```
1950  GOTO 250
1960  PRINT "           ALL GRAPHES"
1970  PRINT "Entrez la fonction y=f(x) en 540"
1980  PRINT "Exemple:  EDIT 540  y=sin(x)"
1990  PRINT "Entrer ensuite RUN 505"
2000  INPUT "LA FONCTION EST-ELLE INTRODUITE ";O$:IF O$<>"O" THEN STOP
2010  CLEAR 15000:NB=100.0:DIM XV(NB),YV(NB):PRINT
2020  INPUT "ENTRER LES LIMITES DE VARIATION DE X";XL,XH
2030  PRINT :DR=15.0:BK=1.0
2040  XS=(XH-XL)/NB
2050  FOR I=0.0 TO NB-1.0
2060  X=XL+I*XS
2070  GOSUB 2200
2080  XV(I+1.0)=X
2090  NEXT
2100  MODE 5:FILL 0,0 XMAX,YMAX BK
2110  FOR I=1.0 TO NB
2120  YV(I)=YMAX*(YV(I)-YL)/(YH-YL)
2130  XV(I)=XMAX*(XV(I)-XL)/(XH-XL)
2140  X0=X1:X1=XV(I)
2150  Y0=Y1:Y1=YV(I)
2160  DRAW X0,Y0 X1,Y1 DR
2170  NEXT
2180  WAIT TIME 500
2190  GOTO 2250
2200  Y=EXP(-X)
2210  YV(I+1.0)=Y
2220  IF YH<Y THEN YH=Y
2230  IF YL>Y THEN YL=Y
2240  RETURN
2250  PRINT "POUR CONTINUER 1,POUR ARRETER 0"
2260  INPUT L
2270  IF L=1.0 THEN 2010
2280  GOTO 250
```

```
1     REM PROGRAMGENERATOR By N.P. LOOIJE 11/82
2     REM BASICLINE MUST BE HIGHER THAN THE GENERATOR !!!
3     REM useful for input of complex functions into BASIC while running a
4     REM program [e.g. 1000 Y=X^2+3*X+10]. You can generate a program in a
5     REM program by putting BASIClines in DATA statements READ them and
6     REM delete the DATAlines afterwards,and by using INPUT statements.
8     REM Expensive software such as CORP,THE LAST ONE and AUTOCODE work this
9     REM way. This program makes all existing DATAstatementgenerators
10    REM oldfashioned because every BASIC statement can be generated.
11    REM Do not use this program within a FOR NEXT LOOP because FOR NEXT
12    REM loops use the symboltable which will be moved.
14    REM STUDY THE LISTING BEFORE AND AFTER THIS PROGRAM HAS RUN.
15    MODE 0:CLEAR 256:PRINT CHR$(12):GOSUB 200:REM INSTALL MLP
20    REM ----------------------GENERATE A PROGRAM----------------------------
30    LIST 300-:READ LINE$:GOSUB 100:READ LINE$:GOSUB 100:REM GEN LINE 1000,1010
40    LINE$="300":GOSUB 100:LINE$="310":GOSUB 100:LIST 300-:REM DEL LINE 300,310
50    REM --------------------- INPUT YOUR OWN LINES ---------------------------
60    INPUT "BASICLINE NO >310";LINE$:GOSUB 100
70    PRINT CHR$(12):LIST 310-:GOTO 60
99    REM -----------------CONVERT LINE$ TO BASIC LINE --------------------------
100   LINE$=LINE$+CHR$(13)+CHR$(0):A=VARPTR(LINE$):REM + dummy end (EDITbuffer)
110   A=PEEK(A)+PEEK(A+1)*256+1:B=A+LEN(LINE$):REM LINE$=begin & start EDITarea
120   POKE #A2,A MOD 256:POKE #A3,A SHR 8
130   POKE #A4,B MOD 256:POKE #A5,B SHR 8
140   POKE #135,2:CALLM #F800:RETURN:REM input from dummy EDITbuffer
199   REM -------------------MACHINELANGUAGE --------------------------------
200   FOR D=0 TO 8:READ C:POKE #F800+D,C:NEXT:RETURN
210   DATA #C5,#CD,#79,#D8,#CD,#18,#C9,#C1,#C9: REM only 9 bytes!
299   REM -------------------DATA FOR BASICLINE -----------------------------
300   DATA 1000 REM THIS IS A PROGRAMGENERATOR EXAMPLE
310   DATA 1010 MODE 0:RETURN
```

program identification	**Shadeshape**
title :	SHADESHAPE
author :	Hermano di Ciris
purpose :	Generates impossible construction
comment :	

```
5     REM SHADESHAPE by HERMANO DI CIRIS
10    MODE 6:COLORG 0 5 10 15:D%=300:R%=100
20    READ S%,T%:IF S%=0 THEN 40
30    READ X%,Y%:IF X%=0 THEN 20:DRAW S%,T% X%,Y% 23:S%=X%:T%=Y%:GOTO 30
40    WAIT TIME D%:COLORG 8 0 0 0:WAIT TIME D%
50    READ X1%,Y1%,X2%,Y2%,W%,C%:IF X1%=0 THEN 100:FOR I%=0 TO W%:DRAW X1%,Y1%-I
% X2%,Y2%-I% C%:NEXT:GOTO 50
100   WAIT TIME D%
110   COLORG 8 5 10 15:WAIT TIME D%:COLORG 8 1 0 0:WAIT TIME R%
120   COLORG 8 0 2 0:WAIT TIME R%:COLORG 8 0 0 3:WAIT TIME R%
140   GOTO 110
1000  DATA 83,127,52,145,52,162,220,255,235,246,235,211,265,228
1010  DATA 280,219,280,35,265,25,235,42,235,9,220,0,52,93,52,110
1020  DATA 83,127,0,0
1030  DATA 235,246,83,162,114,145,205,195,205,212,0,0,220,186,220,220,99,153,0,0
1040  DATA 52,110,205,25,205,59,114,110,99,102,205,42,0,0
1050  DATA 265,26,265,194,235,177,235,77,250,68,250,185,0,0
1060  DATA 205,212,205,195,114,145,0,0,205,195,220,186,0,0
1070  DATA 99,102,114,110,205,59,0,0,114,110,114,127,0,0
1080  DATA 235,211,235,195,280,219,0,0,83,127,99,136,52,162,0,0
1090  DATA 235,42,220,51,220,0,220,51,0,0,205,77,205,161,0,0
1100  DATA 250,68,250,51,220,68,220,169,145,127,205,94,0,0
1110  DATA 205,77,114,127,83,110,99,102,0,0,220,68,235,77,0,0
1120  DATA 220,186,129,136,145,127,0,0,0,0
2000  DATA 52,162,67,170,18,22,52,110,67,119,18,22
2002  DATA 220,86,250,68,18,22
2004  DATA 220,51,235,59,51,23,83,162,99,170,17,23
2006  DATA 83,110,145,145,17,23
2007  DATA 52,110,205,25,17,21,205,161,220,169,18,21
2009  DATA 205,161,220,152,152,21
2010  DATA 220,68,265,43,18,21,114,127,205,77,18,21
2015  DATA 250,185,265,194,18,21
2020  DATA 250,185,265,176,134,21,52,162,83,145,18,21
2025  DATA 83,145,99,136,16,21,205,212,220,220,18,21
2030  DATA 205,212,220,203,18,21
2035  DATA 220,255,235,246,18,22,67,171,220,255,18,22
2040  DATA 67,170,83,162,16,22,83,163,205,94,17,22
2050  DATA 83,111,204,43,18,22
2060  DATA 235,211,265,228,18,22,265,228,280,219,18,22
2070  DATA 67,119,205,195,17,22
2075  DATA 205,195,220,186,18,22,67,119,83,110,17,22
2080  DATA 145,144,265,211,17,23,265,211,280,219,185,23
2090  DATA 99,170,220,237,16,23,220,237,235,246,53,23
2100  DATA 220,169,235,177,101,23
5000  DATA 0,0,0,0,0,0
```

INDATA News

Nieuwe versie Dai Masterdos voor huidige floppy drives.

Er is een toevoeging aan Dai Masterdos aangebracht, dat het mogelijk maakt sectoren en tracks direkt te lezen en te schrijven.

Deze toevoeging werd gemaakt op algemeen verzoek en geeft de mogelijkheid tot het maken van een echte database.

de syntax is als volgt:

RREC filename.ext sect memplace(hex)
WREC filename.ext sect memplace(hex)

of als voorbeeld:

RREC TEST.BAS 1 5000 :

leest de eerste sektor van de file "test.bas" en zet deze informatie op hex 5000.
Alle namen en waarden kunnen ook variabelen zijn, zodat programmamatig deze waarden kunnen worden aangepast.

Prijs: Bf 500,-- voor een disk met uitleg.

SERVICE MANUAL

Voor de DAI Personal computer is er onlangs een service manual uitgegeven.

Dit manual geeft een zeer uitgebreide beschrijving van de hardware funkties, alsmede timing diagrammen, memory map, processor-, ram-, rom- en video beschrijvingen en verder alles wat een professioneel gebruiker moet weten om het interne van de machine goed te leren kennen.

En natuurlijk ook 16 bladen met het volledig schema van de computer.

Prijs: Bfr. 1500,--

GEBRUIK VAN AZERTY USER

1- Steek de cassette in de DCR en schakel de computer aan.
Nadat de DCR is gestopt kan U gewoon beginnen werken met het AZERTY toetsenbord !!!

2- Duwt U op reset zonder dat de cassette in de DCR zit,dan wordt Uw toetsenbord opnieuw als een QWERTY toetsenbord aanzien.

3- U kan echter zonder probleem terug een AZERTY toetsenbord bekomen indien U voordien gebruik hebt gemaakt van de USER cassette door vanuit BASIC in directe mode

```
CALLM #2F0
```

in te tijpen. Let echter op! De M bekomt U door op de vijfde toets van links op de onderste rij te drukken.
Dit is de plaats waar de QWERTY M zit.
Kijkt U altijd goed na wat op het scherm verschijnt.

4- Doe nooit een

```
G2F0
```

vanuit de utility-mode. Uw computer loopt dan gegarandeerd vast. Indien U in BASIC-mode een AZERTY toetsenbord heeft dan hebt U automatisch ook een AZERTY toetsenbord voor alle programma's in BASIC en in utility.

5- Het programma is niet als dusdanig te gebruiken met andere programma's die onder de HEAP zitten zoals bvb. het FGT programma. Indien U toch een aanpassing wenst om ook met dergelijke programma's te kunnen werken met een AZERTY toetsenbord, dan neemt U best contact op met mij. Vermeld steeds duidelijk om welk programma het gaat en geef ook de belangrijke gegevens zoals: begin & eindadres, entrypoint, versienummer van het programma, enz.
Vermeld ook een telefoonnummer indien U dit wenst. Hou er wel rekening mee dat ik U alleen tijdens het weekend kan terugbellen.(evt. prive-telefoon vermelden)

6- U kan een backup maken van het programma op de volgende manier:

```
   *  REW              <ret>     terugspoelen nieuwe cassette
   *  REW              <ret>     terugspoelen van de USER-cassette
   >  UT               <ret>     naar uitility-mode gaan
   >  R                <ret>     inlezen van de USER-file
   >                             nieuwe cassette in de DCR steken
   >  W2F0 37E USER    <ret>     copieren van de USER-file
   >  B                          terug naar BASIC
   *
```

??? Voor vragen,problemen en suggesties kan U terecht op
??? volgend adres:

Jos Schepens

Sint Jorisgilde 53
B-9330 DENDERMONDE (BELGIE)

