Koch

ATARI peeks en pokes

ATARI Bibliotheek 1

DATA BECKER NEDERLANDS

ATARI peekes en pokes

Koch

ATARI peekes en pokes

ATARI Bibliotheek 1

DATA BECKER
NEDERLANDS*

Oorspronkelijke titel: Peeks & Pokes zu Atari 680 XL/800XL

Vertaling: A.W. Bruna & Zoon
Redaktie: A.W. Bruna & Zoon
Produktie: A.W. Bruna & Zoon
Zetwerk: Mat-zet, Beesd
Druk: Vonk, Zeist
ISBN 90 229 3348 2
D/1985/0939/173

Boeken en programma's van

DATA BECKER
NEDERLANDS*

worden in de handel gebracht door:
A.W. Bruna & Zoons Uitgeversmij. b.v.,
Postbus 8411, 3503 RK Utrecht
en
A.W. Bruna & Zoon n.v.
Antwerpsesteenweg 29a, 2630 Aartselaar.

Inhoud

Atari, Basic, Peek en Poke

De programmeertaal BASIC bestaat uit een aantal instructies, waaruit volgens bepaalde regels programma's kunnen worden samengesteld. De processor kan deze instructies uitvoeren, omdat een vertaalprogramma (interpreter) de BASIC-instructies in de taal van de machine omzet. Of anders gezegd, iedere BASIC-instructie roept een bepaalde korte machinetaal-routine op.

De gebruikersvriendelijke toepassing van BASIC kost helaas verwerkings-snelheid. Ten eerste gebruikt de omzetting rekentijd. Ten tweede kunnen de programma's, die zijn samengesteld uit BASIC-instructies, in machinetaal vaak veel eenvoudiger worden gestructureerd.

Dat stoort de meeste gebruikers echter niet, want voor hobby-programmeurs is het niet van belang, of de verwerkingstijd een paar microseconden langer is. Wie echter al enige ervaring opgedaan heeft met het programmeren in BASIC, merkt op een gegeven moment, dat hij of zij met de toegestane woordenschat aan de grens van de mogelijkheden komt. Steeds vaker blijkt dan, dat bepaalde problemen met BASIC niet kunnen worden opgelost.

Maar ook in dit stadium is er een oplossing. Met de statements PEEK (kijken) en POKE (wegzetten) heeft men in een BASIC-programma direct toegang tot de geheugenplaatsen van de computer. Op die manier kan men, naar gelang persoonlijke kennis en ervaring, eigen programma's verbeteren met directe adressering.

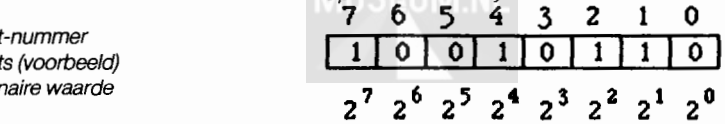
Voorwaarde daarvoor is wel, dat men weet, wat de processor met het geheugen doet, hoe PEEK en POKE werken en op welke geheugenplaatsen iets kan worden veranderd. Dit boek geeft u deze informatie.

Het hart van de ATARI is een 8-bits processor, die 65536 (van 0 t/m 65535) geheugenplaatsen (adressen) kan aansturen. Ieder van deze geheugenplaatsen kan acht bits (binary digits = binaire cijfers) in zich opnemen.

Een bit informatie komt overeen met een ja/nee beslissing. Voor de processor betekent dat spanning of geen spanning, kortweg aan of uit. Wiskundig wordt dat voorgesteld door 1 of 0.

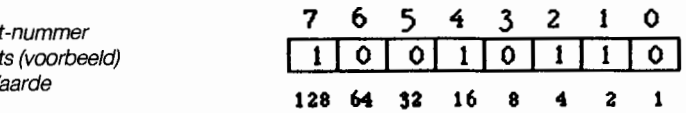
Acht van zulke bits worden samengenomen tot een byte. Een byte is de kleinste in het geheugen adresseerbare eenheid. Iedere geheugenplaats bevat precies een byte. Binnen de byte krijgt ieder bit een plaats toegewezen van 0 t/m 7. Zo ontstaat een binair getal van acht cijfers.

Opbouw van een byte



Het binaire getal van acht cijfers, dat we in het vervolg ook bit-patroon zullen noemen, is zonder veel moeite om te zetten in een decimale waarde. Daartoe moeten de binaire waarden worden opgeteld van alle bits, die in de toestand '1' (aan) zijn.

Decimale waarde van een byte



In dit voorbeeld heeft de byte dus de decimale waarde 128+0+0+16+0+4+2+0=150. Wordt met de instructie POKE n,150 de waarde 150 op adres n 'weggezet', dan ontstaat daar het getoonde bit-patroon. Wordt met de instructie PEEK (n) op adres n 'gekeken', dan vindt de instructie de decimale waarde 150.

In ATARI-BASIC hebben we dus alleen te maken met decimale getallen. Desondanks moet men van binaire getallen weten, dat de plaats van een bit binnen een byte een bepaalde waarde voorsteld. Dit noemt men het gewicht van een bit. Bit n heeft het gewicht 2 tot de macht n. Als men verder weet, dat twee tot de macht 0 gelijk is aan 1, dan kan men met PEEK en POKE aan de slag gaan.

Voor het programmeren in machinetaal zijn decimale waarden niet geschikt, want aan een waarde als 150 kan niemand onmiddellijk zien, welke bits aan zijn en welke niet. De binaire schrijfwijze is echter met z'n eindeloze rijen van nullen en enen onhandig. Daarom worden vier bits vaak tot een waarde samengenomen. Vier bits kunnen de decimale waarden 0 (=0000) t/m 15 (=1111) hebben. Het grondtal 16 is de basis van het hexadecimale (=zestien-tallig) stelsel.

Decimaal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimaal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Voorbeelden:

Binair	0000	0010	0101	1000	1010	1111
Decimaal	000	002	005	008	010	015
Hexadecimaal	0	2	5	8	A	F

Hexadecimale cijfers kunnen aan elkaar worden geregen tot hexadecimale getallen, op dezelfde manier als binaire cijfers tot binaire getallen of decimale cijfers tot decimale getallen.

Binair	0000	0000	0001	0010	0100	1101	1111	1111
Decimaal		000		018		077		255
Hexadecimaal	0	0	1	2	4	D	F	F

Een byte kan een decimale waarde aannemen van 0 t/m 255. Deze wordt in het hexadecimale stelsel door twee cijfers (00 t/m FF) weergegeven. De hexadecimale getallen hebben het grote voordeel, dat ze handzamer zijn dan de binaire getallen en toch een goede oriëntatie in de wiskundige structuur van de computer verschaffen. Dit komt doordat binaire en hexadecimale waarden nauw met elkaar verbonden zijn. Ze kunnen beide worden teruggevoerd tot het grondgetal 2, terwijl het decimale systeem praktisch geen aanknopingspunten biedt.

Wie met ATARI-BASIC werkt, hoeft zich met hexadecimale getallen nauwelijks bezig te houden. Men redt zich uitstekend met een geringe kennis van het binaire stelsel. Wanneer men echter rechtstreeks met het geheugen wil werken, is de hexadecimale notatie gemakkelijker, als men er eenmaal aan gewend is.

Spelen met getallen

Laten we, voordat we het geheugen van de computer nader gaan bekijken, eerst nog even wat oefenen met de talstelsels, die hierna volgen.

Een talstelsel bestaat uit een hoeveelheid cijfers. Deze cijfers worden aan elkaar geregen tot getallen. De plaats van een cijfer in het getal geeft zijn gewicht aan.

Gewoonlijk rekenen we met het decimale stelsel. Dit is opgebouwd uit de cijfers 0 t/m 9. Iedere plaats in een decimaal getal komt overeen met een macht van tien. De eerste of 'nulde' plaats (eenheden) heeft het gewicht 10 tot de macht 0 = 1. De tweede plaats (tientallen) heeft het gewicht 10 tot de macht 1 = 10. De derde plaats (honderdtallen) heeft het gewicht 10 tot de macht 2 = 100. Het getal 417 betekent dan 4*10 tot de macht 2 + 1*10 tot de macht 1 + 7*10 tot de macht 0 of 400 + 10 + 7.

Op dezelfde manier werken ook andere talstelsels. Zo gebruikt het duale of binaire stelsel slechts de twee cijfers 0 en 1. Iedere plaats in een binair getal komt overeen met een macht van twee.

Het hexadecimale stelsel gebruikt zestien cijfers (0 t/m F). Iedere plaats in een hexadecimaal getal komt overeen met een macht van zestien. Het getal 3E bijvoorbeeld geeft dan de waarde 3*16 tot de macht 1 + E*16 tot de macht 0 = 3*16 + 14*1 weer. In decimale schrijfwijze is dat 62 en binair 0011 1110. (De spatie tussen de beide blokken dient slechts om een beter overzicht te krijgen en de nullen op de hoogste plaatsen kunnen ook weggelaten worden, net zoals in de decimale schrijfwijze 00531 dezelfde waarde heeft als 531).

In de informatica is het gebruikelijk, de acht bits van een byte in twee blokken te verdelen. Een zo'n blok van vier bits wordt een nibble genoemd. Een nibble kan een decimale waarde aannemen van 0 t/m 15 en dat komt overeen met een hexadecimale waarde van 0 t/m F. Een hexadecimaal getal stelt dus een binair getal van vier cijfers voor. Dit hebben we in het vorige hoofdstuk ook al gezien.

Wiskundigen, die zich met de meest buitenissige spelletjes bezighouden, werken met alle denkbare talstelsels. Zo is bijvoorbeeld het twaalftallig stelsel zeer interessant. Het is gebaseerd op de twaalf, het geeft dus cijfers van 0 t/m B. Oude maat- en talstelsels gebruikten deze basis, bijvoorbeeld de tijd en de duim. De voorliefde voor dit systeem vindt zijn grond waarschijnlijk hierin, dat men de twaalf niet alleen kan halveren, maar ook gemakkelijk door drie of vier kan delen.

De omrekening van het ene talstelsel naar het andere is vaak moeilijk, vooral als ze op verschillende bases berusten, zoals twee en tien of zestien en tien. Daarom ligt het voor de hand, voor deze omzettingen programma's te schrijven.

Het programma DEZXBIN.BEC zet een decimaal getal van 0 t/m 255 om in het overeenkomstige binaire getal van acht cijfers. Het programma is niet beveiligd tegen verkeerde invoer.

```
0 REM DECXBIN.BEC
1 REM ********************************
2 REM *                              *
3 REM *  OMZETTING DECIMAAL-BINAIR   *
4 REM *                              *
5 REM ********************************
10 DIM B(7)
20 FOR J=0 TO 7:B(J)=0:NEXT J
30 ? CHR$(125):? "Geef een decimaal getal"
40 ? :? "van 0 t/m 255 en <RETURN>":?
50 INPUT D
60 R=D
90 IF D>127 THEN D=D-128:B(7)=1
100 IF D>63 THEN D=D-64:B(6)=1
110 IF D>31 THEN D=D-32:B(5)=1
120 IF D>15 THEN D=D-16:B(4)=1
130 IF D>7 THEN D=D-8:B(3)=1
140 IF D>3 THEN D=D-4:B(2)=1
150 IF D>1 THEN D=D-2:B(1)=1
160 B(0)=D
170 ? :? "decimaal ";R;" = binair ";B(7);B(6);B(5);B(4);B(3);B(2);B(1);B(0)
180 ? :? "Nog een getal? (J/N)"
190 IF PEEK(764)=1 THEN POKE 764,255:GOTO 20
200 IF PEEK(764)=35 THEN POKE 764,255:END
210 GOTO 190
```

10: De variabele B wordt op 8 (0 t/m 7) geDIMensioneerd, om de acht bits op te slaan.

