREX    MEM. RANGE EXCEEDED
197 B0B9 23              INX   H
198 B0BA 34              INR   M
199 B0BB 7E              MOV   A,M
200 B0BC 23              INX   H
201 B0BD BE              CMP   M
202 B0BE C2AAB0          JNZ   NEXTB     NEXT BYTE
203 B0C1 03              INX   B
204 B0C2 0A              LDAX  B         CONTROL OF CHECKSUM
205 B0C3 2B              DCX   H
206 B0C4 2B              DCX   H
207 B0C5 BE              CMP   M
208 B0C6 C2EAB0          JNZ   CHECKE    CHECKSUMERROR
209 B0C9 6B              MOV   L,E
210 B0CA 62              MOV   H,D
211 B0CB 22FAB1          SHLD  DSAOF     ADJUST DSAOF FOR
212 B0CE 03              INX   B         NEXT LOAD
```

```
213 B0CF C35FB0          JMP   NEXTL
214 B0D2 3E00      FL    MVI   A,0        FILE LOAD
215 B0D4 32FFB1          STA   CHB
216 B0D7 C3EFB0          JMP   END
217 B0DA 3E0F      GAPE  MVI   A,:F
218 B0DC 32FFB1          STA   CHB
219 B0DF C3EFB0          JMP   END
220 B0E2 3EF0      MEMREX MVI  A,:F0
221 B0E4 32FFB1          STA   CHB
222 B0E7 C3EFB0          JMP   END
223 B0EA 3EFF      CHECKE MVI  A,:FF
224 B0EC 32FFB1          STA   CHB
225 B0EF E1        END   POP   H
226 B0F0 D1              POP   D
227 B0F1 C1              POP   B
228 B0F2 F1              POP   PSW
229 B0F3 C9              RET
230 B0F4 F5        PROG  PUSH  PSW        PROGRAMMING OF
231                =                      THE EPROM
232 B0F5 C5              PUSH  B
233 B0F6 D5              PUSH  D
234 B0F7 E5              PUSH  H
235 B0F8 3AFEB1          LDA   ECB
236 B0FB 67              MOV   H,A
237 B0FC 2E00            MVI   L,0
238 B0FE FE58            CPI   :58
239 B100 CA0BB1          JZ    E16
240 B103 3E10            MVI   A,:10
241 B105 32FCB1          STA   EXOR
242 B108 C310B1          JMP   E32
243 B10B 3E08      E16   MVI   A,:8
244 B10D 32FCB1          STA   EXOR
245 B110 0100A0    E32   LXI   B,SAWA
246 B113 16E3            MVI   D,KONTR
247 B115 1E80            MVI   E,SCHRYF
248 B117 CDC8D8          CALL  RWOP       8255 GEPROG ON TE
                                            SCHRIJVEN
249                =
250 B11A 0A        NEXTP LDAX  B
251 B11B FEFF            CPI   :FF
252 B11D CA4DB1          JZ    NOPROG     NOT NECESSARY TO
253                =                        PROGRAM
254 B120 5F              MOV   E,A
255 B121 16E0            MVI   D,PRTA
256 B123 CDC8D8          CALL  RWOP       DATA IS APPLIED
257                =
258 B126 5D              MOV   E,L
259 B127 16E1            MVI   D,PRTB
260 B129 CDC8D8          CALL  RWOP       LOW ADDRES IS APPLIED
261                =
262 B12C 5C              MOV   E,H
263 B12D 16E2            MVI   D,PRTC
264 B12F CDC8D8          CALL  RWOP       PROG SIGNAL IS APPLIED
265 B132 C5              PUSH  B
```



```
266 B133 0607            MVI   B,7        LOOP 50 MSEC
267 B135 0EEB      LOOP2 MVI   C,235
268 B137 00        LOOP1 NOP
269 B138 00              NOP
270 B139 00              NOP
271 B13A 0D              DCR   C
272 B13B C237B1          JNZ   LOOP1
273 B13E 05              DCR   B
274 B13F C235B1          JNZ   LOOP2
275 B142 C1              POP   B
276 B143 3AFCB1          LDA   EXOR
277 B146 AC              XRA   H          STOP PROGRAMMING
278                =
279 B147 5F              MOV   E,A        PROG SIGNAL IS INVERTED
280 B148 16E2            MVI   D,PRTC
281 B14A CDC8D8          CALL  RWOP
282                =
283 B14D 03        NOPROG INX  B
284 B14E 23              INX   H
285 B14F 3AFDB1          LDA   MEMR
286 B152 B8              CMP   B
287 B153 C21AB1          JNZ   NEXTP
288 B156 E1              POP   H
289 B157 D1              POP   D
290 B158 C1              POP   B
291 B159 F1              POP   PSW
292 B15A C9              RET
293 B15B F5        READ  PUSH  PSW        READING THE PROM
294 B15C C5              PUSH  B
295 B15D D5              PUSH  D
296 B15E E5              PUSH  H
297 B15F 3AFEB1          LDA   ECB
298 B162 67              MOV   H,A
299 B163 2E00            MVI   L,0
300                =
301 B165 16E3            MVI   D,KONTR
302 B167 1E90            MVI   E,LEES
303 B169 CDC8D8          CALL  RWOP
304                =
305 B16C 0100A0          LXI   B,SAWA
306 B16F 16E1      NEXTR MVI   D,PRTB
307 B171 5D              MOV   E,L
308 B172 CDC8D8          CALL  RWOP
309                =
310 B175 16E2            MVI   D,PRTC
311 B177 5C              MOV   E,H
312 B178 CDC8D8          CALL  RWOP
313                =
314 B17B 16E0            MVI   D,PRTA
315 B17D CDE0D8          CALL  RWIP
316 B180 7B              MOV   A,E
317                =
318 B181 02              STAX  B
```