??? Telefonisch ben ik te bereiken op het nummer 052/21 67 43
??? Alleen op zaterdag & zondag tussen 14.00 & 20.00

```
]#L.
    PAGE 01    USER-AZERTY 13-02-83 SCHEPENS V0.1

    002                 ************************************************
    003                 *           USER  AZERTY                       *
    004                 *                                              *
    005                 * THIS IS A PROGRAM TO RECONFIGURE              *
    006                 * A QWERTY KEYBD. TO AN AZERTY ONE.             *
    007                 * !!! USER-FILE ON DCR !!!                      *
    008                 * CREATED           12-02-83                   *
    009                 * LAST REVISION     13-02-83                   *
    010                 * COPYRIGHT BY      Jos Schepens DENDERMONDE   *
    011                 ************************************************
    012                 HEAP    EQU   :029B
    013                 HSIZE   EQU   :029D
    014                 RNEW    EQU   :DEB5
    015                         ORG   :2F0
    016 02F0 218003     ENTRY   LXI   H,AZEND+1
    017 02F3 229B02             SHLD  HEAP
    018 02F6 210001             LXI   H,:100
    019 02F9 229D02             SHLD  HSIZE
    020 02FC CDB5DE             CALL  RNEW
    021 02FF 3E10               MVI   A,:10
    022 0301 323D01             STA   :013D
    023 0304 210D03             LXI   H,AZTAB
    024 0307 2200F8             SHLD  :F800
    025 030A C383C7             JMP   :C783
    026 030D 303132     AZTAB   ASC   '012'
    027 0310 333435             ASC   '345'
    028 0313 363738             ASC   '678'
    029 0316 39                 ASC   '9'
    030 0317 2F4D3B             ASC   '/M;'
    031 031A 5E3A2D             ASC   '^:-'
    032 031D 0D                 DATA  :0D
    033 031E 514243             ASC   'QBC'
    034 0321 444546             ASC   'DEF'
    035 0324 474849             ASC   'GHI'
    036 0327 4A4B4C             ASC   'JKL'
    037 032A 2C                 DATA  :2C
    038 032B 4E4F50             ASC   'NOP'
    039 032E 415253             ASC   'ARS'
    040 0331 545556             ASC   'TUV'
    041 0334 5A5859             ASC   'ZXY'
    042 0337 572E5B             ASC   'W.['
    043 033A 20                 DATA  :20
    044 033B 000810             DATA  :00,:08,:10
    045 033E 111213             DATA  :11,:12,:13
    046 0341 098000             DATA  :09,:80,:00
    047 0344 00                 DATA  :00
    048 0345 302122             ASC   '0!"'
    049 0348 232425             ASC   '#$%'
    050 034B 26                 ASC   '&'
    051 034C 27                 DATA  :27
    052 034D 28293F             ASC   '()?'
    053 0350 6D2B7E             ASC   'm+~'
    054 0353 2A3D               ASC   '*='
```

```
PAGE 02    USER-AZERTY 13-02-83 SCHEPENS V0.1

055 0355 0D                 DATA  :0D
056 0356 716263             ASC   'qbc'
057 0359 646566             ASC   'def'
058 035C 676869             ASC   'ghi'
059 035F 6A6B6C             ASC   'jkl'
060 0362 3C6E6F             ASC   '<no'
061 0365 706172             ASC   'par'
062 0368 737475             ASC   'stu'
063 036B 767A78             ASC   'vzx'
064 036E 79773E             ASC   'yw>'
065 0371 5D20               ASC   '] '
066 0373 000814             DATA  :00,:08,:14
067 0376 151617             DATA  :15,:16,:17
068 0379 0C8000             DATA  :0C,:80,:00
069 037C 004A53             DATA  :00,:4A,:53
070 037F            AZEND   END

****************************
* S Y M B O L   T A B L E *
****************************

AZEND  037F   AZTAB  030D   ENTRY  02F0   HEAP   029B
HSIZE  029D   RNEW   DEB5
```

```
100   REM
110   REM
120   REM ... A SPEED COMPARISON BETWEEN BASIC AND PLM
130   REM
140   REM
150   REM Gianni Uliana
160   REM Via Zuccoli 40    00137 ROMA      ITALY
170   REM
300   CLEAR 2000:DIM A(38.0)
310   FOR I%=0.0 TO 38.0:READ B%:POKE (#2F1+I%),B%:NEXT
320   REM ~~~~~~~~~~~~~~~~~~~~~ BASIC ~~~~~~~~~~~~~~~~~~~~~~
330   PRINT CHR$(12):CURSOR 18,12:PRINT " BASIC  "
340   WAIT TIME 100:PRINT CHR$(12)
350   I=#B3E7
360   SOUND 1 1 1 0 FREQ(138.0)
370   FOR X=I TO I+131.0 STEP 2.0
380   POKE X,N
390   N=N+1.0:IF N>255.0 THEN N=0.0
400   NEXT
410   IF I<#BF69 THEN I=I+134.0:GOTO 370
420   SOUND OFF :WAIT TIME 100
430   REM ~~~~~~~~~~~~~~~~~~~~~ PLM  ~~~~~~~~~~~~~~~~~~~~~
440   PRINT CHR$(12):CURSOR 24,12:PRINT " PLM "
450   WAIT TIME 100:PRINT CHR$(12)
460   SOUND 1 1 1 1 FREQ(1975.0)
470   CALLM #2F1
480   SOUND OFF :WAIT TIME 100
490   GOTO 320
500   DATA #C5,#D5,#E5,#F5,#16,#3E,#3E,#00,#1E,#17,#01
510   DATA #EB,#BF,#02,#3C,#0B,#0B,#15,#C2,#FE,#02,#16
520   DATA #0A,#0B,#15,#C2,#08,#03,#16,#3E,#1D,#C2,#FE
530   DATA #02,#F1,#E1,#D1,#C1,#C9
```

RUN (Numero de ligne) AVEC LE BASIC V1.0

Avec le BASIC V1.0 , le redémarrage d'un programme à partir d'un numero de ligne donné peut etre a l'origine d'une erreur non recuperable à l'execution , la table des symboles et la zone de stockage des variables ayant été vidées.
La solution suivante est proposée par G.GRUITERS :

1) En utilisant la procedure S (substitute) en UT,tapez en code objet le court programme en MLP appelé RUNLIN :

```
23 23 56 23 5E EB CD F6 CA 44 4D CD 01 E4 21 00
00 22 15 01 AF 32 26 01 31 00 F9 B7 C3 8F C8
```

2) Ajustez le pointeur de pile 29B-29C,mettre le pointeur après la dernière adresse du code objet.

3) Tapez NEW

4) Chargez votre programme BASIC et tapez RUN

5) Pour redemarrer à un certain numero de ligne,definir d'abord une variable entiere a laquelle on donne la valeur du numero de ligne desire , suivi par un appel au MLP.

EXEMPLE : Supposons que l'adresse de depart du MLP soit #300. Redemmarage a la ligne 100

Tapez les commandes suivantes en mode direct :

```
LINE%=100:CALLM #300,LINE%
```

NOTA :
- On peut charger automatiquement le MLP et le programme BASIC en utilisant le DAINAMIC BOOTSTRAP LOADER. Dans ce cas on peut omettre les etapes 2 et 3.
- Si on redemarre souvent au meme numero de ligne, incorporer la variable au programme BASIC , par exemple:

```
10 LINE%=100
```

et redemarrer en faisant seulement CALLM #300,LINE%

DU NOUVEAU POUR LES PADDLES !!!

Par cet article je vais tenter de vous decrire l'organisation materielle et logicielle des PADDLES . Elle debouche sur une routine de conversion Analogique/ Digitale plus perfomante que celle habituellement utulisée.

1) Principe de base (Simplifié)

Chaque paddle est branchée sur un MVM [1] (1/2 556). Quand une impulsion est envoyée sur la broche Trigger [2] de ce circuit, par l'intermediaire de l'adresse 0FD01H, sa sortie change d'etat pendant une durée "d" précisement déterminée par un reseau RC (Resistance-Condensateur). Or la resistance dependant de la position du potentiometre du paddle la durée "d" est donc elle aussi fonction de cette position. Le changement d'etat est transmis à un circuit (8253) qui se met à decompter. Au bout de la periode "d" le MVM revient a son etat initial ce qui a pour effet de stopper le decomptage sur une valeur precise. Il ne reste plus qu'a lire et a mettre cette valeur dans un format convenant à l'utilisateur.

2) Rapport de Temps

Il donne une idée des differentes durée qui peuvent etre en raport avec le temps de la conversion.

a) Le microprocesseur

L'horloge du processeur tourne a 2 MHZ (On trouve un quartz de 18 MHZ dont la frequence est divisée par 9 par un circuit specialisé : le 8224). Nous supposerons donc que la durée d'un cycle [3] est de 0,5 µs (En fait cette durée peut varier en fonction du temps d'accés aux memoires). Une instruction telle que JMP,par exemple, qui prend 10 Cycles mettrait donc 5 µs a etre executée.

b) Les paddles

La durée de l'impulsion du MVM est donné par la relation :

$$1{,}1 \times R \times C$$

Sur le DAI le reseau RC est composé d'un condensateur de 1,2 nF et d'une resistance de 15 Kohms en série avec un potentiometre de 100 Kohms. Après calcul on arrive aproximativement aux resultats suivants :

Durée minimum (R= 15 Kohms) 20 µs
durée maximum (R= 115 Kohms) 150 µs

La durée maximum de la conversion est donc proche de 150 µs ce qui represente 300 cycles d'horloge.

3) La routine de conversion du DAI

Le programme se presente ainsi :

```
0000            ;Le numero du paddle est selectionne grace aux
0000            ;trois premier bits de l'adresse 0FD06H.
0000            ;Le quatrieme bits doit etre a 1 pour autoriser
0000            ;la conversion . Attention car cette adresse
0000            ;contient aussi le commutateur des banques
0000                    ORG     0EBD3H          ;Banque0
EBD3  3E30              MVI A   30H             ;Initialisation du circuit 8253 en
EBD5  0106FC            LXI B   0FC06H          ;mode 'decompteur'
EBD8  02                STAX B
EBD9  21FFFF            LXI H   0FFFFH          ;Le decompteur est charge avec la
EBDC  2200FC            SHLD    0FC00H          ;plus grande valeur possible
EBDF  3A01FD            LDA     0FD01H          ;'lance' le MVM (Trigger)
EBE2  EB        PDL10   XCHG
EBE3  3E00              MVI A   0H
EBE5  02                STAX B
```

```
EBE6  2A00FC        LHLD    0FC00H      ;lit le contenu du compteur et le
EBE9  CD14DE        CALL    0DE14H      ;compare avec sa valeur precedente
EBEC  DAE2EB        JC      PDL10       ;tant qu'elle est inferieur continue
EBEF  CD26DE        CALL    0DE26H      ;quand elle est fixe le decompteur est
EBF2  11CEFF        LXI D   0FFCEH      ;alors arrete
EBF5  19            DAD D               ;Apres ce calcul H contient la valeur
EBF6                END                 ;du PDL . . . ect
```

Nota :Le programme contient une boucle d'attente. Il suffit de lire l'adresse 0FD01H pour envoyer l'impulsion de depart (Trigger).

4) Modification proposée

Dans des programmes ou les durées seront critique, on pourra reduire le temps d'éxecution de la routine de facon appreciable. On aura :

```
0000                ORG     5000H
5000  3E30          MVI A   30H
5002  0106FC        LXI B   0FC06H
5005  02            STAX B
5006  21FFFF        LXI H   0FFFFH
5009  2200FC        SHLD    0FC00H
500C  3A01FD        LDA     0FD01H
500F          ; A la place de la boucle d'attente il suffit
500F          ; de placer ici une partie de votre programme
500F          ; comprenant au moins une soixantaine d'instructions
500F          ; pour laisser au decompteur le temps de s'arreter
500F          ; Il n'y a pas de registres a sauver, il faut
500F          ; seulement veiller a ne pas utiliser le premier
500F          ; oscillateur du 8253 (0FC00H/0FC01H)
500F          ;       XRA A
500F  3206FC          STA     0FC06H
5012  2A00FC          LHLD    0FC00H
5015  CD26DE          CALL    0DE26H
5018  11CEFF          LXI D   0FFCEH
501B  19              DAD D
501C  7C              MOV A,H             ;HL contient la valeur du PDL
501D          ; ... ect
501D                  END
```

Nota :Il n'y a pas de limite maximale à respecter quant au nombres d'intructions que l'on peut mettre à la place de la boucle. En MOYENNE une intrution prend 6,5 cycles ce qui justifie le choix de 60 et respecte une bonne marge de 'securité'

Bonnes Conversions !
Cedric DUFOUR

Ouvrages de references :

Le FIRMWARE Manual par B.J. Boerrigter
DAI p.c. Schematics idem

1 Un MVM (MultiVibrateur Monostable) est un montage qui fournit à chaque impulsion d'entrée une impulsion de sortie d'une durée précise qui ne depend pas de celle d'entrée.