30: CHR$ (125) wist het beeldscherm.

90 t/m 150: Hier wordt vastgesteld, welke bits aan zijn. Als het ingevoerde decimale getal groter is dan 127, dan moet bit 7 (gewicht 128) aan zijn. Is dat het geval, dan wordt het ingevoerde getal D met 128 verminderd. Als het nu nog groter is dan 63, dan moet bit 6 (gewicht 64) aan zijn, enzovoorts.

160: Hier kan D alleen nog maar de waarde 0 of 1 hebben en dat is de waarde van bit 0 (gewicht 1).

170: drukt het resultaat af.

180: Wil de gebruiker nog een getal laten omrekenen, dan moet hier een 'J' ingevoerd worden.

190 en 200: lezen de waarde op adres 764. Hoe dat precies werkt, zal later nog uitvoerig besproken worden.

210: springt terug naar regel 190. Zo ontstaat een lus, die alleen door de toets 'J' of 'N' kan worden verlaten. Omdat het programma verder niet beveiligd is, kan het natuurlijk altijd met BREAK worden gestopt.

Wie enige programmeer-ervaring heeft, zal onmiddellijk zien, dat de regels 90 t/m 150 veel efficiënter gemaakt kunnen worden, bijvoorbeeld door gebruik te maken van een FOR-NEXT lus. Vervang de regels 90 t/m 160 in het voorgaande programma eens door de regels 110 t/m 130 van het programma DEM0001.BEC en probeer het zelf! Een korter programma hoeft echter niet altijd sneller te zijn. De berekening van de tweede macht in de nieuwe regel 110 kost namelijk erg veel tijd. Het verschil is verbazingwekkend:

```
0 REM DEM0001.BEC
1 REM ********************************
2 REM *                              *
3 REM * LANGZAMER MET FOR-NEXT LUS! *
4 REM *                              *
5 REM ********************************
110 FOR J=7 TO 1 STEP -1:BW=2^J
120 IF D>BW-1 THEN D=D-BW:B(J)=1
130 NEXT J:B(0)=D
```

Het volgende programma doet het omgekeerde. De gebruiker voert een binair getal in van acht cijfers en de computer berekent dan de decimale waarde.

```
0 REM BINXDEC.BEC
1 REM ********************************
2 REM *                              *
3 REM *  OMZETTING BINAIR-DECIMAAL   *
4 REM *                              *
5 REM ********************************
10 DIM B$(8),A(8)
20 A(8)=1:A(7)=2:A(6)=4:A(5)=8:A(4)=16:A(3)=32:A(2)=64:A(1)=128
30 ? CHR$(125):? "Geef een binair getal"
40 ? :? "van acht cijfers en <RETURN>":?
50 INPUT B$
60 FOR J=1 TO 8
70 IF B$(J,J)="1" THEN D=D+A(J)
80 NEXT J
100 ? :? "binair ";B$;" = decimaal ";D
110 ? :? "Nog een getal? (J/N)"
120 IF PEEK(764)=1 THEN POKE 764,255:D=0:GOTO 30
130 IF PEEK(764)=35 THEN POKE 764,255:END
140 GOTO 120
```

10: B$ wordt op acht (1 t/m 8) geDIMensioneerd, om het binaire getal op te slaan. A wordt op negen (0 t/m 8) geDIMensioneerd. We gebruiken echter alleen de variabelen A(1) t/m A(8), omdat daardoor de betrekking tot B$ duidelijker is. We zouden natuurlijk net zo goed met A(0) t/m A(7) kunnen werken.

20: De acht variabelen A(1) t/m A(8) krijgen de decimale waarden, die overeen komen met de gewichten van de overeenkomstige binaire plaatsen.

60 t/m 80: Deze FOR-NEXT lus leest de afzonderlijke karakters van het ingevoerde binaire getal, dat als string in B$ is opgeslagen. Het ATARI-BASIC statement B$(n,m) neemt een stuk uit B$ en wel dat stuk, dat begint met het karakter dat op de n-de plaats in de string staat en eindigt met het karakter op de m-de plaats in B$. B$(J,J) neemt dus slechts een enkel karakter uit B$ en wel het karakter, dat zich op plaats J bevindt. De karakters in een string worden van links naar rechts geteld, op dezelfde manier als wij lezen. Het eerste

karakter van B$ is bit 7 dat de decimale waarde 0 heeft als het '0' is en 128 als het '1' is. De bij ieder bit behorende decimale waarde wordt bij de variabele D opgeteld.

Bij de omzetting in hexadecimale getallen is er een kleine bijkomende moeilijkheid, omdat aan de cijfers 0 t/m 9, die de computer als numerieke waarden accepteert, ook nog de cijfers A t/m F moeten worden toegevoegd, die door de computer alleen als string kunnen worden verwerkt. Daarom is het ook gebruikelijk, hexadecimale getallen aan te duiden door er een $ voor te zetten, dus 243 = SF3:

```
0 REM DECXHEX.BEC
1 REM ********************************
2 REM *                              *
3 REM * OMZETTING DECIMAAL-HEXADEC.  *
4 REM *                              *
5 REM ********************************
10 DIM HEX$(16),H$(1),L$(1)
20 HEX$="0123456789ABCDEF"
30 ? CHR$(125):? "Geef een decimaal getal"
40 ? :? "van 0 t/m 255 en <RETURN>":?
50 INPUT D
60 R=D
70 A=INT(D/16):H$=HEX$(A+1,A+1)
80 B=D-A*16:L$=HEX$(B+1,B+1)
170 ? :? "decimaal ";R;" = hexadecimaal ";H$;L$
180 ? :? "Nog een getal? (J/N)"
190 IF PEEK(764)=1 THEN POKE 764,255:GOTO 30
200 IF PEEK(764)=35 THEN POKE 764,255:END
210 GOTO 190
```

10: HEX$ moet de zestien cijfers opslaan, die gebruikt worden. Het gezochte hexadecimale getal bestaat uit twee cijfers. Het hoge gedeelte (HI = high) wordt in H$ ondergebracht, het lage gedeelte (LO = low) in L$.

20: Hier worden de cijfers 0 t/m F in HEX$ opgeborgen.

50: Het decimale getal tussen 0 en 255, dat door de gebruiker is ingevoerd, wordt in D opgeslagen.

70: Wanneer we D delen door 16, bevat het gehele (INTegere) gedeelte A het resultaat voor het hoge gedeelte van het hex-getal. Omdat dit resultaat decimaal is, moeten we uit HEX$ nog het juiste hex-cijfer halen: HEX$(A+1,A+1). Omdat het hex-cijfer O het eerste karakter in HEX$ is, moet bij het resultaat A nog 1 worden opgeteld!

80: De rest van de bovenstaande deling bepaalt het lage hex-cijfer. D-A*16 geeft de rest in decimale vorm, die op dezelfde manier in een hex-cijfer wordt omgezet.

Bij de omzetting van hexadecimale in decimale getallen hebben we hetzelfde probleem, namelijk dat we de hexadecimale cijfers A t/m F moeten omzetten:

10: Het in te voeren hexadecimale getal mag slechts twee plaatsen groot zijn. Overeenkomstig wordt hier geDIMensioneerd.

```
0 REM HEXXBIN.BEC
1 REM ********************************
2 REM *                              *
3 REM *  OMZETTING HEXADEC.-BINAIR   *
4 REM *                              *
5 REM ********************************
10 DIM HEX$(2),H$(1),L$(1),B(7)
20 FOR J=0 TO 7:B(J)=0:NEXT J
30 ? CHR$(125):? "Geef een hexadecimaal getal"
40 ? :? "van twee cijfers en <RETURN>":?
50 INPUT HEX$
60 H$=HEX$(1,1):L$=HEX$(2,2)
70 IF H$="A" THEN H=10:GOTO 150
80 IF H$="B" THEN H=11:GOTO 150
90 IF H$="C" THEN H=12:GOTO 150
100 IF H$="D" THEN H=13:GOTO 150
110 IF H$="E" THEN H=14:GOTO 150
120 IF H$="F" THEN H=15:GOTO 150
130 H=VAL(H$)
150 IF L$="A" THEN L=10:GOTO 220
160 IF L$="B" THEN L=11:GOTO 220
170 IF L$="C" THEN L=12:GOTO 220
180 IF L$="D" THEN L=13:GOTO 220
190 IF L$="E" THEN L=14:GOTO 220
200 IF L$="F" THEN L=15:GOTO 220
210 L=VAL(L$)
220 IF H>7 THEN H=H-8:B(7)=1
230 IF H>3 THEN H=H-4:B(6)=1
240 IF H>1 THEN H=H-2:B(5)=1
250 B(4)=H
260 IF L>7 THEN L=L-8:B(3)=1
270 IF L>3 THEN L=L-4:B(2)=1
280 IF L>1 THEN L=L-2:B(1)=1
290 B(0)=L
300 ? :? "hexadecimaal ";HEX$;" = binair ";B(7);B(6);B(5);B(4);B(3);B(2);B(1);B(
0)
310 ? :? "Nog een getal? (J/N)"
320 IF PEEK(764)=1 THEN POKE 764,255:GOTO 20
330 IF PEEK(764)=35 THEN POKE 764,255:END
340 GOTO 320
```

```
0 REM BINXHEX.BEC
1 REM ********************************
2 REM *                              *
3 REM *  OMZETTING BINAIR-HEXADEC.   *
4 REM *                              *
5 REM ********************************
10 DIM HEX$(16),H$(1),L$(1),B$(8),A(8)
20 HEX$="0123456789ABCDEF"
30 A(8)=1:A(7)=2:A(6)=4:A(5)=8:A(4)=1:A(3)=2:A(2)=4:A(1)=8
40 ? CHR$(125):? "Geef een binair getal"
50 ? :? "van acht cijfers en <RETURN>":?
60 INPUT B$
70 FOR J=1 TO 4
80 IF B$(J,J)="1" THEN H=H+A(J)
90 NEXT J
100 H$=HEX$(H+1,H+1)
110 FOR J=5 TO 8
120 IF B$(J,J)="1" THEN L=L+A(J)
130 NEXT J
140 L$=HEX$(L+1,L+1)
170 ? :? "binair ";B$;" = hexadecimaal ";H$;L$
180 ? :? "Nog een getal? (J/N)"
190 IF PEEK(764)=1 THEN POKE 764,255:H=0:L=0:GOTO 30
200 IF PEEK(764)=35 THEN POKE 764,255:END
210 GOTO 190
```

60: Uit de HEX$-INPUT worden H$ en L$ bepaald.