```
319 B182 03             INX   B
320 B183 23             INX   H
321                 ;
322 B184 3AF0B1         LDA   MEMR
323 B187 B8             CMP   B
324 B188 C26FB1         JNZ   NEXTR
325 B18B E1             POP   H
326 B18C D1             POP   D
327 B18D C1             POP   B
328 B18E F1             POP   PSW
329 B18F C9             RET
330 B190 F5     VERIFY  PUSH  PSW       VERIFY OF PROGRAMMING
331 B191 C5             PUSH  B
332 B192 D5             PUSH  D
333 B193 E5             PUSH  H
334 B194 16E3           MVI   D,KONTR
335 B196 1E90           MVI   E,LEES
336 B198 CDC8D8         CALL  RWOP
337                 ;
338 B19B 3AFEB1         LDA   ECB
339 B19E 67             MOV   H,A
340 B19F 2E00           MVI   L,0
341 B1A1 0100A0         LXI   B,SAWA
342 B1A4 0A     NEXTV   LDAX  B
343 B1A5 16E1           MVI   D,PRTB
344 B1A7 5D             MOV   E,L
345 B1A8 CDC8D8         CALL  RWOP
346                 ;
347 B1AB 16E2           MVI   D,PRTC
348 B1AD 5C             MOV   E,H
349 B1AE CDC8D8         CALL  RWOP
350                 ;
351 B1B1 16E0           MVI   D,PRTA
352 B1B3 CDE0D8         CALL  RWIP
353 B1B6 BB             CMP   E
354 B1B7 C2CBB1         JNZ   BAD
355 B1BA 03             INX   B
356 B1BB 23             INX   H
357 B1BC 3AF0B1         LDA   MEMR
358 B1BF B8             CMP   B
359 B1C0 C2A4B1         JNZ   NEXTV
360 B1C3 3E00           MVI   A,0       VERIFY OK
361 B1C5 32FFB1         STA   CHB
362 B1C8 C3D0B1         JMP   OK
363 B1CB 3EFF   BAD     MVI   A,:FF     VERIFY BAD
364 B1CD 32FFB1         STA   CHB
365 B1D0 E1     OK      POP   H
366 B1D1 D1             POP   D
367 B1D2 C1             POP   B
368 B1D3 F1             POP   PSW
369 B1D4 C9             RET
370 B1D5 CF     UT      RST   1
371 B1D6 04             DATA  4         JUMP TO UTILITY
```



```
372 B1D7                END
```

```
SYMBOL TABLE
```

BAD	B1CB	CHB	B1FF	CHECK	B000	CHECKE	B0EA
DSAOF	B1FA	E16	B10B	E32	B110	ECB	B1FE
EINDE	B041	END	B0EF	EPROM	00E0	EXOR	B1FC
EXREG	B1F5	FADR	3002	FL	B0D2	FLEA	B1F8
GAPE	B0DA	KONTR	00E3	LEES	0090	LOAD	B046
LOOP1	B137	LOOP2	B135	MEMR	B1F0	MEMREX	B0E2
NEXTB	B0AA	NEXTC	B014	NEXTL	B05F	NEXTP	B11A
NEXTR	B16F	NEXTV	B1A4	NOPROG	B140	NORES	B0A0
NOTER	B03C	NOTJET	B083	OK	B1D0	PROG	B0F4
PRTA	00E0	PRTB	00E1	PRTC	00E2	READ	B15B
RWIP	D8E0	RWOP	D8C8	SAWA	A000	SCHRYF	0080
UT	B1D5	VERIFY	B190				

```
SU.OPC UTILITY V3.3
)S2BB2 03-1
)BOBASIC V1.1
```

KRUIPER

```
10    GOTO 200:REM KRUIPER
20    H=GETC:DOT RND(XM),RND(YM) K:IF H=0 GOTO 20
30    FOR I=#2B0 TO #2B8:POKE I,0:NEXT
40    IF H=16 THEN Y=Y+E
50    IF H=17 THEN Y=Y-E
60    IF H=18 THEN X=X-E
70    IF H=19 THEN X=X+E
80    IF SCRN(X,Y)<>0 GOTO 300:DOT X,Y S
90    Z=Z+E:IF Z<35 GOTO 20:P=P+E:Z=0
100   DRAW P,P XM-P,P K:DRAW P,YM-P XM-P,YM-P K
110   DRAW P,P P,YM-P K:DRAW XM-P,P XM-P,YM-P K:GOTO 20
200   MODE 2:MODE 2:K=10:S=15:COLORG 0 5 K S
210   XM=XMAX:YM=YMAX:X=35:Y=32:Z=0:P=0:E=1:GOTO 100
300   PRINT "VERLOREN NA";Z+35*P;" STAPPEN.":PRINT
310   PRINT "SPATIEBALK VOOR NOG EEN KEER.";
320   H=GETC:IF H<32 GOTO 320:IF H=32 GOTO 10:END
```