2 Trigger est la broche d'entrée du MVM (Voir 1)

3 On peut dire qu'un cycle represente UNE opération interne dans le processeur. Toute instrucrion (Operation programmable par l'utilisateur) necessite un certains nombre de ces operations.

```
10    REM *** SCREEN TO BUFFER / BUFFER TO SCREEN **********
20    REM *** GESCHREVEN DOOR DE BONT C.(NAAR HET PROGRAM **
30    REM *** VAN W.HERMANS UIT NEWSLETTER 16,PAGINA 168) **
40    REM ************************************************
50    CLEAR 4100:POKE #29B,0:POKE #29C,#10
60    FOR X=#D00 TO #D3F:CURSOR 20,3:PRINT #D3F-X;" ";
70    READ A:POKE X,A:NEXT
80    MODE 0:PRINT CHR$(12);:COLORG 8 0 3 14
90    PRINT "*************************************************************"
100   PRINT "*              SCREEN  BUFFER : BUFFER  SCREEN              *"
110   PRINT "*              ================================               *"
120   PRINT "*  DIT PROGRAMMA LAADT VAN UIT DATA EEN MLP IN RAM,WAAR-      *"
130   PRINT "*  DOOR HET MOGELIJK WORDT TEKENINGEN VANAF HET SCHERM        *"
140   PRINT "*  NAAR EEN BUFFER TE VERPLAATSEN EN VICE VERSA.ZOWEL IN      *"
150   PRINT "*  DE LAGE ALS IN DE MEDIUM RESOLUTIES IS DEZE TRANSACTIE     *"
160   PRINT "*  MOGELIJK.HELAAS IS BIJ HET GEBRUIK VAN MODE 5 OF 6 DE      *"
170   PRINT "*  GEHEUGENCAPACITEIT NAUWELIJKS TOEREIKEND OM ZOWEL EEN      *"
180   PRINT "*  PROGRAMMA ALS EEN BUFFER TE BEVATTEN.DAAROM BLIJFT DIT     *"
190   PRINT "*  PROGRAMMA BEPERKT TOT HET GEBRUIK IN MODE 1-2-3-4.         *"
200   PRINT "*     (BIJ HET GEBRUIK VAN SPLIT-SCREEN MODE ZAL OOK DE       *"
210   PRINT "*      ONDERSTAANDE TEKST IN HET BUFFER OPGENOMEN WORDEN)     *"
220   PRINT "*                                                             *"
230   PRINT "*         WELKE GRAFISCHE MODE WILT U (1 2 3 4)               *"
240   PRINT "*************************************************************"
250   G=GETC:IF G<49 OR G>52 THEN WAIT TIME 3:GOTO 250
260   PRINT CHR$(12);:IF G=49 THEN MODE 1A:GOTO 300
270   IF G=50 THEN MODE 2A:GOTO 300
280   IF G=51 THEN MODE 3A:GOTO 300
290   MODE 4A
300   PRINT TAB(10);"- RANDOM TEKENING IN MODE ";CHR$(G);"A -"
310   FOR X=50 TO 1 STEP -1:CURSOR 20,2:PRINT X;" ";
320   A=INT((XMAX-30.0)*RND(1.0)):B=INT((YMAX-30.0)*RND(1.0))
330   C=INT(30.0*RND(1.0))+A:D=INT(30.0*RND(1.0))+B
340   IF G MOD 2.0=1.0 THEN E=INT(15.0*RND(1.0)):GOTO 360
350   E=INT(4.0*RND(1.0))+20.0
360   FILL A,B C,D E:NEXT:PRINT
370   CURSOR 10,0:PRINT "DRUK OP DE SPATIEBALK ";
380   CALLM #D6DA:PRINT CHR$(12);
400   PRINT TAB(10);"---*** SCREEN TO BUFFER ***---"
410   WAIT TIME 40:PRINT CHR$(12);:CALLM #D00
420   CURSOR 10,0:PRINT "DRUK OP DE SPATIEBALK ";
430   CALLM #D6DA:PRINT CHR$(12);
500   PRINT TAB(10);"- IK WIS NU DE TEKENING UIT! -"
510   FILL 0,0 XMAX,YMAX 8
520   CURSOR 10,0:PRINT "DRUK OP DE SPATIEBALK ";
530   CALLM #D6DA:PRINT CHR$(12);
600   CALLM #D1C
610   PRINT TAB(10);"---*** BUFFER TO SCREEN ***---"
620   PRINT TAB(7);"EN DAAR IS DE TEKENING WEER TERUG !!"
700   CURSOR 10,0:PRINT "NOGEENS PROBEREN (J/N) ?";
710   G=GETC:G=GETC:G=GETC
720   G=GETC:IF G=0 THEN WAIT TIME 3:GOTO 720
730   IF G=74 OR G=106 THEN PRINT CHR$(12);:GOTO 80
740   IF G=78 OR G=110 THEN PRINT CHR$(12);:GOTO 760
750   GOTO 720
760   CURSOR 20,2:PRINT "E I N D E":GOTO 990
800   REM *** MODE -CORRECTIE
810   MK=PEEK(#9D)
820   IF MK<4 THEN AM=2143:GOTO 860:REM ---------- MODE 1,2
830   IF MK>3 AND MK<8 THEN AM=6563:GOTO 860:REM - MODE 3,4
840   IF MK>7 AND MK<11 THEN PRINT "MODE 5 EN 6:ONMOGELIJK"
850   PRINT "FOUTIEVE MODE ":END
860   BS=#BFEF-(AM*2)
865   POKE #D05,AM MOD 256:POKE #D06,AM SHR 8
```

```
870   POKE #D0B,BS MOD 256:POKE #D0C,BS SHR 8
880   POKE #D21,AM MOD 256:POKE #D22,AM SHR 8
890   POKE #D24,BS MOD 256:POKE #D25,BS SHR 8:RETURN
900   REM *** CALLM #D00 : SCREEN TO BUFFER
910   REM *** CALLM #D1C : BUFFER TO SCREEN
920   DATA #E5,#C5,#D5,#F5,#11,#2E,#17,#21
930   DATA #EF,#BF,#01,#00,#90,#7E,#02,#2B
940   DATA #0B,#1B,#7A,#B3,#C2,#0D,#0D,#F1
950   DATA #D1,#C1,#E1,#C9,#E5,#C5,#D5,#F5
960   DATA #11,#2E,#17,#21,#00,#90,#01,#EF
970   DATA #BF,#7E,#02,#2B,#0B,#1B,#7A,#B3
980   DATA #C2,#29,#0D,#F1,#D1,#C1,#E1,#C9
990   DATA #00,#00,#00,#00,#00,#00,#00,#00
```

ON ERROR GOTO demo

```
100   REM *** ON ERROR GOTO DEMO **************************
110   REM *** GESCHREVEN DOOR : DE BONT CORNEEL ************
120   REM *** GEBASEERD OP HET PROGRAMMA VAN N.P. LOOIJE **
130   REM *** ( NEWSLETTER 16 : PAGINA 172-173 ) ***********
140   REM *** SUBROUTINE 300-330 SCHAKELT DE 'ON ERROR' ***
150   REM *** AAN OF UIT NAARGELANG HUIDIGE STATUS ********
160   REM *** INLEZING MLP-ROUTINE VANUIT DATA
170   CLEAR 4000:POKE #29B,#60:POKE #29C,#3:PRINT CHR$(12);
180   FOR X=#300 TO #35F:CURSOR 20,20:PRINT #35F-X;" ";
190   READ PQ:POKE X,PQ:NEXT
200   REM *** DEMO PROGRAM
210   MODE 0:PRINT CHR$(12);:LNR=230:LIST 200-290
220   GOSUB 300:REM --------------SWITCH 'ON ERROR GOTO' ON
230   A=A+1:IF A<10 THEN PRINT " ";
240   PRINT A;"e MAAL ON ERROR"
250   IF A>10.0 THEN 270:REM -----------------10 x ON ERROR
260   DOT A,500 500:GOTO 230:REM -----NUMBERS OUT OF RANGE
270   GOSUB 300:REM -------------SWITCH 'ON ERROR GOTO' OFF
280   GOTO 260:REM ----NOW AN ERROR MESSAGE WILL BE PRINTED
290   END
300   REM *** 'ON ERROR GOTO' - ON EN OFF (LNR%=LIJNNUMMER)
310   IF PEEK(#6C)=#FD THEN POKE #6C,0:POKE #6D,#3:GOTO 330
320   IF PEEK(#6C)=0 THEN POKE #6C,#FD:POKE #6D,#C6
330   POKE 6,LNR IAND #FF:POKE 7,LNR SHR 8:RETURN
400   REM *** MLP 'ON ERROR GOTO' 300-35F IN DATA
410   DATA #F5,#2A,#06,#00,#7C,#B5,#CA,#21,#03,#21,#0A,#00
420   DATA #39,#7E,#FE,#53,#C2,#21,#03,#23,#7E,#FE,#DA,#C2
430   DATA #21,#03,#23,#7E,#FE,#40,#CA,#25,#03,#F1,#C3,#FD
440   DATA #C6,#2A,#06,#00,#CD,#F6,#CA,#D2,#4D,#03,#44,#4D
450   DATA #21,#00,#01,#3E,#15,#36,#00,#23,#3D,#C2,#35,#03
460   DATA #F3,#32,#31,#01,#32,#40,#00,#32,#06,#FD,#FB,#31
470   DATA #00,#F9,#C3,#92,#C8,#21,#00,#00,#22,#06,#00,#3E
480   DATA #04,#C3,#F5,#D9,#00,#00,#00,#00,#00,#00,#00,#00
```

DAI VIDEO SCREEN RAM

(from DAInamic 12, page 248)

DEAR DAI FRIENDS.

Here are some more programs for you. Two of them complement, with demonstrations, the article in DAInamic 10 on 240*528 resolution, ie, MODE 7 and MODE 8. There is also an explanation of the pointers. After reading the aforementioned article I was keen to try it out. The programme was typed in but alas, in spite of the screen being initialised nothing else would happen. I wanted at once to find out why and set about investigating what the DAI manual had to say on screen pictures and control words. The investigations produced the following:-
SCREEN STRUCTURE MODE 2A after COLORT 0 1 2 3, COLORG 0 1 2 3
N.B. Everything is referenced from the TOP of the screen.

```
#BFFF
* SCREEN ------------------------------------------------------------------------------------------|
|  36  80--00  00    UNITCOLORMODE  /  COLORG 0       ==========                                  |
|  36  91--00  00        '     '    /  COLORG 1       = HEADER =                                  |
|  36  A2--00  00        '     '    /  COLORG 2       ==========                                  |
|  36  B3--00  00        '     '    /  COLORG 3                                                   |
* SCTOP -------------------------------------------------------------------------------------------|
|  03  40--00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  |
|  total 53 [GRL-GAL] lines of GRAPHICS with #18 [GXB] bytes/line                                  |
* GRE ---------------------------------------------------------------------------------------------|
|  30  80--00  00    UNITCOLORMODE  /  COLORT 0                                                    |
|  30  91--00  00        '     '    /  COLORT 1       ================                            |
|  30  A2--00  00        '     '    /  COLORT 2       = INTERMEDIATE =                            |
|  30  B3--00  00        '     '    /  COLORT 3       ================                            |
* CHS ---------------------------------------------------------------------------------------------|
|  7A  40--20  00  (66*[20 00])                                                                    |
|  total 4 [TXL] lines of TEXT with #86 [TXB] bytes/line                                           |
* CHE ---------------------------------------------------------------------------------------------|
|  3F  80--00  00    UNITCOLORMODE  /  COLORT 0                                                    |
|  3F  91--00  00        '     '    /  COLORT 1       ==========                                  |
|  3F  A2--00  00        '     '    /  COLORT 2       = TRAILER =                                 |
|  3F  B3--00  00        '     '    /  COLORT 3       ==========                                  |
* SCE/GTS -----------------------------------------------------------------------------------------|
|  03  40--00  00  00  00 00  00  00  00  00  00   00  00  00  00  00  00  06  00  00  00  00  00  |
|  total 12 [GAL] lines scrolled (from the upper part of screen)                          *FFB     |
* GTE ---------------------------------------------------------------------------------------------|
#B8B7
```

#BFFF-#BFF0 this line contains the HEADER. For mode 2A it is:
#BFFF-#BFFD 36 80 00 00 where ....
#36 is the MODE BYTE 'HIGH address'
bit 7,6 [00] display-mode control : 4 colour graphics
bit 5,4 [11] resolution control : 528 dots per line
bit 3,2,1,0 [0110] line repeat count : 6 repartitions
#80 is the COLOUR TYPE BYTE 'LOW address'

bit 7 [1] makes possible a colour change
bit 6 [0] puts this line in 'unit colour mode' so that the two data bytes can be repeated as many times as required by the resolution control bits.
bit 5,4 [00] defines colour register 0 from the COLORG in bits 3-0. Each time the HIGH and LOW data bits are 0 and 0 the colour from bits 3-0 are used. This continues until colour 0 is changed in another colour type byte by means of bits 7, 5 and 4.
bit 3,2,1,0 [0000] selects colour 0 from the 16 colours.
#00 is the HIGH BYTE 'HIGH address'
bit 7-0 [00000000] in conjunction with the low byte selects colour 0 from COLORG for all dots
#00 is the LOW BYTE 'LOW address'
bit 7-0 [00000000] see HIGH address. These two bytes will be repeated 65 times by the 'unit colour mode'

The subsequent 12 bytes will be dealt with in a similar way in the 'unit colour mode'. This makes a total of 28 empty lines. Bits 3-0 of #BFFA, #BFF6 and #BFF2 contain respectively colours 1, 2 and 3 from the COLORG registers, bits 5 and 4 stipulating which ones. These 28 empty lines appear in this way in all 4 colour modes. In 16 colour modes something rather similar happens, only here the bits with the 4 colour mode data are replaced by the 16 colour data. This means that among other things, colour changes do not occur at once after COLORG. The intermediate and trailer (line repeat count 0 and #F respectively) operate the same way but with data from COLORT. COLORG applies to the HEADER. COLORT applies to the INTERMEDIATE and TRAILER. The line mode bytes for graphics and text contain data for display mode control, resolution control, line repeat count and the length of the lines. The colour byte remains neutral, bit 7=0 (no colour change) and bit 6=1 (no unit colour mode). Information on the above is detailed in the manual in the section entitled PROGRAMMABLE GRAPHICS GENERATOR.

MAXIMUM RESOLUTION - 512*244 MODE 7 & 8
We can read in the manual that a max of 32K byte of RAM, #4000 - #BFFF, is available for the screen. Calculation shows that with 528 dots per line there are #86 (134) bytes per line and so 32768/134 = 244 lines max are possible; thus more than 240. The control words must be:-

MODE BYTE
bit 7,6 [00] for 4 colour graphics
[10] for 16 colour graphics
bit 5,4 [11] for 528 dots per line
bit 3,2,1,0 [0000] for 1 line, so no repeats
COLOUR TYPE BYTE
bit 7 [0] bits 5 - 0 ignored
bit 6 [1] no unit colour mode
bit 5-0 [XXXXXX] no effect
Thus we find for MODE 7 #B0 #40 and for MODE 8 #30 #40. The start-up for MODE 7 is via MODE 1, 3, or 5 and MODE 8 via MODE 2, 4, or 6.

From #BFEF with a step of #86 these bytes will be filled in 244 times. Afterwards must come a trailer (line repeat count #F) identical to the header and which serves to fill the screen with empty lines. If we do not do this we get a muddle lower down the screen. If this happens the screen is initialised but we can do nothing else with it. The pointers for the graphical functions must then be set with CURRENT STATE OF SCREEN, CHARACTER MODE and START OF SCREEN. After a short period of experimentally setting the various pointers it seems that DOT, DRAW, FILL and SCRN work excellently. Only error messages like COLOR NOT AVAILABLE and OFF SCREEN were given; XMAX apparently is hardware restricted to 511 and YMAX to 243.