70 t/m 120: Dan wordt in H de decimale waarde van H$ genoteerd, als H$ een cijfer van A t/m is.

130: Is dat niet het geval, dan kan met het statement VAL het cijfer in H$ direct in een numerieke waarde worden omgezet.

150 t/m 210: Op dezelfde manier wordt L$ omgezet in L.

290: De gezochte decimale waarde kan nu eenvoudig berekend worden. De hoge plaats van het hexadecimale getal heeft het gewicht 16 (dus H*16), de lage plaats het gewicht 1 (dus L*1) = H*16+L.

De omzetting van een binair in een hexadecimaal getal is nu nog slechts een combinatie van de al gebruikte programma-routines:

Ook voor de omgekeerde bewerking, de omzetting van hex in binair, is een verdere toelichting overbodig:

Als u het leuk vindt, dan kunt u deze zes kleine programma's samenvatten in een groot utility- (= hulp-) programma. Op het beeldscherm verschijnt dan een menu, dat de zes mogelijke omzettingen laat zien. Door op de bijbehorende getallentoets te drukken, wordt gesprongen naar dat gedeelte van het programma dat u nodig heeft. Het programma UMWAND.BEC geeft de structuur aan. U hoeft alleen nog maar tussen de regels 1000 t/m

```
0 REM OMZETTEN.BEC
1 REM ********************************
2 REM *                              *
3 REM *   GETALOMZETTINGEN / MENU    *
4 REM *                              *
5 REM ********************************
10 DIM HEX$(16),H$(1),L$(1),B$(8),B(7),A(8)
20 HEX$="0123456789ABCDEF"
100 ? CHR$(125):POKE 82,0:POKE 752,1
110 ? :? "            GETALOMZETTINGEN"
120 ? "  ____________________________________"
130 ? :? " decimaal getal in binair getal    <1>"
140 ? :? " decimaal getal in hexadec. getal  <2>"
150 ? :? :? :? " binair getal in decimaal getal     <3>"
160 ? :? " binair getal in hexadec. getal     <4>"
170 ? :? :? :? " hexadec. getal in decimaal getal   <5>"
180 ? :? " hexadec. getal in binair getal     <6>"
200 OPEN £1,4,0,"K:":GET £1,T:CLOSE £1
210 IF T<49 THEN 200
220 IF T>54 THEN 200
230 GOTO (T-48)*1000
1000 REM BEGIN PROGRAMMA: DEC IN BIN
2000 REM BEGIN PROGRAMMA: DEC IN HEX
3000 REM BEGIN PROGRAMMA: BIN IN DEC
4000 REM BEGIN PROGRAMMA: BIN IN HEX
5000 REM BEGIN PROGRAMMA: HEX IN DEC
6000 REM BEGIN PROGRAMMA: HEX IN BIN
```

6000 de aangegeven programma's in te voegen. U kunt dan tevens beveiligingen (TRAPs) aanbrengen tegen verkeerde invoer:

200: Hier wordt een leeskanaal geopend naar het toetsenbord (K=keyboard), de ATASCII-waarde van de ingedrukte toets wordt in de variabele T opgeslagen en het datakanaal wordt weer gesloten.

210 en 220: testen, of een van de toetsen '1' t/m '6' (ATASCII-waarden 49 t/m 54) werd ingedrukt.

230: Is dat het geval, dan wordt hier de bijbehorende sprong berekend.

Bit-parade

Een computer is in feite een schakel-netwerk, waarin elk bit als een schakelaar werkt. Acht van zulke schakelaars zijn tot een informatie-eenheid (byte) samengenomen en worden als decimale waarde (data-byte) weergegeven. Aan zo'n waarde, bijvoorbeeld 191, kan men, ook met veel oefening, niet onmiddellijk zien welke bits aan (1) zijn en welke bits uit (0) zijn. Er bestaat ook geen BASIC-instructie, waarmee direct de toestand van bepaalde bits kan worden uitgelezen (ofschoon veel BASIC-instructies wel direct invloed hebben op afzonderlijke bits).

In veel gevallen is het echter gewenst, een byte te bekijken en het daarin vastgelegde bit-patroon te bepalen of de toestand van een bepaald bit op te vragen. Het statement PEEK (n) vindt slechts een decimale waarde. Met de instructie PRINT PEEK (n) wordt de waarde van de op adres n gevonden data-byte op het beeldscherm getoond. Met B=PEEK(n) kan deze waarde in een variabele worden opgeslagen.

Om het bit-patroon op een bepaalde geheugenplaats vast te stellen, moet de data-byte in afzonderlijke bits worden opgesplitst. Hoe dat gedaan moet worden, is reeds getoond in het programma DEZXBIN.BEC. Hier volgt nog eens de verkorte versie:

programma BITBREAK.BEC

20: Het toeval bepaalt, welk van de acht bits (0 t/m 7) door het programma wordt onderzocht.

30: Het data-byte krijgt met R eveneens een toevallige waarde en wel uit de getallenreeks van 0 t/m 255.

40 t/m 120: Hier wordt het data-byte in zijn afzonderlijke bit-bestanddelen ontleed.

```
0 REM BITBREAK.BEC
1 REM ********************************
2 REM *                              *
3 REM *OPSPLITSEN VAN EEN DATA-BYTE *
4 REM *                              *
5 REM ********************************
10 DIM B(7)
20 X=INT(RND(0)*8)
30 R=PEEK(53770):D=R
40 FOR J=0 TO 7:B(J)=0:NEXT J
50 IF D>127 THEN D=D-128:B(7)=1
60 IF D>63 THEN D=D-64:B(6)=1
70 IF D>31 THEN D=D-32:B(5)=1
80 IF D>15 THEN D=D-16:B(4)=1
90 IF D>7 THEN D=D-8:B(3)=1
100 IF D>3 THEN D=D-4:B(2)=1
110 IF D>1 THEN D=D-2:B(1)=1
120 B(0)=D
130 ? :? "In de data-byte ";R;" is bit £";X;" = ";B(X)
140 GOTO 20
```

Deze programma-routine gebruikt relatief veel geheugenruimte. Moet slechts de toestand van een enkel bit worden vastgesteld, dan kan met een kortere formule worden volstaan, die echter niet sneller wordt verwerkt. Om die formule te begrijpen, moet men weten wat de verhouding is tussen de gewichten van de verschillende bits. De waarde van bit n+1 is precies twee keer zo groot als die van bit n. Wordt de waarde van een hoger bit door de waarde van een lager bit gedeeld, dan is het resultaat een even getal.

Voorbeeld: het data-byte is 128.

bit-nummer	7	6	5	4	3	2	1	
gewicht G	128	64	32	16	8	4	2	
quotiënt (128/G)	1	2	4	8	16	32	64	1

Dus als een data-byte door het gewicht van een bit wordt gedeeld en het resultaat is een even getal, dan is het desbetreffende bit uit (0). Is het resultaat een oneven getal, dan is dat bit aan (1).

Voorbeeld: het data-byte is 133.

bit-nummer	7	6	5	4	3	2	1	0
gewicht G	128	64	32	16	8	4	2	1
INT-quotiënt	1	2	4	8	16	33	66	133
bit-toestand	1	0	0	0	0	1	0	1

Ook op deze manier kan men dus zien, dat het data-byte 133 het bit-patroon 1000 0101 voorstelt. Uit dit voorbeeld blijkt tevens, dat bij het delen van het data-byte door het gewicht van een bit er niet altijd een geheel (integer) getal uitkomt.

De formule die moet vaststellen, of een bepaald bit aan is, moet dus de volgende bewerkingen verrichten:

1: de waarde van het X-de bit berekenen (= 2 tot de macht X).
2: het data-byte R door deze waarde delen.
3: van het quotiënt alleen het gehele deel bewaren (= INT(R/2 X).
4: vaststellen of deze waarde even of oneven is.

Een getal is even, als het zonder rest deelbaar is door 2, met andere woorden als het quotiënt een geheel getal is. Wordt een getal n door 2 gedeeld en INT (n/2) = n/2, dan is n een even getal. Is INT(n/2) niet gelijk aan n/2, dan is n oneven. De volledige formule luidt dus:

```
0 REM FORMULE01.BEC
10 REM INT(INT(R/2↑X)/2)<>INT(R/2↑X)/2
```

programma FORMEL01.BEC

Als aan deze voorwaarde voldaan wordt, dan is bit X = 1 (aan).

Nu volgt nog een klein programma, dat de moeizaam verkregen formule ook daadwerkelijk gebruikt:

```
0 REM BITPEEK.BEC
1 REM ********************************
2 REM *                              *
3 REM *   VASTSTELLEN VAN EEN BIT    *
4 REM *                              *
5 REM ********************************
10 B=0
20 X=INT(RND(0)*8)
30 R=PEEK(53770)
40 IF INT(INT(R/2^X)/2)<>INT(R/2^X)/2 THEN B=1
50 ? :? "In de data-byte ";R;" is bit £";X;" = ";B
60 GOTO 10
```

programma BITPEEK.BEC

20 en 30: produceren weer toevallige waarden voor de bit-positie X en het data-byte R.

40: De super-formule in volle actie.

50: Het resultaat wordt gegeven.

Natuurlijk zou men de regels 40 en 50 in een FOR-NEXT lus kunnen zetten, die alle acht bit-nummers (0 t/m 7) afloopt. B moet dan op 8 worden geDIMensioneerd (DIM B(7)). Het programma is daarmee nog steeds aanzienlijk korter dan BITBREAK.BEC en zal daarom minder geheugen gebruiken. Het loopt echter een stuk langzamer, zodat deze formule eigenlijk alleen interessant is, als naar de toestand van een enkel bit wordt gevraagd.

Hoe met binaire getallen wordt gerekend, hoeft men voor ATARI-BASIC niet te weten. PEEKs en POKEs werken met decimale getallen. Moet de waarde op een adres worden veranderd, dan moet met PEEK(n) de waarde op geheugenplaats n worden uitgelezen, door optellen of aftrekken van een decimale waarde worden veranderd en met POKE n de nieuwe waarde weer op geheugenplaats n worden weggezet. Dus POKE n,PEEK(n +/- m). Is n-m kleiner dan 0 of is n+m groter dan 255, dan geeft de computer de foutmelding ERROR-3 (getallenbereik overschreden).

Tot slot nog een kleine tip. In veel gevallen is het gewenst een bit-patroon om te keren, dus waar een 1 staat, moet een 0 worden geschreven en waar een 0 staat moet een 1 komen. Zo'n bewerking noemt men inverteren en wordt vaak toegepast in het grafische geheugengebied.

oorspronkelijk bit-patroon (decimale waarde)	1 0 0 1 0 1 1 0 = 150
+ geinverteerd bit-patroon (decimale waarde)	0 1 1 0 1 0 0 1 = 105
= som (decimale waarde)	1 1 1 1 1 1 1 1 = 255

Het omgekeerde rekenproces verloopt overeenkomstig:

alle acht bits aan (decimale waarde)	1 1 1 1 1 1 1 1 = 255
– bit-patroon (decimale waarde)	1 0 0 1 0 1 1 0 = 150
= inverse bit-patroon (decimale waarde)	0 1 1 0 1 0 0 1 = 105

Om een bit-patroon te inverteren, moet de decimale waarde daarvan worden afgetrokken van 255 en de decimale waarde van het resultaat bevat dan het inverse bit-patroon. POKE n,255-PEEK(n) inverteert het bit-patroon op adres n.