The new pointers are:-

80/81	SCREEN	#BFFF	90/91	GTE	XXXX
82/83	SCTOP	#BFEF	92/93	GAS/GTS	XXXX
84/85	FFB	#4027	94/95	GRC	#0200
86/87	GRR	XXXX	96	GRL	# F4
88/89	GRE	#4037	97	GAL	# XX
8A/8B	CHS	#4027	98	GXB	# 86
8C/8D	GAE/CHE	#4037			
8E/8F	SCE	#4027			

HOW TO CALCULATE THE POINTERS

Without having had to study the ROM I have managed to discover the following values for the pointers. With some help from the memory map V4.4 I have tested the screen functions; there may be defects but at least it works in all modes. You can see the locations of the pointers in the sketches. The important ones for designing in a single mode are given below with a brief description and the method of calculating them. They are:-

A. Text (MODE 0)
B. Graphics (MODE 1-8)
C. Split mode, text below (MODE 1A-8A)
D. Split mode, text above (MODE 1B-8B)

Some useful values:-

HDRL = header length (normally #10)
INTL = length of intermediate (normally #10)
TRAL = trailer length (normally #10)
TXL = number of text lines
TXB = number of bytes per text line (#86) may also be #5A, #2E or #18 but the screen driver can only work with #86

#72/#73 CURSOR.

Position of the cursor on the screen. The setting for the cursor in a new text or split-mode must be 8 bytes after the line mode byte. In a graphics mode the cursor position will be set to 0000.

A. + C. + D. >> CHS - 8
B. >> 0000

#78/#79 LNSTR - LiNe STaRt.

Line mode byte for the cursor line.

A. + C. + D. >> CHS
B. >> don't care

#7A LNEND - LiNe END.

Low byte of the end of the cursor line. Used to check for end of line.

A. + C. + D. >> (CHS+#80) IAND #FF
B. >> don't care

#80/#81 SCREEN.

Top of the screen. Used for the cursor position and memory check during mode changing.

A. + B. + C. + D. >> #BFFF

#82/#83 SCTOP - SCreen TOP.
Pointer to the first byte (line mode byte) of the graphics, immediately after the intermediate or header. The name SCTOP is an unfortunate choice in this instance; GRS (GRaphics Start) would have been better because the pointer can be to anywhere in the screen memory. Used for COLORG and drawing commands.

A. >> don't care
B. + C. >> #BFFF-HDRL
D. >> #BFFF-HDRL-TXL*TXB-INTL

#84/#85 FFB - First Free Byte.
Pointer to the first byte after the graphics screen.

A. >> #BFFF-HDRL-TXL*TXB-TRAL
B. >> #BFFF-HDRL-GRL*GXB-TRAL
C. + D. >> #BFFF-HDRL-TXL*TXB-INTL-(GRL-GAL)*GXB-TRAL

#86/#87 GRR - GRaphics Rolled.
Pointer used in split modes. It indicates from where the screen is scrolled up to make room for the text. The portion above GRR to SCTOP is shifted to under the visible screen. Since scrolling is not possible in single (non-split) modes the pointer has little value.

A. >> don't care
B. + C. + D. >> #BFFF-HDRL-GAL*GXB or don't care

#88/#89 GRE - GRaphics End.
Points 1 byte after the end of the graphics section in use in the screen memory. It is used for the drawing functions.

A. >> don't care
B. >> #BFFF-HDRL-GRL*GXB
C. >> #BFFF-(GRL-GAL)*GXB
D. >> #BFFF-HDRL-TXL*TXB-INTL-(GRL-GAL)*GXB

#8A/#8B CHS - CHaracter Start.
Points to the first byte (line mode byte) of the text. Its use, amoung other things, is in PRINT and COLORT commands.

A. + D. >> #BFFF-HDRL
B. >> #BFFF-HDRL-GRL*GXB or don't care
C. >> #BFFF-HDRL-(GRL-GAL)*GXB-INTL

#8C/#8D GAE/CHE - Graphics Archive End/CHaracter End.
In either text or split-mode the pointer is to 1 byte after the text portion. In an unsplit mode it points 1 byte after the trailer. It is used in PRINT commands.

A. >> #BFFF-HDRL-TXL*TXB
B. >> #BFFF-HDRL-GRL*GXB-INTL
C. >> #BFFF-HDRL-(GRL-GAL)*GXB--INTL-TXL*TXB
D. >> #BFFF-HDRL-TXL*TXB

#8E/#8F SCE - SCreen End.
Points 1 byte after the end of the screen, thus immediately after the trailer. Uses : COLORT and COLORG, amoung others.

A. >> #BFFF-HDRL-TXL*TXB-TRAL
B. >> #BFFF-HDRL-GRL*GXB-TRAL
C. + D. >> #BFFF-HDRL-GRL*GXB-INTL-TXB*TXB-TRAL

#90/#91 GTE - Graphics Temporary save area End.
In a graphics mode the pointers indicate the end of the area where the scrolled section of graphics is located.

A. >> don't care
B. + C. + D. >> #BFFF-HDRL-(GRL+GAL)*GXB or don't care

#92/#93 GAS/GTS - Graphics Archive Start/Graphics Temporary save Start
In a graphics mode the pointer indicates the start of the area where the scrolled section of graphics is located.

A. >> don't care
B. >> #BFFF-HDRL-TXL*TXB-GRL*GXB or don't care
C. + D. >> #BFFF-HDRL-GRL*GXB

94/#95 GRC - GRaphics Columns.
This pointer gives the number of dots in the X direction; the value must be a minimum of 1 and a maximum of as many as stipulated in the line mode byte. For example with very high resolution at #86 bytes per line the maximum is 8*(#86/2-3)=#200=512 and for low resolution at #18 bytes per line it is 8*(#86/2)=#48=72. Used for drawing commands

A. >> don't care
B. + C. + D. >> GRC or XMAX+1

#96 GRL - GRaphics Lines.
This pointer gives the number of dots and hence the number of lines in the Y direction. Its value must be a minimum of 16 and a maximum of 256 lines. Used with drawing commands.

A. >> don't care
B. + C. + D. >> GRL = YMAX+1

#97 GAL - Graphics Archive Lines.
This pointer gives the number of lines scrolled or to be scrolled. You can use the value 0 when planning your own modes.

A. + B. + C. + D. >> don't care

#98 GXB - Graphics X Bytes.
The number of bytes per line in the mode used, graphics or split. Used with drawing commands.

A. >> don't care
B. + C. + D. >> as stipulated in the line mode bytes graphics, #86, #5A, #2E, #18 for MODE 7/8, 5/6, 3/4, 2/1 respectively

#99/#9A GREO - GRaphics End Old.
The previous end of the graphics. Need not be filled in when planning a mode.

A. + B. + C. + D. >> don't care

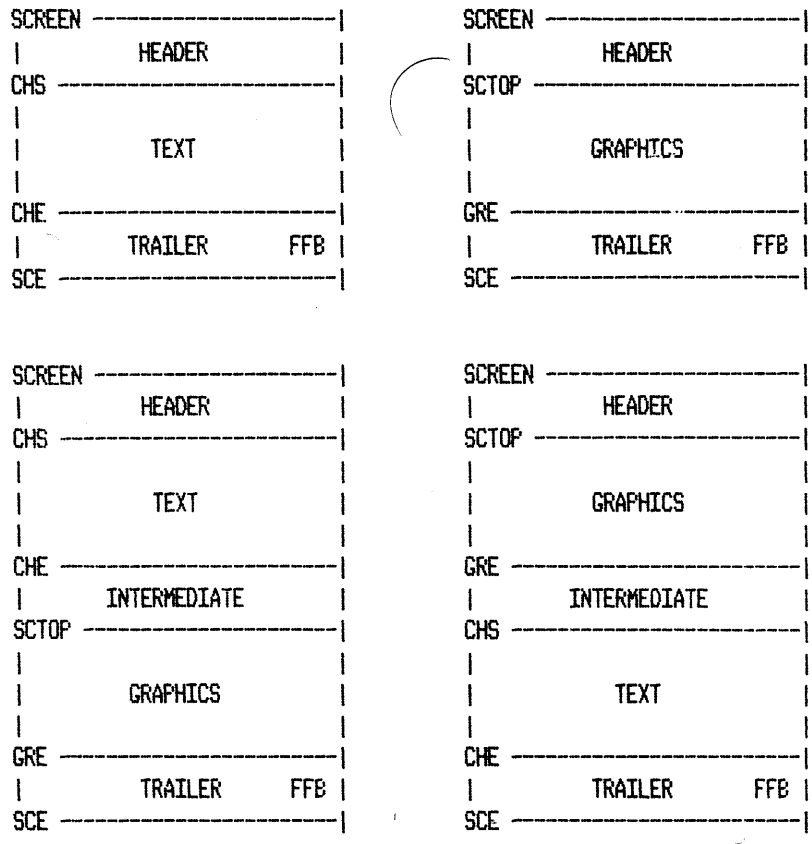
#9B/#9C CHSO - CHaracter Start Old.
The previous end of characters. Need not be filled when planning a mode.

A. + B. + C. + D. >> don't care

#2A5/#2A6 SCBOT - SCReen BOTtom.
This is where the first byte of screen memory is. Amoung other uses it is for testing for OUT OF SPACE FOR MODE.

A. >> #BFFF-HDRL-TXL*TXB-TRAL+1
B. >> #BFFF-HDRL-GRL*GXB-TRAL+1
C. >> #BFFF-HDRL-GRL*GXB-INTL-TXL*TXB-TRAL+1

Maps of the various modes with the positions of the pointers.



Now follow the values for the HEADER, INTERMEDIATE & TRAILER after a COLORT w x y z and/or COLORG w x y z. The HEADER and the INTERMEDIATE contain the colour of the section above which they stand (graphics COLORG/ text COLORT). The TRAILER contains the colour for the area below it.

```
HEADER        4 COLOURS
36 8w 00 00 36 9x 00 00 36 Ay 00 00 36 Bz 00 00
INTERMEDIATE  4 COLOURS
30 8w 00 00 30 9x 00 00 30 Ay 00 00 30 Bz 00 00
TRAILER       4 COLOURS
3F 8w 00 00 3F 9x 00 00 3F Ay 00 00 3F Bz 00 00
HEADER       16 COLOURS
B6 8w w0 FF B6 9x w0 FF B6 Ay w0 FF B6 Bz w0 FF
INTERMEDIATE 16 COLOURS
B0 8w w0 FF B0 9x w0 FF B0 Ay w0 FF B0 Bz w0 FF
TRAILER      16 COLOURS
BF 8w w0 FF BF 9x w0 FF BF Ay w0 FF BF Bz w0 FF
```

Remarks.
In a character mode the text lines will be restored to normal spacing by LIST, ?CHR$(12) and UT. In 16 colour text these commands will also reset the screen to 4 colours and when scrolling the picture the bottom-most line will be printed in 4 colour mode. The lowest line of text will always be forced to normal line spacing. From experiments it would appear that MODES 8A & 7A

will not be affected by a BREAK because the screen driver does not recognise these modes. This must be accomplished by some other artifice, namely by taking 240 line graphics and 4 line text with a line repeat count of zero. With a BREAK the screen driver then forces 1 line to a repeat count of #A. On a ?CHR$ (12) the line repeat count of all text lines is set to #A. With full 244 line graphics it is advisable to insert in a program something like '65335 IF GETC=0 THEN 65335 : MODE 0' The dots in MODE 7/8 are not square (Y=approx 1.5*X). In practice these proportions appear to be suitable for such things as drawing a circle. The disadvantage of a somewhat lower vertical resolution is compensated by the higher horizontal resolution in, for example, graphics.

With friendly greetings, N. P. Looije.

PROGRAMMING TECHNIQUES - Drawing Circles.

(from DAInamic 12, page 268)

In this article I will deal further with the task of drawing circles. Many programs need circles or ellipses and drawing them can give rise to some fair problems. There are also several ways of achieving the desired result. Firstly let me say that there is no ideal method; one way may be quick to program but slow to run, while another is difficult to program but runs faster. Still further variations exist on the need for much or little memory space. Let us assume that the circle we wish to draw will always have its centre point in the middle of the screen and have a radius of 100. The method of drawing most readily to hand is:

```
10  MODE 6
20  FOR I=0.0 TO 2.0*PI STEP PI/180.0
30  DOT XMAX/2+100*SIN(I),YMAX/2+100*COS(I) 22:NEXT
```

As we know that there are 180 degrees in PI radians we take STEPs of PI/180, or 1 degree. When the RUN command is given the resultant circle looks as if it has gaps in it. We can overcome this by making the steps smaller but at the cost of speed. If the radius is 100 then the circumference of the circle is 200*PI, which is about 628. Therefore the steps must be such that each of the dots appears only once. The following uses a step size of 2*PI/200*PI which is the same as PI/315 or 0.01, or put even more simply, 1/radius. Eventually, after a bit of rounding off, we settled for 0.9/radius. Working with a variable is faster than with a constant. Therefore:

```
10  MODE 6:FOR I=0.0 TO 2.0*PI STEP PI/315.0
20  R=100.0:DOT XMAX/2+R*SIN(I),YMAX/2+R*COS(I) 22:NEXT
```

... works faster, but each time R is made equal to 100 I lose the trifling gain I had made. More to the point, I have made the DAI call XMAX & YMAX about 600 times and each time have made it divide them by 2. The next program improves that:

```
10  MODE 6:MX=XMAX/2.0:MY=YMAX/2.0:R=100.0
20  FOR I=0 TO 2.0*PI STEP 1.0/R:DOT MX+R*SIN(I),MY+R*COS(I) 22:NEXT
```

We notice that XMAX/2.0 is shown instead of XMAX/2 as hitherto. This is because the DOT

instruction expects integers and therefore constants like 2 were not turned into 2.0. While running the XMAX/2 is changed to floating point for the calculation of R*SIN(I) and then changed back again to integer to satisfy the DOT instruction. So by putting everything in integers we can speed up the program. The first attempt at this misfired because SIN(I) and COS(I) always became zero and consequently the loop variable I also remained zero. After more thought we came up with the following program:

```
IMP INT
10  MODE 6:MX=XMAX/2:MY=YMAX/2:R!=100.0
20  FOR I!=0.0 TO PI+PI STEP 1.0/R!:X=R!*SIN(I!):Y=R!*COS(I!)
30  DOT MX+X,MY+Y 22:NEXT
```

... and this works faster. Have you spotted why? Simply that PI+PI is faster than 2.0*PI; but that is not the whole story, the combination of line 20 and line 30 also contributes. However our programme is still slow. We let sines and cosines be calculated 628 times. We can change that when we realise that the left and right parts are mirror images, and arrive at:

```
10  MODE 6:MX=XMAX/2:MY=YMAX/2:R!=100.0
20  FOR I!=0.0 TO PI STEP 1E-2:X=R!*SIN(I!):Y=R!*COS(I!)
30  DOT MX+X,MY+Y 22:DOT MX-X,MY+Y 22:NEXT
```

This seems good, it works smoothly and it is obvious that we can achieve further improvements in speed by exploiting other symmetries. In line 20's FOR statement we will only go to PI/2 instead of to PI, and line 30 becomes :- DOT MX+X,MY+Y 22:DOT MX+X,MY-Y 22:DOT MX-X,MY+Y 22:DOT MX-X,MY-Y 22:NEXT. Becoming enthusiastic now we bring into use the symmetrical diagonals too:

```
10  MODE 6:MX=XMAX/2:MY=YMAX/2:R!=100.0
20  FOR I!=0.0 TO PI/4.0 STEP 1/R!:X=R!*SIN(I!):Y=R!*COS(I!)
30  DOT MX+Y,MY+Y 22:DOT MX+X,MY-Y 22:DOT MX-X,MY+Y 22:DOT MX-X,MY-Y 22
40  DOT MX+Y,MY+X 22:DOT MX+Y,MY-X 22:DOT MX-Y,MY+X 22:DOT MX-Y,MY-X 22:
NEXT
```

... and although the program runs reasonably quickly the programming itself does involve a fair amount of effort. Now we consider if the delays associated with the sines and cosines are really necessary. We can establish that one of the two can be overcome but that again does not lead to simple programming. Let us try another way, this time with Pythagoras' theorem. Reckoning from the centre there is a point on the circle's circumference with co-ordinates x and y such that x^2 + y^2 must be equal to the radius^2. Recognising that for each x value there must be two y values we start with:

```
10  MODE 6:XM=XMAX/2:YM=YMAX/2:R=100
20  FOR X=-R TO R:Y=SQR(R*R-X*X)
30  DOT MX+X,MY+Y 22:DOT MX+X,MY-Y 22:NEXT
```

This apparently does not work correctly. On the left and right many points are missed because the x values are larger versions of the y values. Could we make use of the upper/lower symmetry as well as the left/right? No, because although it would be quite feasible we would still find the left and right points failing. We saw that the horizontal was drawn correctly, thus we could use the above method to obtain that half and achieve the other half by exchanging the x and y co-ordinates, as follows: Line 10 remains the same.

```
20  FOR I=-R/2 TO R/2:J=SQR(R*R-I*I)
30  DOT MX+I,MY+J 22:DOT MX+I,MY-J 22
40  DOT MX+J,MY+I 22:DOT MX-J,MY+I 22:NEXT
```

But this also does not work satisfactorily. Drawing half a circle does not mean from -R/2 to R/2 but from -45 to +45 degrees. So we must put the boundaries at +/-70 and not 50. The figure of 70 can be obtained (it will be needed for other circles) by dividing the radius by root 2. Again using the symmetries we now arrive at:

```
10  MODE 6: MX=XMAX/2:MY=YMAX/2:R=100
20  FOR I=0 TO R/SQR(2):J=SQR(R*R-I*I)
30  DOT MX+I,MY+J 22:DOT MX+I,MY-J 22:DOT MX-I,MY+J 22:DOT MX-I,MY-J 22
40  DOT MX+J,MY+I 22:DOT MX+J,MY-I 22:DOT MX-J,MY+I 22:DOT MX-J,MY-I 22:NEXT
```

This works fast, especially on machines without the maths chip. It is clearly an improvement but again with a ponderous program. For small circles the previous methods are just as good. Did you notice that I used R*R and not R^2? It is faster and also works for negative numbers.

If we have to draw numerous circles it would be worthwhile putting our sine and cosine values in an array so that they can be used time and time again. The advantage is that the value is available quickly for rapid multiplication with the radius and conversion into integer. When the radii are different, one smaller than 100 will give us a small time advantage and with one less than half the size we have been using we can miss some array elements and save more time. There are snags too - we have used up more memory (more than half of it is filled with zeroes), and we have to wait until the table is ready. However let us try it:

```
10  MODE 6A:CLEAR 8000:DIM X(3,159),Y(3,159)
20  MX=XMAX/2:MY=YMAX/2 R!=100.0
30  FOR A!=0.0 TO PI+PI STEP PI/316.0
40  X=R!*SIN(A!):X=MX+X:Y=R!*COS(A!):Y=Y+MY
50  X(P,Q)=X:Y(P,Q)=Y:Q=Q+1
60  IF Q=159 THEN Q=0:P=P+1:PRINT P
70  NEXT:MODE 6
80  FOR P=0 TO 3:FOR Q=0 TO 158:DOT X(P,Q),Y(P,Q) 22:NEXT:NEXT
```

Note the PRINT P, it is unnecessary and serves only as an indicator. We are using a 2-dimensional array for X and for Y. This is certainly much quicker than a 3-dimensional array. Always use integers for the indices of the arrays; look-up times will be 4 times as long with floating point. Improvements can still be made such as cutting to a quarter the calculations by mirroring. The program below does this and also works out the +/-. Study it and check the working:

```
10  MODE 6A:CLEAR 8000:DIM X(159),Y(159)
20  MX=XMAX/2:MY=YMAX/2:R!=100.0
30  FOR A!=0.0 TO PI/2.0 STEP PI/316.0
40  X(Q)=R!*SIN(A!):Y(Q)=R!*COS(A!):Q=Q+1:NEXT
50  MODE 6
60  FOR Q=0 TO 159:FOR M=-1 TO 1 STEP 2:FOR N=-1 TO 1 STEP 2
70  DOT MX+M*X(Q),MY+N*Y(Q) 22:NEXT:NEXT:NEXT
```

The next step is to build up the array by Pythagoras but that can be left to the reader. We can now think about the use of sine and cosine, or rather an approximation of them. For this refer to the article by T.Berkx in issue 10 but take note that this method is only faster for machines

without the maths chip. For that reason I have not worked it out. I also have not worked on the following idea but it should certainly be useable for small circles. Put the required values in a number of DATAs, or read them in from an array. The latter will only interest owners of MDCRs and floppy discs. Now I will draw the circle really fast, no longer doing it point by point but with little lines. In joining up the points we can make use of one of the previously discussed methods.

```
10  MODE 6:MX=XMAX/2:MY=YMAX/2:R!=100.0:XO=MX:YO=MY+100
20  FOR I!=0.0 TO PI+PI STEP PI/15.0:X=R!*SIN(I!):Y=R!*COS(I!)
30  DRAW XO,YO MX+X,MY+Y 22:XO=MX+X:YO=MY+Y:NEXT
```

If you find that the circle is too angular the STEP needs to be altered. It may be necessary to go rather more than right round otherwise the final dash may not be drawn in. This method can be combined with the one which reads from an array and thus we get a fast running but awkward to write program.

```
10  MODE 6:CLEAR 2000:MX=XMAX/2:MY=YMAX/2:R!=100.0:DIM X(30),Y(30)
20  FOR A!=0.0 TO 2.06 STEP PI/15.0
30  X=R!*SIN(A!):X(P)=MX+X:Y=R!*COS(A!):Y(P)=MY+Y:P=P+1:NEXT
40  FOR I=0 TO 29:DRAW X(I),Y(I) X(I+1),Y(I+1) 22:NEXT
```

Even in this program some minor items can be trimmed. The 2 variables X and Y can be replaced by one. In place of the new variable I we can use P - it worsens the readability and does nothing for the speed but we gain a little in the memory (8 bytes). Combining lines 20 and 30 will also change the speed. We have only been discussing circles having a radius of 100 but small circles can sometimes be handled better with one of our earlier methods. I must also mention that if we want the circle sectioned we should avoid drawing a bundle of radii from the centre to the rim because through rounding off, some of the points will be lost, at least with larger circles. An obvious remedy is to reduce the size of the steps or even better, to draw diameters instead of radii. Such lines should be drawn horizontally, not vertically because of the DAI's screen handling system. I will leave the reader to have a go at programming this.

Frank H. Druijff

CONVERSION APPLE - ATARI - DAI

(from DAInamic 12, p289)

The conversion of Apple and Atari programs for DAI gives rise to various problems, among which is that counting in their graphics modes starts from the top left corner of the screen. Thus conversion, especially for long programs, involves extensive calculations with consequently more chance of error. An example :-

```
Apple                   1st solution          2nd solution          3rd solution
                                                                     5  POKE #2FF,179: POKE #6C,0: POKE #6D,3
10  HGR                 MODE 6                MODE 6                10  MODE 6
20  Y=0:COLOR=15        Y=179:C=15            Y=0:C=15              20  Y=0:C=15
30  PLOT 100,Y          DOT 100,Y C           DOT 100,179-Y C       30  DOT 100,Y C
40  Y=Y+1               Y=Y-1                 Y=Y+1                 40  Y=Y+1
50  IF Y>179 THEN END   IF Y<0 THEN END       IF Y>179 THEN END     50  IF Y>179 THEN END
60  GOTO 30             GOTO 30               GOTO 30               60  GOTO 30
```

The advantage of the third solution is that after the initial line 5, the Apple or Atari program can be input almost literally. The working is as follows. A machine language program does for solution 3 what solution 2 achieves in BASIC, namely the subtraction of each Y value from a previously specified value (179 in this example). This is possible because each BASIC command associated with the screen RAM makes use of RST + byte combination. RST 5 stands in for a CALL #28, and at #28 is the interrupt routine for vector 5 (see also DAInamic 11, pages 180-181). Interrupt routine 5 accomplishes a jump in accordance with the contents of locations #6C and #6D. If locations #6C and #6D contain #300 then on every screen command there will be a jump to #300, with the required parameters being in the registers. (see also DAInamic 5, page 120). From #300 onwards a part of the ROM is copied so that when a screen function has been executed there will be a jump back to the ROM. #313 looks for a DOT, DRAW, FILL or SCRN function. If there is none then it jumps back to the ROM, but if there is one then Y will be deducted from the contents of #2FF. Then follows the jump back to ROM and BASIC is none the wiser.

```
#300  E1       POP H
#301  F3       DI
#302  224300   SHLD #43     ;SAVE HL
#305  F5       PUSH PSW
#306  3E80     MVI A,#80    ;BANK SELECT BITS
#308  E1       POP H
#309  224100   SHLD #41     ;SAVE PSW
#30C  67       MOV H,A
#30D  AF       XRA A        ;CLEAR A
#30E  E3       XTHL
#30F  86       ADD M        ;ENTRY NUMBER IN ACCU
#310  23       INX H
#311  E3       XTHL
#312  6F       MOV L,A      ;COMPLETE ENTRYPOINT ADDRESS
#313  FE1E     CPI #1E      ;SEE IF ENTRY NUMBER
#315  DAD4C6   JC #C6D4     ;IS EQUAL OR BETWEEN
#318  FE2A     CPI #2A      ;#1E AND #27
#31A  D2D4C6   JNC #C6D4
#31D  3AFF02   LDA #2FF     ;Y1:=(#2FF)-Y1
#320  90       SUB B
#321  47       MOV B,A
#322  3AFF02   LDA #2FF     ;Y2:=(#2FF)-Y2
#325  91       SUB C
#326  4F       MOV C,A
#327  C3D4C6   JMP #C6D4

#300  E1 F3 22 43 00 F5 3E 80 E1 22 41 00 67 AF E3 86
#310  23 E3 6F FE 1E DA D4 C6 FE 2A D2 D4 C6 3A FF 02
#320  90 47 3A FF 02 91 4F C3 D4 C6
```

The clever ones who have followed this up to now will have noticed that for DOT and SCRN functions only one Y value is given in the BASIC although in the machine language there are always two subtractions. How does that happen? Indeed I do not know, but it appears to work all right in BASIC. When one has input the ML program the BASIC pointers have to be adapted: (UT S29B EC-30 02-3) then go back to BASIC and give a NEW command or clear 256. The ML program can be switched in with: POKE #6C,0: POKE #6D,3 and switched out again with: POKE #6C,#FD: POKE #6D,#C6. The previously dimensioned Y value is poked into #2FF. For completeness, the YMAXs of the Apple in the different graphic modes are: GR = 39 & HGR = 179.

In conclusion, if one can fathom out the workings a lot of pleasing effects can be achieved, with minor extensions such as moving drawings, reflections (mirroring) etc. One can also write a similar program for the CURSOR statement.

With best wishes, Frans Verstegen.