Rom en Ram

De constructie van de microprocessor bepaalt hoeveel geheugenplaatsen bestuurd kunnen worden. Het hart van de ATARI is een 6502, die 65536 geheugenplaatsen kan adresseren. Om waarden van 0 t/m 65535 weer te geven zijn zestien bits = twee bytes nodig. Telkens worden 256 bytes samengenomen tot een pagina (page). Het hoogstwaardige (HI) byte adresseert de geheugenpagina. Er kunnen dus 256 pagina's worden bestuurd. Het laagstwaardige (L0) byte adresseert de geheugenplaats op de pagina. Op deze manier kunnen dus 256*256=65536 geheugenplaatsen met een getal, het adres, worden gerangschikt.

Om een bepaald adres te registreren, zijn dus twee bytes nodig, die altijd in de volgorde L0, H1 worden opgeslagen. Het gezochte adres volgt dan uit de formule L0+H1*256. Als bijvoorbeeld op de adressen 88 en 89 een wijzer (vector) naar een bepaald adres is opgeslagen, dan vindt men met de volgende opdracht het gezochte adres:

```
ADRES = PEEK(88)+PEEK(89)*256
```

Naast de twee-byte vectoren zijn er ook wijzers, die slechts een byte groot zijn en dus alleen naar het begin van een pagina kunnen wijzen. Het gezochte adres is in zo'n geval te vinden met:

```
ADRES = PEEK(106)*256
```

4 pagina's van ieder 256 bytes, dus 1024 bytes, worden een kbyte (kilo-byte) genoemd. De eenheid kilo wordt dus in een van het normale afwijkende betekenis gebruikt. In veel gevallen moet ook op de 4k-grenzen gelet worden. Adresseert men het geheugen hexadecimaal, dus van $0000 t/m $FFFF, dan geeft het hoogste cijfer van dit hex-getal het veelvoud van vier kbyte aan.

De ATARI 800XL beschikt in totaal over 64 kbyte (namelijk 65536 bytes) geheugen. Deze ruimte staat de gebruiker echter niet geheel ter beschikking. Ongeveer de helft van alle adressen zijn nodig voor het in bedrijf zijn van de computer, voor de diverse rand-apparaten en om bepaalde gegevens op te slaan, die altijd ter beschikking moeten staan.

Deze gegevens zijn voor een deel blijvend in het geheugen opgeslagen, zodat ze ook bewaard blijven als de spanning wordt uitgeschakeld. Deze geheugeninhoud kan door de gebruiker natuurlijk niet worden veranderd. Zulke geheugengedeeltes heten ROM (Read Only Memory = alleen leesgeheugen).

Geheugenplaatsen, die steeds andere waarden kunnen opslaan, worden RAM (Random Access Memory = geheugen met willekeurige toegang of lees-/schrijf-geheugen) ge-

noemd. Het systeem gebruikt een groot aantal van zulke RAM-adressen om variërende waarden, bijvoorbeeld bepaalde wijzers, vast te houden. Veel van deze registers kunnen door de gebruiker worden veranderd, wanneer bepaalde waarden worden ingePOKEd. Vele ervan worden echter door het systeem zo snel vernieuwd dat deze ingreep met de trage BASIC-POKE practisch zonder uitwerking blijft.

Bij veel registers hebben afzonderlijke bits bepaalde functies, bij andere vervullen meerdere bits samen een bepaalde taak. De acht beschikbare bits op een geheugenplaats worden niet altijd allemaal gebruikt.

Bij het inschakelen van de ATARI 800XL staan de gebruiker ruim 37 kbyte, 148 pagina's of om precies te zijn 37902 bytes ter beschikking voor programma's en gegevens. Is een diskette-station aangesloten, dan wordt deze waarde vermeerderd met circa 31,5 kbyte, 126 pagina's of 32274 bytes, omdat door het DOS (Disc Operating System = diskette besturingssysteem) ook veel plaats wordt gebruikt. De BASIC-instructie PRINT FRE(0) geeft de nog te gebruiken geheugenruimte.

Geheugenoverzicht

Adressering Relatief aan de Programmateller
Adresseringsmethoden, die gebruik maken van offsets (= verplaatsingen) ten opzichte van de programmateller, helpen ons bij het schrijven van positie-on.

d = $-adres functie

---

0 = $0000 laagste adres (bottom of memory)

0 = $0000 OS zero-page RAM

128 = $0080 BASIC zero-page RAM

256 = $0100 6502 stack

512 = $0200 interrupt-wijzer, paddles, kleurenregister, enz.

768 = $0300 disk-parameter, IOCBs 0-7, printerbuffer

1024 = $0400 cassettebuffer

1152 = $0480 OS-RAM

1536 = $0600 vrij voor korte machinetaal-routines

1792 = $0700 opstart-RAM voor gebruiker of DOS, indien geladen

2048 = $0800 RAM of DOS, indien geladen

11008 = $2B00 RAM (het einde van DOS is variërend in lengte)

32768 = $8000 RAM of rechter ROM-module (#)

Display list, beeldschermgeheugen.
Het start-adres is afhankelijk van de gekozen grafische modus. Eind-adres inclusief tekstraam is altijd 40959 = $9FFF

40960 = $A000 RAM of linker ROM-module (#) / BASIC-ROM (XL)

49152 = $C000 ongebruikte ROM (#). OS-ROM (XL)

53504 = $D000 CTIA (US), GTIA (NL)

53504 = $D100 vrij voor toekomstige uitbreidingen

53760 = $D200 POKEY

54016 = $D300 PIA

54272 = $D400 ANTIC

54528 = $D500 module-interface controle

54784 = $D600 vrij voor toekomstige uitbreidingen

55296 = $D800 I/O chips, floating point ROM

57344 = $E000 standaard tekenset ROM

Hier begint de OS-ROM, het besturingssysteem. Het loopt tot 65535 = $FFFF

58368 = $E400 editor

58384 = $E410 beeldscherm-vectoren (screen)

58400 = $E420 toetsenbord-vectoren (keyboard)

58416 = $E430 afdrukeenheid-vectoren (printer)

58432 = $E440 datarecorder-vectoren (cassette)

58448 = $E450 sprong-adressen (JMP-vectoren)

58496 = $E480 RAM-vectoren voor 'Powerup'

58534 = $E483 CIO

59093 = $E605 interrupt-driver

59716 = $EC00 SIO

60906 = $EDEC disk-driver

61048 = $EE78 printer-driver

61249 = $EF41 cassette-driver

61667 = $F0E3 monitor-routine

62436 = $F3E4 beeldscherm- en toetsenbord-driver

65535 = $FFFF hoogste adres (top of memory)

Bespreking van de geheugenplaatsen

0,1 $0,$1 LINZBS

Wordt gebruikt door de reset-routine bij de geheugentest. Wordt vernieuwd door de monitor-RAM en kan gebruikt worden als opslag voor de timer op van VBLANK.

2,3 $2,$3 CASINI

Vector voor de initiatie van het opstarten van de cassetterecorder. Was het opstarten succesvol, dan volgt een sprong (JSR) naar het aangegeven adres. De vijfde (LO) en zesde (H1) byte van een cassette-file bevatten het beginadres.

4,5 $4,$5 RAMLO

RAM-vector voor de geheugengrootte-test bij Powerup. Wordt ook gebruikt voor het disk opstart-adres (1798 = $706).

6 $6 TRAMSZ

Tussenregister voor de geheugengrootte-test. Wordt gebruikt tijdens Powerup en geeft zijn waarde dan af aan RAMTOP (106 = $6A). Bevindt zich een insteekmodule in de cartridge-ingang (XL), respectievelijk in de linker cartridge-ingang, bijvoorbeeld BASIC, (#), dan wordt hier een 1 gelezen.

7 $7 TSTDAT

Dataregister voor de RAM-test, neemt de waarde over van de op een bepaald moment te testen geheugenplaats. Wordt hier een 1 gelezen, dan bevindt zich in de rechter ingang een cartridge (#).

8 $8 WARMST

Warme-start-aanduiding (flag). De Powerup zet hier een 0 neer. Deze waarde blijft behouden, totdat hij wordt omgePOKEd of RESET wordt ingedrukt. RESET zet hier 255 neer (alle bits aan).

9 $9 BOOT?

Wanneer het opstarten (boot) van de diskette-eenheid lukt, zet de processor hier een 1 neer (bit 0 aan), na het opstarten van de cassetterecorder een 2 (bit 1 aan). BOOT? bevat een 0, wanneer geen van beide apparaten werd opgestart. Geeft tevens aan, of bij RESET

de cassette- (CASINI 2,3 = $2,$3) of de DOS-vector (DOSVEC 10,11 = $A,$B) wordt gebruikt. Koude start probeert beide opstart-routines en zet dan de desbetreffende bits aan.

Wordt in BOOT? 255 gezet (alle bits aan), dan blijft het besturingssysteem steken (hang-up), als op RESET wordt gedrukt. Na het doorlopen van de opstart-routines kan dus met deze wijzer de RESET-knop onschadelijk worden gemaakt (software bescherming).

10,11 $A,$B DOSVEC

Startvector voor diskettesoftware. Het BASIC-commando DOS veroorzaakt een sprong naar dit adres. De wijzer kan worden veranderd, maar wordt door RESET weer hersteld. Om dit te verhinderen moeten de adressen 5446 (L0) en 5450 (H1) = $1546,$154A, die gewoonlijk naar DOSVEC wijzen, worden veranderd.

Het volgende programma laat zien, hoe met behulp van DOSVEC het BASIC-commando DOS vervangen kan worden:

```
0 REM ADR10.BEC
1 REM ********************************
2 REM *                              *
3 REM *  DOS DOOR INDRUKKEN TOETS    *
4 REM *                              *
5 REM ********************************
10 REM PROGRAMMA
20 REM PROGRAMMA
30 REM PROGRAMMA
40 REM PROGRAMMA
50 REM PROGRAMMA
60 IF PEEK(764)=255 THEN 10
70 OPEN #1,4,0,"K:":GET #1,D:CLOSE #1
80 IF D=68 THEN 100
90 GOTO 10
100 LET DOS=USR(PEEK(10)+PEEK(11)*256)
```

programma ADR10.BEC

10 t/m 50: staan voor een willekeurig programma.

60: Deze regel controleert, of een toets van het toetsenbord is ingedrukt.

70: Als dat het geval is, wordt hier via een datakanaal naar het toetsenbord de ATASCII-waarde van de ingedrukte toets in de variabele D opgeslagen.

80: Als de waarde van D 68 is, dan werd de toets D ingedrukt en gaat het programma verder op regel 100.

90: Als toets 'D' niet werd gevonden, dan wordt teruggesprongen naar regel 10.