TV TENNIS REMs

(from DAInamic 12, page 278)

```
2       REM See the DAInamic original for rest of program.
10      REM initialise
49      REM Drawing the bats and erasing the earlier ones
99      REM Drawing the ball and erasing the old one
125     IF BX1<5.0 THEN GOSUB 1000:REM Ball at height of red bat
126     IF BX1>XMAX-5.0 THEN GOSUB 1200:REM Ball at height of blue bat
129     REM Ball at end of court
139     REM Ball at the edge of the court
150     BX1=BX1+SNX:BY1=BY1+SNY:REM Calculation of the new position for the ball
1021    IF BEW=BAT1 THEN RETURN:REM Effective hit or not
1400    SOUND 0 0 15 0 FREQ(35.0):WAIT TIME 10:SOUND OFF :BER$="RED missed..."
1600    SOUND 0 0 15 0 FREQ(50.0):WAIT TIME 10:SOUND OFF :BER$="BLUE missed..."
2002    REM Initialise
2015    REM START PAGE
2110    CURSOR 30,3:PRINT "by Luc Maes"
2398    REM PAGE 1:Choice of the level of difficulty
2410    CURSOR 10,20:PRINT "Which level of difficulty do you want ?"
2420    CURSOR 20,14:PRINT "1: easy (large racquets)"
2430    CURSOR 20,13:PRINT "2: hard (small racquets)"
2435    CURSOR 2,6:PRINT "On court: 'EVENT' to serve":CURSOR 12,5:PRINT "'M' for the
alternative level of difficulty"
2436    CURSOR 12,4:PRINT "'CHAR DEL' to stop play"
2500    REM Reckoning  the  score
2570    IF GR=40 AND J$="advantage RED" THEN J$="":GOTO 2800
2580    IF GR=40 AND J$="" THEN J$="advantage BLUE"
2710    IF GB=40.0 AND J$="advantage BLUE" THEN J$="":GOTO 2800
2720    IF GB=40.0 AND J$="" THEN J$="advantage RED":GOTO 2800
2800    REM PAGE 3:court + scoreboard
2840    CURSOR 8,3:PRINT "Red's score"
2860    CURSOR 39,3:PRINT "Blue's score"
2950    IF (SB+SR) MOD 2.0=0.0 THEN BX1=7.0:SNX=3.0:CURSOR 0,0:PRINT "RED
serves.";:GOTO 2990
2960    BX1=XMAX-7.0:SNX=-3.0:CURSOR 40,0:PRINT "BLUE serves.";
3005    IF (SB+SR) MOD 2=0 THEN BY1=3.0+PDL(5)*Q:GOTO 3050
4999    REM End of the game
5020    CURSOR 10,20:PRINT "'RETURN' to play again.":CURSOR 10,17:PRINT "'CHAR DEL' to
finish."
5030    CURSOR 10,14:PRINT "'L' to load the next program"
```

A. ALGEMEENHEDEN

HOOFDSTUK I : INLEIDENDE BEGRIPPEN

1.1. Situering van microprocessoren in de digitale techniek

Met de opkomst van de microprocessoren ontstond, naast de klassieke combinatorische en sequentiële logica, een belangrijke tak die micro-elektronica wordt genoemd.
Behalve het feit dat het aantal componenten per chip en dus de gecompliceerdheid per IC sterk is toegenomen, dienen we de schakelingen te benaderen met een nieuwe filosophie.
Met de klassieke logica was alles erop gericht om, vertrekkende van het 'lastenboek' van de toepassing een schema te ontwikkelen, waarbij volgende stadia worden doorlopen :

- Opstellen van de vergelijking(en) van de gewenste toepassing.
- Vereenvoudiging van deze vergelijking(en) met behulp van klassieke methoden zoals Karnaugh, Quine en Mc Cluskey.
- Opstellen van het schema aan de hand van de vereenvoudigde vergelijking(en).

Deze denkwijze dient op het gebied van de micro-elektronica grondig herzien te worden. Inderdaad, het komt er niet meer op aan het schema samen te stellen, maar vertrekkend van een standaardschema, of beter gezegd van standaard materiaal, (een microcomputer) de gewenste toepassing te 'programmeren'.
We dienen dus te spreken van programmeerbare logica of programmeerbare systemen. Het standaard materiaal is in feite een microcomputer waarvan de microprocessor zelf meestal slechts de centrale verwerkingseenheid vormt. We kunnen dus de microprocessor definiëren als een programmeerbare logische LSI bouwsteen.
In plaats van te spreken van logische schakelingen spreken we van logische systemen ; hiermee bedoelen we dat we niet één schakeling gebruiken maar dat meerdere schakelingen samengevoegd worden tot een systeem als fundamenteel schema.

1.2. Architectuur van een microprocessorsysteem.

Het principiële schema van een microprocessorsysteem herleidt zich tot het basisschema van een computer of beter gezegd een microcomputer (fig. 1.1).

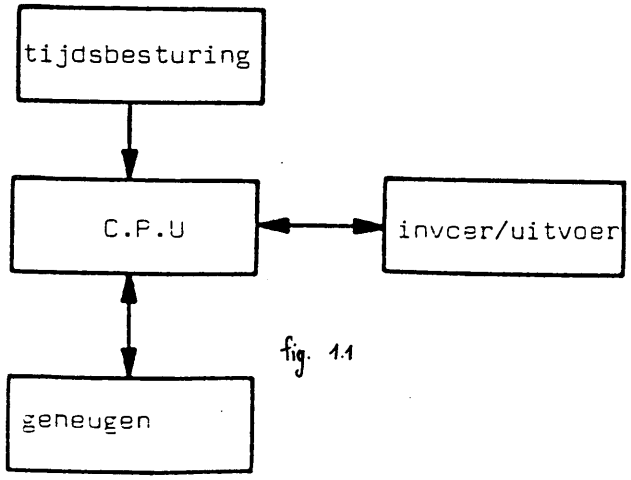


fig. 1.1

De samenstellende delen van dit schema zijn :

- de centrale verwerkingseenheid (microprocessor) (CPU : *central processing unit*).
  Deze schakeling voert alle logische en rekenkundige bewerkingen uit en is tevens verantwoordelijk voor de besturing en de controle van operaties in het ganse circuit.
  De centrale verwerkingseenheid wordt zelf gesynchroniseerd vanuit de klok of tijdsbesturing.
- De invoer- uitvoer schakelingen maken het mogelijk informatie te ontvangen of te sturen naar de randapparaten.
- Het geheugen laat toe programma's en of gegevens op te slaan voor verdere verwerking.

Opdat de centrale verwerkingseenheid de aangegeven functies naar behoren zou kunnen vervullen, moet de microprocessor geheugen of invoer/uitvoer poorten kunnen adresseren en informatie (gegevens, data) kunnen ontvangen, verwerken of doorzenden.
De verbinding van de microprocessor met de andere schakelingen gebeurt derhalve meestal met zogenaamde bussen.
Een bus is een meeraderige verbinding met een aantal draden bepaald door de soort informatie dat overgedragen wordt.
De klassiek gebruikte bussen zijn :

- Een data of gegevensbus waarmede informatie (data) van- en naar de microprocessor gebracht wordt.
  Bij een 8 bits microprocessor (die met een woordlengte van 8 bits of 1 byte werkt) is de breedte van de gegevensbus 8 bits; ze bestaat dus uit 8 parallelle geleiders.
- Een adresbus via dewelke de microprocessor het geheugen of de invoer/uitvoer poorten kan adresseren.
  De breedte van de adresbus is meestal 16 bits, dit in verband met het aantal te adresseren geheugencellen. Met 16 bits is het inderdaad mogelijk 2^{16} of 65536 (64K) geheugencellen te adresseren.
- Een controlebus of stuurbus.
  Via de signalen op de controlebus kan de microprocessor het systeem besturen, d.w.z. de gewenste doorschakelingen tussen de verschillende samenstellende schakelingen zo regelen dat de uitvoering van een bepaalde programma opdracht of van een gedeelte ervan mogelijk gemaakt wordt.

De uitbouw van het elementair microcomputer blokschema van fig. 1.1 tot een blokschema waar gebruik gemaakt wordt van busstructuur, brengt ons tot fig. 1.2.

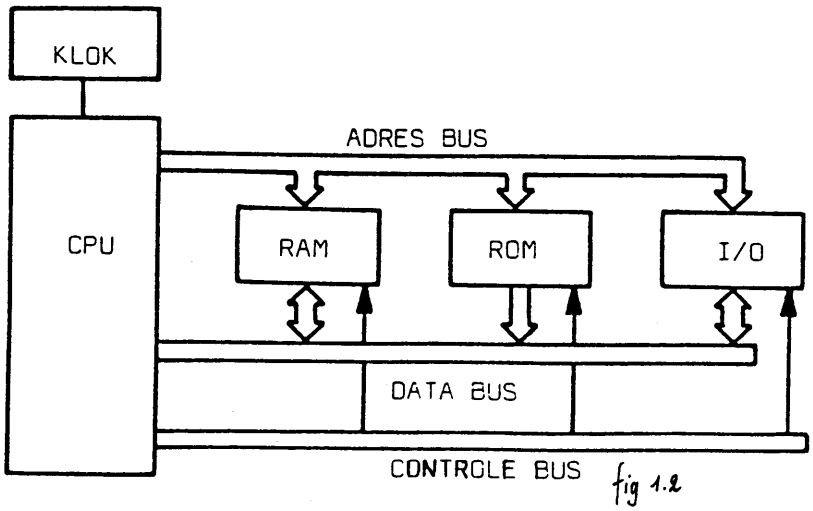


fig 1.2

In dit schema is het geheugen van de microcomputer tevens onderverdeeld in twee types :

- RAM geheugen (*random access memory*)
  Het is een geheugen waarvan elke cel willekeurig toegankelijk is en dat gebruikt wordt om veranderlijke gegevens en tussenresultaten in te schrijven of uit te lezen.
- ROM geheugen (*read only memory*).
  Dit is een geheugen waaruit enkel kan gelezen worden en dat meestal het programmageheugen is, d.w.z. het geheugen waarin het uit te voeren programma van een microprocessortoepassing is ondergebracht. Elk van deze geheugenblokken kan daarenboven nog uit verschillende modulen bestaan.

Als we tenslotte nog rekening houden met de speciale bouwstenen voor de in- en uitvoer, die zowel in serie als in parallel kan gebeuren, kan het blokschema uitgebreid worden tot dit van fig. 3.

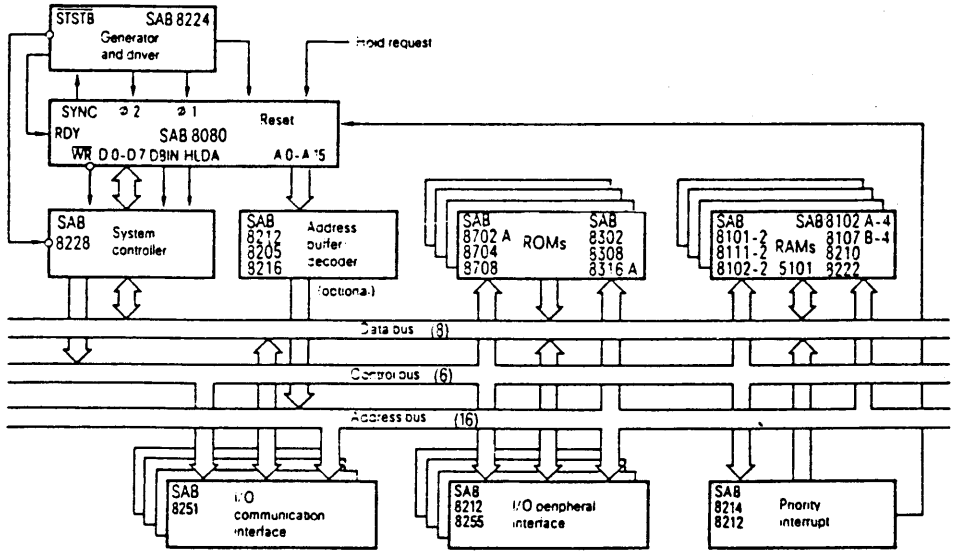


fig. 1.3

1.3. Inwendige architectuur van een microprocessor

Ook inwendig in de CPU gebeurt de verbinding van de verschillende samenstellende delen met behulp van een busstructuur. Het essentieelste element in de microprocessor is de ALU (rekenkundige en logische eenheid : *arithmetic and logical unit*) waarrond de microprocessor als het ware is opgebouwd. Het verschil tussen de individuele microprocessor types wordt o.a. bepaald door de inwendig gebruikte busstructuur.

1.3.1. Enkel busstructuur

Alle inwendige registers alsmede de ALU zijn verbonden met de enige inwendige databus (fig. 1.4). De ingangen van de ALU waaraan de twee te verwerken gegevens dienen aangelegd te worden, zijn uitgerust met een bufferregister.

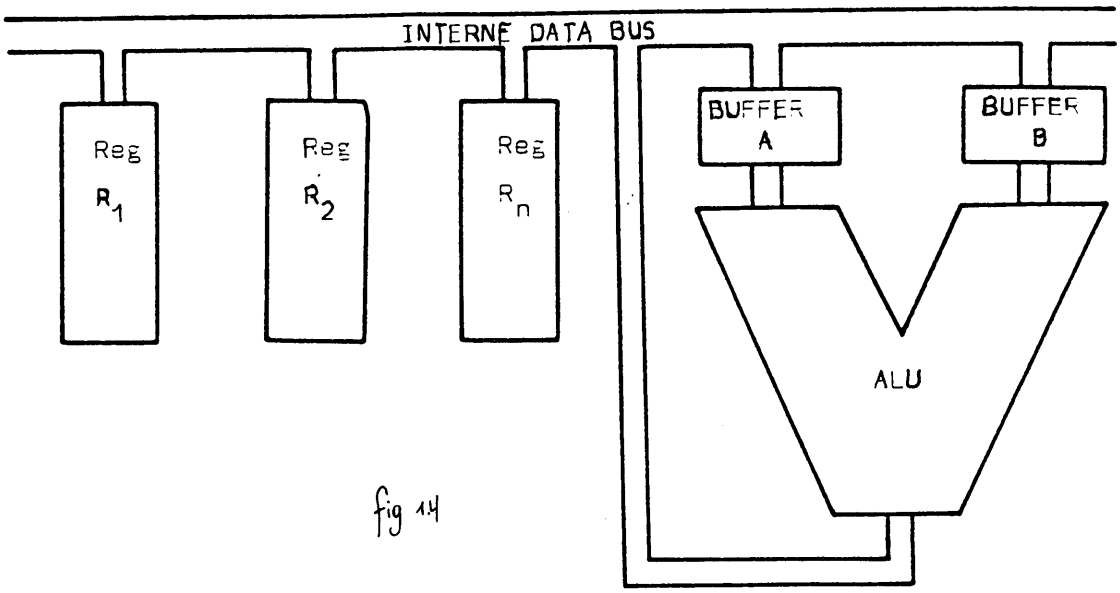


fig 14

Om een inzicht te verkrijgen in de werking van zulk systeem veronderstellen we dat de inhoud van R_1 en R_2 dient opgeteld te worden en dat het resultaat in R_1 dient ingeschreven te worden. De volgorde van de bewerkingen is :
Inhoud reg 1 in buffer A
Inhoud reg 2 in buffer B
Optellen van inhoud van buffer A en buffer B : resultaat naar reg 1.
Het register waarin het resultaat wordt ingeschreven (reg 1) en dat ook één van de op te tellen grootheden bevatte, noemt men de accumulator.
Voor alle operaties die dienen uitgevoerd te worden met de ALU dient een van de operanden ingeschreven te worden in de accumulator. Het resultaat wordt ook steeds en dit automatisch in de accumulator ingeschreven. Het volstaat dus in fig. 1.4 één van de registers R_1,R_n te voorzien voor deze functie en dus de naam accumulator te geven.
Bij al deze bewerkingen verloopt het transfert van de informatie via de inwendige bus die de naam draagt van inwendige databus.
Het voordeel van deze enkele busstructuur ligt in de plaatswinst op de geïntegreerde schakeling (er moet slechts één bus geïntegreerd worden). Het is ook daarom dat de meeste microprocessoren volgens deze architectuur samengesteld zijn. Het nadeel ligt in de snelheid van uitvoering van de bewerkingen. Inderdaad dienden drie transferten plaats te vinden om een optelling te kunnen uitvoeren.

1.3.2. Multibus architectuur

Fig. 1.5 geeft een voorbeeld van een 3 bus architectuur voor een microprocessor. Elk van de ingangen van de ALU alsmede de uitgang zijn aangesloten op één bus.
Bij dezelfde optelling als hoger, gebeuren de bewerkingen nu als volgt.
De inhoud van register 1 wordt op ingang A gebracht via bus A.
Tegelijkertijd wordt de inhoud van register 2 op ingang B gebracht via bus B.
Het resultaat van de optelling wordt ingeschreven in register R_1 via de resultaatbus, vanuit de uitgangsbuffer van de ALU.
Het is duidelijk dat hierdoor de verwerkingssnelheid hoger is.
Deze architectuur wordt slechts zeer uitzonderlijk gebruikt bij 8 bits microprocessoren maar meer bij de 16 bits types.

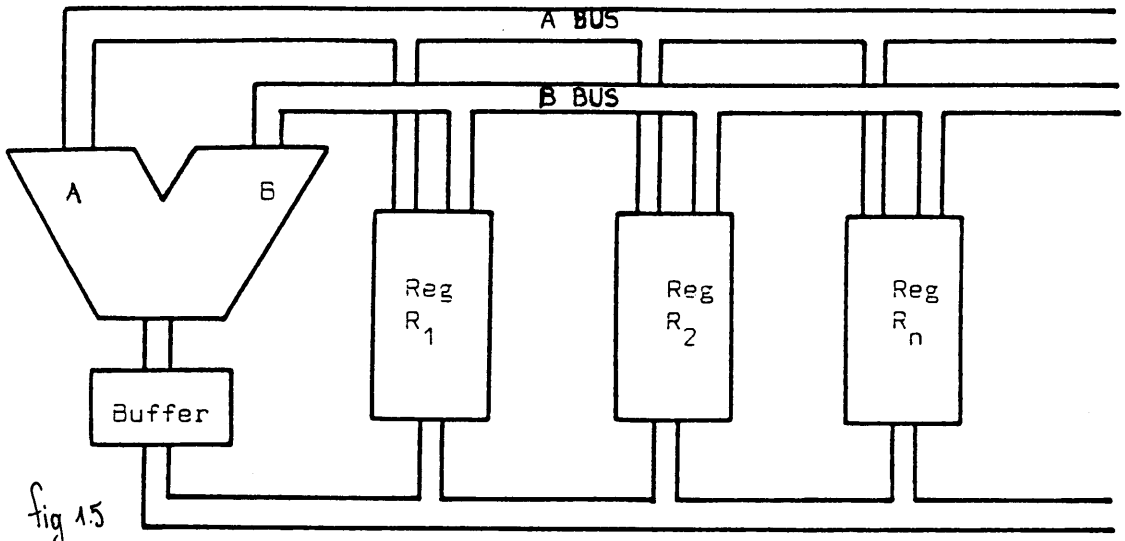


fig 1.5

1.4. Indelingscriteria voor microprocessoren

De indeling van microprocessoren kan gebeuren volgens verschillende criteria zoals :

- de familie bepaald door de constructeur waartoe de microprocessor behoort.
- de lengte of het aantal bits waaruit een datawoord bestaat.
- het aantal IC's dat dient gebruikt te worden om een microcomputer samen te stellen.

1.4.1. Indeling volgens de constructeur

Alhoewel de microprocessoren nog vrij jong zijn (de eerste werd geïntroduceerd in 1971) kennen we reeds een zeer grote verscheidenheid aan constructeurs en dus aan types.
In annex 1 wordt een tebel gegeven met een overzicht van de constructeurs en de door hen gefabriceerde microprocessoren. Er zijn echter slechts een viertal types die voldoende universeel gebruikt worden om als industriële standaard in aanmerking te komen. Het betreft :

- de familie van de 8080, 8085, 8086 van de firma INTEL met belangrijke second source producenten, waaronder SIEMENS.
- de familie van de 6800, 68000 van de firma MOTOROLA
- de familie van de Z80, Z8000 van de firma ZILOG
- de familie 650X van de firma MOSTEK

In dit boek bespreken we deze vier types, maar wat betreft programmatie en schakelschema's zal de nadruk gelegd worden op de familie van INTEL-SIEMENS.

1.4.2. Indeling volgens woordlengten

Volgens woordlengte kennen we de meest uiteenlopende cijfers, gaande van de 1 bit processor van MOTOROLA tot 32 bits processors zoals IAPX32 van INTEL met als tussenstadia 4, 8 en 16 bits.
De 8 bits en 16 bits processoren zijn tot op heden de meest gebruikte en net zijn deze die we in het bijzonder bestuderen.

Merken we op dat de woordlengte niet mag verward worden met de breedte van de adresbus. Inderdaad, de meeste 8 bits processoren (die dus met 8 bits data werken) maken gebruik van een 16 bits adresbus om voldoende adresseringscapaciteit te hebben. Met een 16 bits adresbus is het inderdaad mogelijk 65536 geheugencellen te adresseren.
De 16 bits processoren hebben bv. een adresbus van 20 bits (INTEL 8086), of 23 bits (ZILOG Z8000). Hiermede kunnen, gebruik makend van de zogenaamde segmenteringsadressering, respectievelijk 1MByte of 8 MByte geadresseerd worden. (zie verder).

1.4.3. Het aantal IC's dat dient gebruikt te worden

Bij de verdere evolutie van de integratietechnieken is het mogelijk geworden microprocessoren, geheugens en I/O poorten onder te brengen op één chip. Derhalve is men gaan spreken van multi-chip en één-chip microcomputers. In annex 2 is een overzicht gegeven van de verschillende types van elke constructeur met vermelding van de functies die ze vervullen.

1.5. Adresseringsmethodes

Vermits een microprocessor een programmeerbare LSI bouwsteen is, zal zijn taak er altijd in bestaan een programma uit te voeren.
Een programma bestaat, zoals we verder in detail bestuderen, uit een reeks programma opdrachten of instructies die in niet vluchtige geheugens ingeschreven zijn. Deze instructies dienen stuk voor stuk opgeroepen te worden om ze te kunnen uitvoeren.
Om een instructie op te roepen wordt een geheugencel geadresseerd waarna de inhoud van deze cel (instructie) naar de microprocessor gebracht wordt om daar behandeld te worden.
Het adresseren kan op verschillende wijzen gebeuren.
Gegevens zowel als adressen worden voorgesteld onder hexadecimale vorm.
Vermits een adres bestaat uit 16 bits kan het voorgesteld worden door een hexadecimaal getal van 4 digits. (elk hexadecimaal digit stelt immers 4 bits voor).
De totale adresomvang gaat bij een 16 bits adres van 0000 Hex tot FFFF Hex (fig. 1.6).

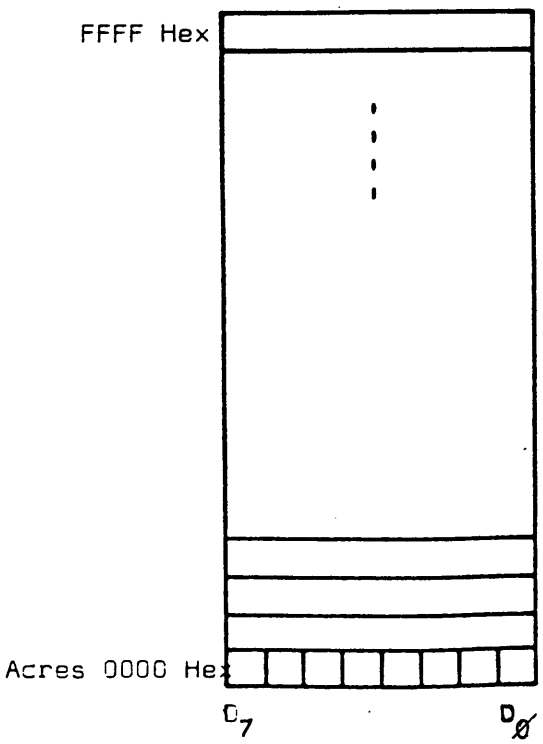


fig. 1.6

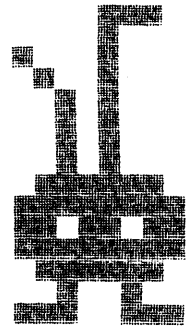
program identification

title	:	SHAPE-DESIGN (for FFGT & machine language prgs)
author	:	W.Hermans
purpose	:	to create objects in a very easy way
comment	:	we will extend the program in next issue ...

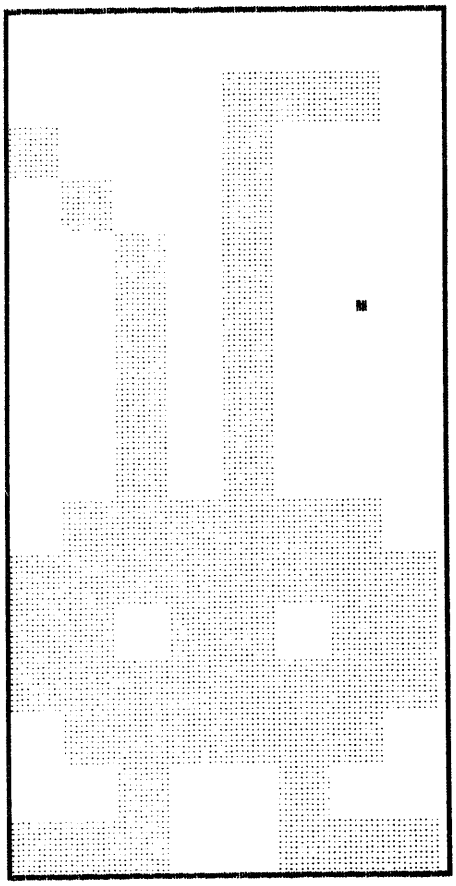
```
1     POKE #131,1:REM --- SCREEN ONLY ---
5     CLEAR 3000
10    MODE 6:COLORG 0 5 10 15
20    DIM A(8,16):REM --- ARRAY FOR PIXEL INFORMATION ---
30    FILL 9,9 91,171 21:FILL 10,10 90,170 20
35    FILL 200,100 217,133 20:FILL 120,100 153,165 20
37    FILL 250,100 259,117 20
40    X=1:Y=1:REM --- CURSOR START ---
100   IF GETC<>0 THEN 100
110   G=GETC:IF G=0 THEN 110
120   IF G=ASC("C") THEN 20:REM --- CLEAR ALL ---
125   IF G=13 THEN GOSUB 1000:REM --- CLEAR PIXEL ---
130   IF G=32 THEN GOSUB 2000:REM --- SET PIXEL ---
132   IF G=ASC("F") THEN GOSUB 6000:REM --- FILL ALL ---
135   IF G=9 THEN GOSUB 3000:REM --- PRINTOUT ---
140   IF G>=16 AND G<=19 THEN GOSUB 900
150   FILL X*10+5,Y*10+5 X*10+6,Y*10+6 23
160   GOTO 100:REM NEXT ACTION
200   IF Y=16 THEN RETURN
210   Y=Y+1:RETURN
300   IF Y=1 THEN RETURN
310   Y=Y-1:RETURN
400   IF X=1 THEN RETURN
410   X=X-1:RETURN
500   IF X=8 THEN RETURN
510   X=X+1:RETURN
900   COL=20:IF A(X,Y)=1 THEN COL=22
905   FILL X*10+5,Y*10+5 X*10+6,Y*10+6 COL:REM --- CLEAR OLD CURSOR POSITION ---
910   ON G-15 GOSUB 200,300,400,500
920   RETURN
1000  FILL X*10,Y*10 X*10+9,Y*10+9 20
1010  FILL 200+X*2,100+Y*2 200+X*2+1,100+Y*2+1 20
1012  FILL 120+X*4,100+Y*4 120+X*4+3,100+Y*4+1 20
1015  DOT 250+X,100+Y 20
1020  A(X,Y)=0:RETURN
2000  FILL X*10,Y*10 X*10+9,Y*10+9 22
2010  FILL 200+X*2,100+Y*2 200+X*2+1,100+Y*2+1 23
2012  FILL 120+X*4,100+Y*4 120+X*4+3,100+Y*4+3 23
2015  DOT 250+X,100+Y 23
2020  A(X,Y)=1:RETURN
3000  REM HEX-CONVERSION
3002  MODE 6A:INPUT " NAME ";N$
3005  POKE #131,0:PRINT TAB(20);N$:PRINT :PRINT CHR$(27);"A";CHR$(8)
3010  FOR Y=16 TO 1 STEP -1
3020  V=1:T=0
3030  FOR X=8 TO 1 STEP -1
3040  IF A(X,Y)=1 THEN T=T+V
3050  V=V SHL 1
3060  NEXT
```