100: ATARI speelt zelfs mee, als een BASIC-opdracht (hier DOS) als variabele wordt gebruikt. We moeten het alleen met LET nadrukkelijk kenbaar maken.

De instructie USR spring naar een machinetaal-routine, die begint bij de achter USR aangegeven geheugenplaats. Wanneer hier DOSVEC als startadres wordt gegeven, dan springt USR naar DOS. De USR-instructie moet een variabele toegewezen krijgen (hier DOS), die echter verder geen functie heeft.

12,13 $C,$D DOSINI

Initiatie-vector voor het opstarten van de diskette-eenheid. Hier wordt het adres opgeslagen, waar het gebruikersprogramma begint na het laden van DOS. Met een indirecte JSR hierheen kan gebruikers-software worden geïnitieerd.

14,15 $E,$F APPMHI

Vector naar de eerste vrije geheugenplaats in de gebruikers-RAM. Wijst aan, tot waar het geheugen door een BASIC-programma in gebruik is. Het beeldschermgeheugen en de display-list kunnen uitbreiden van het adres, waar deze vector naar toe wijst, t/m RAMTOP (40959 = $9FFF).

16 $10 POKMSK

Interrupt-masker voor POKEY. Parallel-register van IRQEN (53774 = $D20E). Elk van de acht bits heeft betrekking op een mogelijke interrupt. Het 'uit' zetten van een bit maakt de desbetreffende interrupt onmogelijk:

bit 7	(128)	BREAK-toets
bit 6	(64)	andere toets
bit 5	(32)	seriële invoer-gegevens beschikbaar
bit 4	(16)	seriële uitvoer-gegevens beschikbaar
bit 3	(8)	seriële uitvoer beeindigd
bit 2	(4)	POKEY-tijdregister 4
bit 1	(2)	POKEY-tijdregister 2
bit 0	(1)	POKEY-tijdregister 1

Hiermee kan men nieuwsgierige gebruikers tegenhouden. Waarden moeten hier en in IRQEN (53774 = $D20E) worden gePOKEd. Iedere waarde kleiner dan 128 schakelt de BREAK-toets uit. Met 127 stopt BREAK weliswaar het programma, maar onmiddellijk blijft het besturingssysteem steken. RESET is dan de enige manier. En hoe de RESET-toets voor onbevoegden onschadelijk gemaakt kan worden, werd bij BOOT? al uitgelegd. Probeer eens verschillende waarden uit.

In ieder geval zet iedere beeldscherm-instructie (screen 'S:' of editor 'E:') in bit 7 een 1 en geeft daarmee de interrupt-mogelijkheid van BREAK weer vrij. Na iedere PRINT-, GRAPHICS- of OPEN-instructie ('S:' of 'E:') moet de BREAK-toets dus opnieuw worden uitgeschakeld. Dat kan men het beste doen met een korte subroutine:

```
0 REM ADR16BRK.BEC
1 REM ********************************
2 REM *                              *
3 REM * SUBROUTINE BREAK-BLOKKERING *
4 REM *                              *
5 REM ********************************
10 GOSUB 5000
20 ? "De BREAK-toets werkt niet meer.":GOTO 20
5000 BRK=PEEK(16)
5010 IF BRK>127 THEN BRK=BRK-128
5020 POKE 16,BRK
5030 POKE 53774,BRK
5040 RETURN
```

programma ADR16BRK.BEC

ROMs met de nieuwe 'B'-versie van het OS hebben een aparte wijzer voor de BREAK-interrupt. Zie hiervoor BRKKY (566,567 = $236,$237).

17 $11 BRKKEY

Vlag voor de BREAK-toets. Hier staat een 0, als op BREAK gedrukt is. Iedere andere waarde betekent dat niet op BREAK gedrukt is.

18,19,20 $12,$13,$14 RTCLOK

Dit is de interne klok van de ATARI. Ze tikt op de frequentie van het lichtnet. Hoewel de frequentie van US-Amerikaanse apparaten op 60 ligt, werken de hier verkochte apparaten natuurlijk met 50 hertz. Als u met US-software werkt, waarbij gebruik wordt gemaakt van de interne klok, dan moeten de waarden met de factor 5/6 aan ons kalmere levensritme worden aangepast.

Bij het inschakelen van de computer begint de interne klok te lopen vanaf de waarde 0. Register 20 telt in een 50 hertz-ritme van 0 t/m 255, dus in een seconde van 0 t/m 49. Is de waarde 255 bereikt, dan verhoogt register 20 de waarde van register 19 met 1 en begint zelf weer bij 0. Als register 19 van 255 naar 0 verandert, verhoogt hij op zijn beurt de waarde van register 18 met 1.

Deze drie registers kunnen samen dus tot 16.777.215 tellen. Het duurt dus 355.544,32 seconden tot register 18 weer op 0 staat, dat is 5.592 minuten of ruim 93 uur of bijna 4 dagen, of, voor degenen die het graag precies willen weten: 3 dagen, 21 uur, 12 minuten, 24 seconden en 32/100 seconde. En zo kan een programma eruit zien voor een digitale klok van het merk ATARI:

```
0 REM ADR18.BEC
1 REM ********************************
2 REM *                              *
3 REM * ECHTE TIJD MET INTERNE KLOK *
4 REM *                              *
5 REM ********************************
10 FOR J=0 TO 2:POKE 18+J,0:NEXT J
20 ? CHR$(125):POKE 752,1
30 T=INT((PEEK(18)*65536+PEEK(19)*256+PEEK(20))/50)
40 D=INT(T/86400)
50 HH=INT(T/3600):H=HH-D*24
60 MM=INT(T/60):M=MM-D*1440-H*60
70 S=T-D*86400-H*3600-M*60
80 POSITION 10,11
90 ? D;" d : ";H;":";M;"'";S;CHR$(34);"   "
100 GOTO 30
```

programma ADR18.BEC

10: Bij de start van het programma moet de interne klok eerst op 0 worden teruggezet, want hij loopt al, sinds de computer werd aangezet.

20: wist het beeldscherm en onderdrukt de cursor.

30: berekent de actuele waarde van de registers 18 t/m 20, deelt deze waarde door 50 (hertz = tellen/seconde) en slaat het integere deel van het quotiënt op in T.

40: bepaalt, hoeveel hele dagen de waarde T bevat en slaat het resultaat in de variabele D op.

50: bepaalt op dezelfde manier het aantal gehele uren uit T. HH bevat het resultaat. Worden de uren in het berekende aantal dagen (= D*24) afgetrokken van HH, dan blijft het aantal uren van vandaag over. Dit wordt opgeslagen in H.

60: Op dezelfde manier wordt het aantal minuten berekend.

70: De seconden S kunnen worden gevonden, door het overeenkomstige aantal seconden in D, H en M af te trekken van het totale aantal seconden T.

80 en 90: Het resultaat wordt op het beeldscherm gePRINT en met

100: wordt het programma opnieuw doorlopen.

In plaats van dagen, uren, minuten en seconden uit te rekenen en te wachten, totdat de registers op 255 staan en weer op 0 springen, kunnen we deze klok ook slechts 24 uur laten tellen en dan RTCLOK weer op 0 zetten en in een variabele een dag erbij tellen. Zo kan deze klok eindeloos blijven lopen. Er zijn 24 uren voorbij, wanneer RTCLOK de waarde 24*60*60*50 = 4.320.000 = 65*65536 + 235*256 + 0*1 heeft. De opdracht luidt dus: IF PEEK (18)=65 THEN IF PEEK (19)=235 THEN IF PEEK (20)=0 THEN POKE 18,0:POKE 19,0:POKE 20,0

Ook een programma, dat de interne klok uitleest, gebruikt een bepaalde verwerkingstijd, waarin de interne klok natuurlijk rustig verder loopt. Maar bij zo'n kort programma is zelfs BASIC snel genoeg, om iedere hele seconde op het beeldscherm te tonen, ook als de registers 18 t/m 20 alleen gelezen worden, als het bovenstaande programma regel 30 verwerkt.

In feite is een seconde voor de ATARI een zeer lange tijd. Hij doorloopt in deze tijd zo ongeveer 40.000 machine-cycli, dus wanneer register 20 met 1 wordt verhoogd en daarmee 1/50 seconde registreert, heeft de processor alweer rond de 8.000 cycli doorlopen. Zelfs wanneer deze digitale klok als gedeelte van een langer programma wordt gebruikt, dan moet het wel heel lang zijn of ingewikkelde berekeningen (goniometrische functies, wortels, machten, enz.) bevatten, voordat de gePRINTe seconden een stap overslaan.

De interne klok hoeft niet speciaal in werkelijke tijd te worden omgerekend. We kunnen bijvoorbeeld alleen register 19 opvragen, dat bij benadering de tiende seconden telt, en de gevonden waarde omzet in een grafische balk, die langzaam langer of korter wordt. Of we laten iets gebeuren bij het bereiken van een opgegeven waarde:

```
IF PEEK(18)*65536 + PEEK(19)*256 + PEEK(20) = X THEN...
```

21,22 $15,$16 BUFADR

Tijdelijke wijzer van de S10 naar de actuele disk-buffer. Register voor het indirecte bufferadres.

23 $17 ICCOMT

Commando voor de C10-wijzer. Wordt gebruikt, om de offset in de instructie-tabel te vinden.

24,25 $18,$19 DSKFMS

Wijzer naar FMS (Disk File Manager System). Wordt JMPTBL genoemd door DOS.

26,27 $1A,$1B DSKUTL

Disk utilities wijzer. Wordt BUFADR genoemd door DOS.

28 $S1C PTIMOT

Printer-tijduitschakeling (timeout).

29 $1D PBPNT

Wijzer naar de printer-buffer. Wijst naar de actuele positie (byte) in de printer-buffer. Waarden mogelijk van 0 t/m de waarde uit PBUFSZ.

30 $1E PBUFSZ

Grootte van de printerbuffer in de actuele modus. De standaardgrootte van de buffer is 40 bytes.

31 $1F PTEMP

Wordt door de printer-driver gebruikt, om tijdelijk de waarde op te slaan van het teken, dat naar de printer gestuurd moet worden.

32 $20 ICHIDZ

Index-nummer voor de driver. Wordt door het OS als index gebruikt naar de tabel met de filenamen voor het geopende file. Als er geen files geopend zijn, staat hier 255.

33 $21 ICDNOZ

Nummer van de diskette-eenheid. Wordt MAXDEV genoemd door DOS om het maximale aantal diskette-eenheden aan te geven. Wordt geïnitieerd op 1.

34 $22 ICCOMZ

Door de gebruiker te bepalen commando code-byte, die aangeeft hoe de rest van de IOCB geformateerd moet worden en welk I/O-proces uitgevoerd moet worden.

35 $23 ICSTAZ

Status-byte van de laatste IOCB-behandeling. Wordt gegeven door het laatst aangesproken randapparaat.

36,37 $24,$25 ICBALZ/HZ

Buffer-adres voor het data-transport of adres van de filenaam voor instructies als STATUS, OPEN, enz.

38,39 $26,$27 ICPTLZ/HZ

Adres van de put-byte-routine. Een CLOSE-instructie zet deze wijzer naar CIO's 'IOCB not OPEN'.