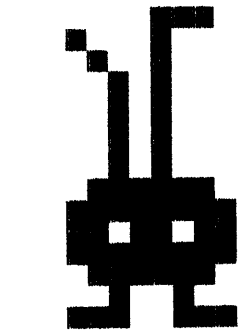
SCOOBY

```
# 0
# E
# 0
#88
#48
#28
#28
#28
#28
#7E
#FF
#D8
#FF
#7E
#24
#E7
```

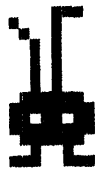


```
3070  PRINT "#";:IF LEN(HEX$(T))=1 THEN PRINT " ";
3071  PRINT HEX$(T);TAB(10);
3072  D$=""
3075  FOR P=1 TO 8
3077  IF A(P,Y)=1 THEN GOSUB 4000:REM --- PIXEL ON ---
3078  IF A(P,Y)=0 THEN GOSUB 5000:REM --- PIXEL OFF ---
3080  NEXT:PRINT CHR$(27);"K";CHR$(64);CHR$(0);D$:REM --- BIT GRAPHICS ON EPSON ---
3085  NEXT:PRINT :PRINT
3090  POKE #131,1
3100  IF GETC<>0 THEN 3100
3110  IF GETC=0 THEN 3110
3120  X=1:Y=1:MODE 6:RETURN
4000  FOR D=1 TO 8:D$=D$+CHR$(#FF):NEXT
4020  RETURN
5000  FOR D=1 TO 8:D$=D$+CHR$(0):NEXT
5020  RETURN
6000  FILL 10,10 90,170 22:FILL 200,100 216,132 22
6010  FILL 250,100 258,116 22
6020  FOR M=1 TO 8:FOR N=1 TO 16:A(M,N)=1:NEXT:NEXT:REM --- ALL PIXELS ON ---
6030  RETURN
```





LOW-RES



MED.-RES

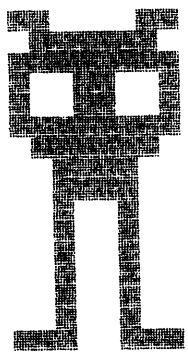
HIGH-RES

THE COMMANDS :

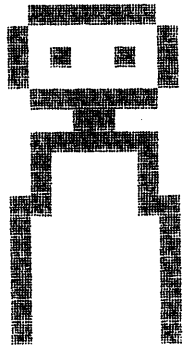
---

* CURSOR-KEYS : MOVE CURSOR
* SPACE-BAR : SET PIXEL
* RETURN : CLEAR PIXEL
* TAB : PRINT
* C : CLEAR ALL PIXELS
* F : SET ALL PIXELS

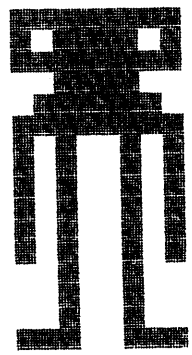
```
#03
#7E
#FF
#99
#99
#FF
#7E
#3C
#3C
#24
#24
#24
#24
#24
#24
#E7
```



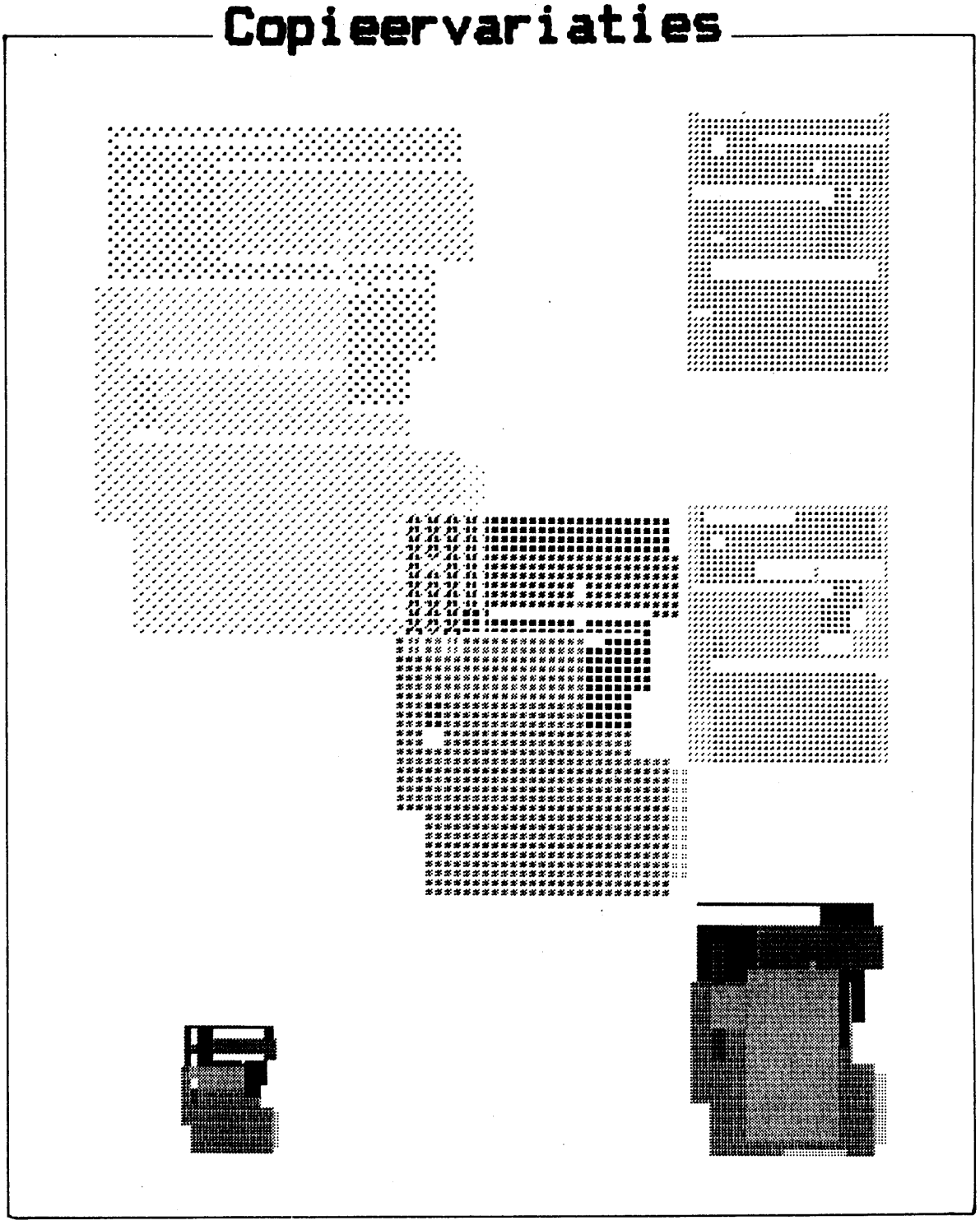
```
#7E
#81
#A5
#81
#7E
#18
#7E
#42
#42
#C3
#91
#81
#81
#81
#81
#81
```



```
#FF
#BD
#FF
#3C
#7E
#FF
#A5
#A5
#A5
#A5
#A5
#A5
#24
#24
#24
#E7
```



```
10    REM BRAM VINGERLING / COPIEERVARIATIES
100   MODE 0:COLORG 0 5 10 15:MODE 5
110   XM2=XMAX/2:YM2=YMAX/2:XM4=XMAX/4:YM4=YMAX/4
120   FOR I=1 TO 20:FILL RND(XM4),RND(YM4) RND(XM4),RND(YM4) RND(16):NEXT
130   FOR X=0 TO XM4 STEP 2:X2=2*X:X3=X*3/2
140   FOR Y=0 TO YM4 STEP 2:Y2=2*Y:Y3=Y*3/2:C=SCRN(X,Y)
150   DOT X/2,YMAX-YM4+Y/2 C:C2=C/2:C3=C*3/4:DOT XMAX-XM4+X,Y C2
160   FILL XM4+X3,YM4+Y3 XM4+X3+1,YM4+Y3+1 C:DOT XM2+X2,YM2+Y2 C
170   DOT XM2-XM4/2+X,Y C3:DOT XM2+X2+2,YM2+Y2+2 C:NEXT:NEXT
180   GOTO 120
```



alle adressen in HEXvorm!

0	zwart
1	blauw
2	d.rood
3	rood
4	paars
5	groen
6	d.bruin
7	l.bruin
8	grijs
9	blauw
10	oranje
11	rose
12	l.blauw
13	l.groen
14	geel
15	wit

29B-29C	start heap
29D-29E	size heap
29F-2A0	start text buffer
2A1-2A2	start symbol table
2A3-2A4	end of symbol table
2A5-2A6	bottom screen ram

75	cursor symbol
74	cursor mode
72-73	cursor position

40,28 cass motor 1 ON
40,18 cass motor 2 ON
40,30 1 and 2 OFF

131,0	output scrn+ RS232
131,1	screen only
131,2	edit buffer
135,2	read from edit buffer

MODE	XMAX	YMAX
1/2	71	64
3/4	159	129
5/6	335	255

MERGE
°CLEAR XXX
°LOAD"A"
°EDIT BREAK/BREAK
°LOAD"B"
°POKE 135,2

IMP INT *** IMP FPT
°IMP FPT
°CLEAR XXXX
°EDIT BREAK/BREAK
°IMP INT
°POKE 135,2

COLORG R1 R2 R3 R4
20 21 22 23

16	:R2*R1 R4*R3	
17	:R1*R2 R3*R4	32K 7XXX
18	:R3*R1 R4*R2	12K 2XXX
19	:R1*R3 R2*R4	8K 1XXX

LIJN	CTRL	COLOR	LIJN	CTRL	COLOR
23	BFEF	BFEE	11	B9A7	B9A6
22	BF69	BF68	10	B921	B920
21	BEE3	BEE2	9	B89B	B89A
20	BE5D	BE5C	8	B815	B814
19	BDD7	BDD6	7	B78F	B78E
18	BD51	BD50	6	B709	B708
17	BCCB	BCCA	5	B683	B682
16	BC45	BC44	4	B5FD	B5FC
15	BBBF	BBBE	3	B577	B576
14	BB39	BB38	2	B4F1	B4F0
13	BAB3	BAB2	1	B46B	B46A
12	BA2D	BA2C	0	B3E5	B3E4

CTRL&COLOR BYTES IN A-MODE

MODE	CTRL	COLOR	LIJN
1A/2A	BAE7	BAE6	3
	BA61	BA60	2
	B9DB	B9DA	1
	B955	B954	0
3A/4A	ACD3	ACD2	3
	AC4D	AC4C	2
	ABC7	ABC6	1
	AB41	AB40	0
5A/6A	7557	7556	3
	74D1	74D0	2
	744B	744A	1
	73C5	73C4	0

FD00 b2 page signal
b3 serial out rdy
b4 right paddle
b5 left paddle
b6 random data
b7 cass. input
FD01 Trigger paddle
FD04 0-3 volume ch.1(0)
4-7 volume ch.2(1)
FD05 0-3 volume ch.3(2)
4-7 volume noise
FD06 b0 cass.out
b1/2 paddle select
b3 paddle enable
b4 cass motor 1
b5 cass motor 2
b6/7 ROM BANK SWITCH

FF00 ser.inp.buff
FF01 b0-6 keyb.inp.
b7 in7 DCE
FF02 Interr.reg.
FF03 b1 frame error
b2 overrun error
b3 rec.buf.loaded
b4 trans.buf.empty
FF04 COMMAND REGISTER
FF05 BAUD RATE REGISTER
FF06 ser.out buf.
FF07 keyb.output
FF08 interr.mask reg.

TEST EVENT
PEEK(éFD00) IAND 32
PEEK(éFD00) IAND 16
PEEK(éFD00) IAND 48

FF09 TIMER 0
FF0A TIMER 1
FF0B TIMER 2
FF0C TIMER 3
FF0D TIMER 4

8253
CH 0 FC00/FC01
CH 1 FC02/FC03
CH 2 FC04/FC05
STATUS FC06/FC07