40,41 $28,$29 ICBLLZ/HZ

Teller voor de bufferlengte bij PUT en GET. Wordt voor ieder overgedragen byte met 1 verminderd.

42 $2A ICAX1Z

Eerste byte van de hulpinformatie, die bij OPEN wordt gebruikt, om de soort van de benodigde file-toegang vast te leggen.

43 $2B ICAX2Z

Tweede byte van de hulpinformatie en CIO werk-variabelen.

44,45 $2C,$2D ICAX3Z/4Z

Wordt gebruikt bij de BASIC-instructies NOTE en POINT om de disk sectornummers over te sturen.

46 $2E ICAX5Z

Bepaalt de byte, in de door ICAX3Z/4Z vastgelegde sector, die verwerkt moet worden.

47 $2F ICAX6Z

Reserve byte. Wordt ook CIOCHR genoemd en gebruikt als tijdelijke opslagplaats voor het tekenbyte in een lopende PUT-bewerking.

48 $30 STATUS

Interne status-opslag. DE SIO-routines in ROM gebruiken deze byte, om de status van de actuele SIO-bewerking op te slaan.

49 $31 CHKSUM

Controletotaal (checksum), dat door S10 wordt gebruikt.

50,51 $32,$33 BUFRLO/HI

Wijzer naar de data-buffer. Wijst naar de byte, die moet worden weggezonden of ontvangen.

52,53 $34,$35 BFENLO/HI

Wijzer naar de eerst volgende byte na het einde van de data-buffer.

54 $36 CRETRY

Aantal herhalingen van een instructie door een apparaat, wanneer een poging mislukt is, om bijvoorbeeld een diskette te formateren of een sector te lezen. De standaard waarde is 13.

55 $37 DRETRY

Aantal herhalingen van een poging, een apparaat (bijvoorbeeld een diskette-eenheid) aan te sturen. De standaard waarde is 1.

56 $38 BUFRFL

Vlag die aangeeft of de data-buffer vol is. 255 betekent vol.

57 $39 RECVDN

Vlag die aangeeft of de data is ontvangen. 255 betekent ontvangen.

58 $3A XMTDON

Vlag die aangeeft of de data is verstuurd. 255 betekend verstuurd.

59 $3B CHKSNT

Vlag die aangeeft of de totaalcontrole verzonden is. 255 betekent verzonden.

60 $3C NOCHKSM

Vlag die aangeeft of er na de data een totaalcontrole volgt. 0 betekent controle, iedere andere waarde betekent geen controle.

61 $3D BPTR

Wijzer van de cassette-buffer. Een index naar de data van de dataregels, die net zijn gelezen of geschreven.

62 $3E FTYPE

Bepaalt de afstand tussen de datablokken op cassette.

63 $3F FEOF

Vlag van de cassette-driver, die EOF (End Of File) aangeeft.

64 $40 FREQ

In dit register slaat de cassette-driver op, hoeveel pieptonen de ingebouwde luidspreker moet geven: een voor weergave, twee voor opname.

65 $41 SOUNDR

Vlag voor het geluidssignaal bij datatransport. Een 0 maakt het transport geluidloos, iedere andere waarde maakt akoestische controle mogelijk. Kan bijvoorbeeld worden gebruikt, om bij cassettes met synchrone geluids- en data-sporen het digitale spoor tot zwijgen te brengen.

66 $42 CRITIC

Kritieke I/O vlag. Een waarde ongelijk aan 0 zorgt ervoor, dat een aantal OS-processen worden uitgezet. Dit kan nodig zijn, wanneer een groot aantal interrupts worden verwacht. Een neveneffect is, dat de toets-herhalingsfunctie ook wordt uitgezet. Bovendien verandert de toon van de zoemer (control-2). Het wordt echter afgeraden, CRITIC op een andere waarde dan 0 te zetten, omdat de interne timers dan in de war raken.

67-73 S43,$49 FMZSPG

Zero-page registers van FMS. Zeven bytes.

74 $4A CKEY

Controle-vlag voor het opstarten van de cassette-recorder vlak na het inschakelen (koude start). Controleert of START wordt ingedrukt tijdens het inschakelen. Als dat het geval is, wordt CKEY aangezet. Daardoor wordt het opstarten van de cassette-recorder mogelijk.

75 $4B CASSBT

Vlag voor het opstarten van de cassette-recorder. Bij Powerup probeert het OS gelijktijdig de cassette-recorder en de diskette-eenheid op te starten. Als het opstarten van de cassette-recorder niet lukt, dan wordt hier een 0 neergezet. Zie ook BOOT? (9 = $9).

76 $4C DSTAT

Register voor de beeldscherm- en toetsenbord-status. Wordt gebruikt door de display-driver. Hier wordt bijvoorbeeld ook een foute cursor-positie, te weinig geheugenruimte voor een grafiche modus of de toestand bij de onderbreking door BREAK aangegeven.

77 $4D ATRACT

Als een beeld te lang hetzelfde blijft, kan het inbranden in de forforlaag van de beeldbuis. De ATARI-constructeurs zijn van hun klanten blijkbaar enige nalatendheid gewend. Ze hebben namelijk de attract-modus ingebouwd. Wanneer gedurende een bepaalde tijd geen toets wordt ingedrukt, dan verandert deze modus met korte tussenpozen de waarden in de kleurenregisters en vermindert de totale helderheid van het beeldscherm.

Eigenlijk is dit alleen maar lastig. Andere huiscomputers hebben ook geen bescherming van de beeldbuis; er is echter nooit iets vernomen, over grote aantallen beeldbuizen die ineens zijn ingebrand. Bovendien is de attract-modus erg storend, omdat hij door andere invoer dan van het toetsenbord, bijvoorbeeld van stuurknuppels, niet wordt onderbroken.

ATRACT fungeert als vlag en timer voor de attract-modus. Het register wordt door IRQ (Interrupt Request) op 0 gezet, steeds wanneer een toets van het toetsenbord wordt ingedrukt. Het reageert niet op de functietoetsen SELECT, OPTION en START. Als tijdens de loop van een programma een willekeurige toets wordt ingedrukt, hoewel dit anders geheel zinloos zou zijn, dan wordt hier een 0 neergezet, ook als het lopende programma het toetsenbord helemaal niet aftast.

Zolang er geen toets wordt ingedrukt, wordt de waarde in dit register ongeveer iedere vijf seconden met 1 verhoogd door VBLANK. Als ATRACT de waarde 127 overschrijdt, krijgt de timer (bit 0 t/m 6) een overloop en zet de vlag (bit 7) aan: de decimale waarde van het register komt op 254 en de attract-modus begint te werken.

Het volgende demonstratie-programma laat zien hoe dit register werkt. Tegelijkertijd wordt de interne klok (RTCLOK) gebruikt, om de tijdsduur tot het in werking treden van de attract-modus te controleren:

```
0 REM ADR77.BEC
1 REM ********************************
2 REM *                              *
3 REM *  DEMO-PROGR. SCHERM WISSEN   *
4 REM *                              *
5 REM ********************************
10 DIM B(6)
20 POKE 19,0:POKE 20,0
30 ? CHR$(125):POKE 752,1
40 T=INT((PEEK(19)*256+PEEK(20))/50)
50 X=PEEK(77)
60 IF X>127 THEN R=10:POSITION 3,6:? "GESTOPT"
70 FOR J=0 TO 6:B(J)=0:NEXT J
80 M=INT(T/60)
90 S=T-M*60
100 Y=X
110 IF X>127 THEN X=X-128:B(6)=1
120 IF X>63 THEN X=X-64:B(5)=1
130 IF X>31 THEN X=X-32:B(4)=1
140 IF X>15 THEN X=X-16:B(3)=1
150 IF X>7 THEN X=X-8:B(2)=1
160 IF X>3 THEN X=X-4:B(1)=1
170 IF X>1 THEN X=X-2:B(0)=1
200 POSITION 2,2:? CHR$(17);CHR$(18);CHR$(18);CHR$(18);CHR$(18);CHR$(18);CHR$(18
);CHR$(18);CHR$(5)
210 POSITION 2,3:? CHR$(124):POSITION 10,3:? CHR$(124)
220 POSITION 4,3:? M
230 POSITION 6,3:? ":"
240 POSITION 7,3:? S;" "
250 POSITION 2,4:? CHR$(26);CHR$(18);CHR$(18);CHR$(18);CHR$(18);CHR$(18);CHR$(18
);CHR$(18);CHR$(3)
260 POSITION 0,16:? "----------------------------------------"
270 POSITION 5,20
280 ? "ATRACT : ";B(6);" ";B(5);" ";B(4);" ";B(3);" ";B(2);" ";B(1);" ";B(0);" "
;X;" = "
290 POSITION 33,20:? Y;" "
300 GOTO 40+R
```

programma ADR77.BEC

10: B$ moet de bits van ATRACT opslaan. De string wordt slechts op zes geDIMensioneerd, om plaats te besparen. Waarom zes genoeg is, zal hieronder worden verklaard.

20: Voor dit doel zijn de registers 19 en 20 van RTCLOK voldoende. Ze kunnen samen bijna twintig minuten aangeven. Hier worden ze op 0 gezet.

40: De totale tijd T in seconden wordt op de bekende manier berekend.

50: In X wordt de inhoud van adres 77 (ATRACT) opgeslagen.

60: Als X de kritische waarde 127 overschrijdt, dan wordt de interne klok gestopt. Dit gebeurt, door de variabele R op 10 te zetten, zodat aan het einde van het programma niet meer naar regel 40 wordt teruggesprongen, waar RTCLOK wordt gelezen, maar naar regel 50. Tegelijkertijd verschijnt onder de stopwatch het woord 'STOPPED'.

70: De zes elementen van de geïndiceerde variabele B worden op 0 gezet.

80 en 90: berekenen de tijd in minuten en seconden.

100: De waarde van X wordt in Y opgeslagen.

110 t/m 170: Dan worden uit de databyte X de bits 1 t/m 7 berekend en in B(J) opgeslagen. Omdat de desbetreffende waarden steeds van X worden afgetrokken, bevat X na regel 170 de toestand van bit 0. Dat verklaart regel 10. De variabele X hoeft overigens niet voor iedere programma-doorloop weer op 0 te worden gezet, omdat X telkens wordt gevuld met de waarde uit register 77.

200 t/m 260: PRINTen van het beeld. Omdat een naaldprinter de pseudo-grafische karakters niet kan omzetten zonder een speciale software aanpassing, worden voor de grafische elementen de CHR$-symbolen gebruikt.

220 t/m 240: PRINTen in het getekende raam de lopende tijd in minuten en seconden.

280 en 290: geven het actuele bit-patroon van ATRACT en de bijbehorende decimale waarde.

300: Eindeloze lus door een sprong terug.

Wordt tijdens het lopende programma het toetsenbord gebruikt, dan wordt de waarde in register 77 teruggezet, maar de stopwatch loopt verder. Door het opvragen van CH (764 = $2FC) kan het indrukken van een toets worden geregistreerd en door terug te springen naar regel 20 wordt de stopwatch dan op 0 gezet.

Hoeveel tijd ATRACT nodig heeft om de atract-modus in werking te stellen, kan met dit programma zelf worden vastgesteld. De waarde is aanmerkelijk groter dan in de diverse publicaties altijd wordt beweerd. Als we een spel of een grafisch programma schrijven, dat deze tijdgrens overschrijdt, zonder dat er een toets wordt ingedrukt (omdat bijvoorbeeld met stuurknuppels wordt gespeeld), dan moeten we voorkomen dat de attract-modus de kleuren steeds laat verspringen. Daartoe hoeft u alleen maar een POKE 77,0 in uw programma tussen te voegen.

78 $4E DRKMSK

Zolang de attract-modus niet aktief is, staat hier de waarde 254 en bij attract wordt deze 246 gezet, wat verhindert dat de kleurenhelderheid de 50% overschrijdt.

79 $4F COLRSH

Masker voor de kleurverandering. De kleurenregisters (vanaf 708 = $2C4) worden met de adressen 78 en 79 verbonden door een EOR-bewerking (exclusive OR).

80 $50 TMPCHR

Tijdelijk register, dat de display-driver gebruikt, om data naar het beeldscherm te sturen of daarvan te lezen.

81 $51 HOLD1

Hetzelfde als TMPCHR. Wordt ook gebruikt, om het aantal display-list registraties in op te slaan.

82 $52 LMARGN

Kolom-waarde voor de linkerrand van het beeld. O geeft de meest linkse kolom aan. De standaard-waarde is 2. Zie verder RMARGN.

83 $53 RMARGN

Kolom-waarde voor de rechterrand van het beeld. 39 geeft de meest rechtse kolom aan. De standaard-waarde is 39. De beide margin-registers kunnen (in beperkte mate) op willekeurige waarden worden gezet. Zo verandert de beeldscherm-editor gewoon mee, als op 82 en 83 dezelfde waarde wordt gePOKEd. Laad bijvoorbeeld eens een willekeurig programma in het geheugen, geef de opdracht POKE 82,n en POKE 83,n (waarbij n een waarde is van 0 t/m 39) en vraag een LIST. Wat gebeurt er?

Andere experimenten: de rechterrand staat links van de linkerrand, bijvoorbeeld POKE 82,37 en POKE 83,47. Of de rechterrand krijgt een waarde, die groter is dan de maximale waarde; POKE 82,37 en POKE 83,47. Alleen wanneer de linkerrand op een waarde groter dan 39 wordt gezet, zal gebruik worden gemaakt van de RESET-toets.

RESET zet opnieuw de standaard-waarden van de kantlijnen neer. Het veranderen van de schermranden heeft geen invloed op het weghalen of tussenvoegen van regels. 'Insert line' voegt altijd een fysieke regel toe, onverschillig hoe lang deze is. 'Delete line' haalt steeds een logische regel weg.

Ook de akoestische aanduiding voor het einde van een logische regel wordt beïnvloed door de kantlijnen. Deze richt zich naar het einde van de logische regel, die door het verkorten van de fysieke regel ook korter wordt. De zoemer klinkt dus altijd aan het einde van de derde regel op het scherm.

84 $54 ROWCRS

Aktuele regel-positie van de grafische- of de tekst-cursor. Afhankelijk van de grafische modus zijn waarden mogelijk van 0 (boven) t/m 19 of 191 (onder).

85,86 $55,$56 COLCRS

Aktuele kolom-positie van de grafische of de tekst-cursor. Afhankelijk van de grafische modus zijn waarden mogelijk van 0 (links) t/m 39 of 319 (rechts). Het nu volgende programma springt met het POSITION-statement naar de verschillende beeldposities en drukt daar de aktuele cursor-positie af. Het eerste teken van de afdruk staat op de positie waarnaar gesprongen is, de overige tekens volgen in de schrijfrichting:

```
0 REM ADR84.BEC
1 REM ********************************
2 REM *                              *
3 REM * DEMO-PROGR. CURSOR-POSITIES *
4 REM *                              *
5 REM ********************************
10 ? CHR$(125):POKE 82,0
20 FOR X=0 TO 30 STEP 6
30 FOR Y=0 TO 22 STEP 2
40 POSITION X,Y:? PEEK(85);",";PEEK(84);
50 NEXT Y
60 NEXT X
70 GOTO 70
```

programma ADR84.BEC

40: Adres 86 bevat het HI-byte van de kolom-positie van de cursor. Hij is altijd 0, behalve in GRAPHICS 8, waar hij de waarde 1 kan aannemen. Daarom is het hier voldoende om alleen adres 86 uit te lezen.

De inhouden van de registers worden in omgekeerde volgorde afgedrukt, want alle instructies, die betrekking hebben op de beeldscherm-positie, geven de waarden aan in de volgorde: kolom, regel. Dus: PLOT kolom, regel of POSITION X,Y. Omdat register 85 de kolom-positie bevat, wordt hij voor de komma gePRINT.

70: voorkomt het einde van het programma met READY.

Het is overigens zinloos, de volgende opdracht in te voeren:

POSITION n,m:? PEEK (85); ' ';PEEK(84)

Deze opdracht zal nooit de waarde n,m vinden, want als aan het eind van deze directe invoer op RETURN wordt gedrukt, springt de cursor onmiddellijk naar het begin van de volgende fysieke regel en deze positie-waarde vindt de gegeven opdracht dan ook.

Het BASIC-statement LOCATE onderzoekt niet alleen de aangestuurde beeldscherm-positie, maar beweegt ook de cursor een plaats naar rechts bij het volgende PRINT- of PUT-statement, als hij de waarden in ROWCRS en COLCRS verandert. Overigens, CHR$(253), de signaaltoon (BELL), is geen afdrukbaar karakter en heeft daarom ook geen invloed op de cursorpositie.

Omdat het LOCATE-statement zijn eigenaardigheden heeft, volgt hier een demonstratie-programma:

```
0 REM ADR85.BEC
1 REM ********************************
2 REM *                              *
3 REM * LOCATE-INVLOED OP DE CURSOR *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 0:? CHR$(125)
20 POSITION 30,20
30 ? PEEK(85);",";PEEK(84);
40 LOCATE 4,4,D
50 ? PEEK(85);",";PEEK(84),
60 LOCATE 0,0,X
70 POSITION 10,10:? "A"
75 FOR Z=0 TO 1000:NEXT Z
80 LOCATE 10,10,D:? D
90 FOR Z=0 TO 1000:NEXT Z
100 LOCATE 10,10,D:? D;" "
110 FOR Z=0 TO 1000:NEXT Z
120 LOCATE 10,10,D:? D;" "
200 GOTO 90
```

programma ADR85.BEC

10: Hoewel GRAPHICS 0 niet extra opgeroepen hoeft te worden, is dat statement hier noodzakelijk, omdat LOCATE alleen werkt, als daarvoor een GRAPHICS-statement is gegeven.
20: Hier wordt gesprongen naar de positie 30,20. Daardoor bevinden deze waarden zich in de cursor-registers, die in regel

30: gelezen en op het beeldscherm getoond worden. Let op de puntkomma aan het einde van het statement. Die zorgt ervoor, dat de cursor na de PRINT-opdracht niet naar het begin van de volgende regel springt, maar direct achter het afgedrukte blijft staan. Omdat echter in regel

40: een LOCATE-statement volgt, blijft de cursor optisch op positie 30,20 staan, terwijl hij zich feitelijk op 4,4 bevindt. LOCATE vindt de COLOR-waarde (grafische punt) of de ATASCII-waarde (karakter) en bewaart die in de aangegeven variabele.

50: Weer wordt het cursor-register afgedrukt: de komma aan het einde van de regel veroorzaakt een tabulator-sprong. Door de in regel

60: volgende LOCATE-instructie blijft de cursor optisch weer staan.
70: Voor een volgend experiment wordt hier op de plaats 10,10 een A gePRINT.

80: LOCATE vindt bij 10,10 de ATASCII-waarde van A (=65), die hier meteen wordt afgedrukt. De PRINT direct achter LOCATE verandert de waarde op de gelocaliseerde plaats.

90: Een korte rust voor de ogen.

100: Het LOCATE vindt deze keer bij 10,10 niet meer de waarde 65 (A), maar 32 (spatie).

110: Weer een ogenblik pauze.

120: Door achtereenvolgens LOCATE en PRINT is de waarde op die plaats weer veranderd. Nu zien we een geïnverteerde spatie en de waarde 160 wordt gevonden.

200: Dit spelletje gaat eindeloos verder, totdat we op BREAK drukken.

```
0 REM ADR87.BEC
1 REM ********************************
2 REM *                              *
3 REM *        GRAPHICS 7 1/2        *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 24
20 POKE 87,7
30 COLOR 3
40 FOR X=0 TO 159:PLOT X,0:NEXT X
50 COLOR 2
60 FOR Y=0 TO 95:PLOT 0,Y:PLOT 159,Y:NEXT Y
70 COLOR 3
80 PLOT 0,0:DRAWTO 159,95
90 COLOR 2
100 PLOT 0,95:DRAWTO 159,0
110 COLOR 1
120 FOR X=0 TO 159:PLOT X,95:NEXT X
200 POKE 89,PEEK(89)+15
210 COLOR 3
220 FOR X=0 TO 159:PLOT X,0:NEXT X
230 COLOR 2
240 FOR Y=0 TO 95:PLOT 0,Y:PLOT 159,Y:NEXT Y
250 COLOR 1
260 PLOT 0,0:DRAWTO 79,95:DRAWTO 159,0
270 PLOT 0,95:DRAWTO 79,0:DRAWTO 159,95
280 COLOR 2
290 FOR X=0 TO 159:PLOT X,95:NEXT X
300 GOTO 300
```

10: Grafische modus 8 zonder tekstraam wordt opgeroepen.

20: In DINDEX wordt echter de waarde 7 neergezet.

30 t/m 120: tekenen verscheidene lijnen in 'verschillende kleuren'. Natuurlijk kunnen in plaats van de FOR-NEXT PLOTs ook PLOT-DRAWTOs worden geschreven.

Let erop, dat de opgeroepen GRAPHICS 8 eigenlijk 320 PPL (pixels per line = beeldpunten per beeldlijn) toelaat, maar door het omPOKEn van adres 87 op 7 is de regel maar voor de helft (159) gevuld. Dat zijn de PPL van GRAPHICS 7. Omdat hier voor iedere pixel twee bits kleurinformatie ter beschikking staan (zie aanhangsel beeldscherm-geheugen voor verdere informatie), kunnen drie COLOR-waarden worden aangesproken.

Met het GRAPHICS-statement wordt de display-list opgeroepen. Bij GRAPHICS 8 is iedere modus-regel zo breed als een tv-beeldlijn. Het beeldscherm wordt dus in 191 modus-regels verwerkt. Wordt echter een beeldscherm-positie door PLOT of DRAWTO opgeroepen, die de toelaatbare waarde van GRAPHICS 7 overschrijdt, dan volgt

ERROR-141 (cursor-positie ontoelaatbaar), omdat dan de in DINDEX omgePOKEte waarde gaat werken. Met de regel- en kolom-waarden van GRAPHICS 7 kan hier dus alleen de bovenste helft van het beeldscherm gebruikt worden. Wat te doen?

Door het GRAPHICS 8 statement wordt genoeg geheugenruimte gereserveerd om beeldscherm-gegevens voor het totale scherm op te nemen. Door de hiervoor verrichte handelingen is het niet meer mogelijk, de onderste helft van de geheugenplaatsen met grafisch werkzame BASIC-statements te beschrijven.

Alle opdrachten, die betrekking hebben op beeldscherm-posities, berekenen intern de offset van de SM-byte, die de gegevens van het gezochte punt bevat. Dat gaat naar het formele SM-begin-adres + kolomwaarde /PPB (pixels per byte) + regelwaarde * bpl (bytes per line). Met de GRAPHICS 7 coördinaten is dus slechts een maximale offset mogelijk van 40 + 95*40 = 3840.

200: Om bij de beeldscherm-gegevens van de onderste helft van het beeldscherm te komen, wordt hier de wijzer naar het SM-begin-adres met de hierboven berekende 3840 bytes verhoogd.

210 t/m 290: Nu kunnen met de toegestane positie-waarden de grafische punten op de onderste beeldhelft worden gePLOT.

300: verhindert het einde van het programma. Omdat er geen tekstraam beschikbaar is, zou GRAPHICS 0 worden ingeschakeld, waardoor het opgebouwde grafische beeld zou worden uitgewist, alleen om te zorgen dat de computer zijn READY kan afgeven.

Als u nu nieuwsgierig bent geworden, wat er allemaal mogelijk is met ATARI in het grafische bereik, dan kunt u zichzelf nog dit kleine extraatje gunnen. Verander ADR87.BEC in de volgende regels:

```
0 REM ADR87A.BEC
1 REM ********************************
2 REM *                              *
3 REM *    EEN GRAFISCHE TOEGIFT     *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 24:A=2
15 TRAP 15:A=A+1
20 POKE 87,A
300 GOTO 15
```

programma ADR87A.BEC

(zie ook aanhangsel display list en beeldscherm-geheugen.)

88,89	$58,$59	SAVMSC

Wijzer naar het begin-adres van het beeldschermgeheugen (SM = screen memory). In deze byte liggen de gegevens voor de pixels, die in de linker bovenhoek van het scherm worden geschreven.

De grafische gegevens staan in het geheugen van SAVMSC t/m RAMTOP (40959 = $C000). Door het gebruiken van een GRAPHICS-instructie wordt de wijzer ingesteld op de daarvoor benodigde geheugenruimte. De wijzer kan echter door de gebruiker naar wens worden veranderd.

Het is bijvoorbeeld mogelijk, de gegevens van meerdere beeldscherm-inhouden in het geheugen op te slaan en dan door het verplaatsen van de SM-wijzer naar de verschillende begin-adressen, de beeldscherm-inhoud in een keer te verwisselen. Dat gaat echter alleen na iedere VBLANK, als de display list weer van voren af aan wordt gelezen, maar dit gebeurt wel vijftig keer per seconde.

In het volgende programma wordt GRAPHICS 3 (resp. 19) gebruikt, omdat deze modus de aangename eigenschap bezit, dat de SM slechts 240 bytes groot is, dus minder dan een pagina. Het is dus gemakkelijk om op onder elkaar liggende geheugenpagina's gegevens voor telkens één beeldscherm neer te zetten. Door het veranderen van de HI-byte van de SM-wijzer kan dan het beeld worden verwisseld. Het heeft hier echter geen zin, SAVMSC te veranderen, want de SM-wijzer is ook een deel van de display-list; waar de video-processor zijn beeldschermgegevens uit het geheugen haalt, daar haalt hij ook deze wijzer uit de DL:

```
0 REM PAGING.BEC
1 REM ********************************
2 REM *                              *
3 REM *  SCHERMBEELDEN VERWISSELEN   *
4 REM *                              *
5 REM ********************************
10 D=1
20 GRAPHICS 19
30 POKE 708,50
40 POKE 709,52
50 POKE 710,54
60 FOR P=0 TO 20
70 COLOR C+1
80 X=INT(RND(0)*12)
90 Y=INT(RND(0)*12)
100 PLOT 19-X,11-Y
110 PLOT 19+X,11-Y
120 PLOT 19+X,11+Y
130 PLOT 19-X,11+Y
140 PLOT 19-Y,11-X
150 PLOT 19+Y,11-X
160 PLOT 19+Y,11+X
170 PLOT 19-Y,11+X
180 NEXT P
200 D=D+1:C=C+1:IF C=3 THEN C=1
210 IF D=7 THEN 300
220 SM=PEEK(88)+PEEK(89)*256
230 FOR K=0 TO 239
240 POKE SM-D*256+K,PEEK(SM+K)
250 NEXT K
260 GOTO 20
300 DL=PEEK(560)+PEEK(561)*256
310 Z=PEEK(DL+5)
320 FOR J=2 TO 6
```

```
330 POKE DL+5,Z-J
340 FOR W=0 TO 50:NEXT W
350 NEXT J
360 FOR J=5 TO 2 STEP -1
370 POKE DL+5,Z-J
380 FOR W=0 TO 50:NEXT W
390 NEXT J
400 GOTO 320
```

programma PAGING.BEC

30 t/m 50: De kleurwaarde voor de COLOR-instructie worden in de kleurenregisters gezet (zie COLORO, 708 = $2C4 en volgende).

60 t/m 180: De voorfilm. Het programma toont een paar willekeurige beelden in wisselende kleuren.

220 t/m 250: De data-bytes uit het beeldschermgeheugen worden in volgorde gelezen (PEEK SM+K) en in een hoger liggende pagina (SM - D*256) in dezelfde volgorde weer neergezet (POKE SM - D*256 + K). Hierbij bepaalt D hoeveel pagina's hoger de gegevens telkens worden gePOKEd en K telt de 240 bytes van GRAPHICS 3 af.

300: berekent het begin-adres van de display-list (zie SDLSTL, 560,561 = $230,$231).
310: De HI-byte van de SM-wijzer in de DL is de vijfde byte (de nulde meegeteld!).

320 t/m 390: verzetten de SM-wijzer telkens na een korte wachtlus. Zo verwisselen de in het bovenste gedeelte van het programma geproduceerde vijf beelden steeds in een keer op het beeldscherm.
Het is niet moeilijk te bedenken hoe men op deze manier tekenfilmachtige animaties kan maken. Zolang men zich tevreden stelt met grafische modus 3 en het BASIC-programma zelf niet te lang is, heeft men (bij de 800XL) plaats voor ongeveer honderd beelden. Beperkt men zich tot een niet-knippervrije beeldsnelheid van 16 projecties per seconde, dan kan men toch nog acht seconden echte filmtijd met de computer opnemen.

Verdere grafische spelletjes staan in het hoofdstuk beeldschermgeheugen.

90 $5A OLDROW

De vorige regel van de grafische cursor wordt door ROWCRS (84 = $54) vernieuwd, voordat deze een nieuwe waarde aanneemt. Zo staan voor DRAWTO of de FILL-instructie (X10 18) in OLDROW de begin- en in ROWCRS de eind-coördinaten ter beschikking.

91,92 $5B,$5C OLDCOL

Dezelfde functie als OLDROW, maar dan voor de kolompositie, dus twee bytes groot (320 kolommen in GRAPHICS 8).

93 $5D OLDCHR

Bevat de waarde van het karakter onder de cursor en wordt gebruikt om het karakter te vernieuwen, als de cursor wordt verplaatst.

94,95 $5E,$5F OLDADR

Bevat het beeldschermgeheugen-adres van de huidige cursorpositie. Wordt samen met OLDCHR gebruikt, om het karakter onder de cursor te vernieuwen, als de cursor wordt verplaatst.

96 $60 NEWROW

Regel-coördinaat van het punt, waarnaar DRAWTO of X10 18 (FILL) moet gaan.

97,98 $61,$62 NEWCOL

Kolom-coördinaat van het punt, waarnaar DRAWTO of X10 18 moet gaan. NEWROW en NEWCOL krijgen hun waarden van ROWCRS en COLCRS (84-86 = $54-$56), zodat tijdens de DRAWTO- of FILL-routine in deze registers alvast de volgende cursor-coördinaten opgeslagen kunnen worden.

99 $63 LOGCOL

Geeft de positie van de cursor binnen de logische regel aan. Een logische regel kan bij ATARI maximaal drie GRAPHICS 0-regels lang zijn, dus 3*40 = 120 karakters. LOGCOL kan dus waarden aannemen van 0 t/m 119. Dit register wordt door de display-driver gebruikt, bijvoorbeeld om het geluidssignaal 'regel vol' te geven.

100-105 $64-$69 diversen

Tijdelijke opslagplaatsen, die de display-driver gebruikt, om diverse waarden op te slaan.

106 $6A RAMTOP

Wijst de beschikbare hoeveelheid gebruikers-RAM in pagina's (= 256 bytes) aan. PEEK(106)*256 geeft het hoogste vrije adres.

Dit register kan worden gebruikt om een beschermd geheugengebied te maken, bijvoorbeeld voor machinetaal-routines, vrij definieerbare tekensets, player-missile-data of alternatief beeldschermgeheugen. De volgende instructie beschermt het gewenste gebied:

POKE(106),PEEK(106)-n

n is het aantal geheugenpagina's, dat gereserveerd moet worden. De waarde PEEK(1906)*256 mag nooit kleiner worden dan PEEK(144) + PEEK(145)*256 (MEMTOP).

Als RAMTOP wordt veranderd, om geheugen te reserveren, moet onmiddellijk daarna een GRAPHICS-instructie volgen. Daarbij kan gerust de grafische modus worden opgeroepen, die toch al ingeschakeld is. Daardoor worden de display-list en de grafische gegevens met de nieuwe RAMTOP meegenomen.

(#) Een GRAPHICS- of CLEAR-instructie wist 64 bytes boven RAMTOP. Door het scrollen van het tekstraam in een grafische modus worden 800 bytes door RAMTOP overschreven, want het tekstraam scrollt in werkelijkheid de totale GR.0 beeldscherm-gegevens, behalve de vier regels van het tekstraam. Dat zijn dus 20 regels van ieder 40 bytes = 800 bytes. Als geheugenruimte gereserveerd moet worden, dan moet er in het gegeven geval dus rekening mee worden gehouden, dat een overeenkomstig groter bereik wordt beschermd, zodat achter de te beschermen gegevens een veiligheidszone overblijft die groot genoeg is.

Wanneer met GRAPHICS 7, 8, 9, 10, 11, 14 of 15 wordt gewerkt, dan kan het tot storingen leiden, als RAMTOP zo wordt verschoven, dat de display-list en de SM-gegevens een 4k-grens kruisen. Als er dan vreemde effecten op het beeldscherm optreden, dan kan het verholpen worden door RAMTOP te verlagen met een veelvoud van 16 (4*4 pagina's = 4k).

Een andere mogelijkheid om een stuk geheugenruimte te reserveren is beneden MEMLO '743,744 = $2E7,$2E8).

107 $6B BUFCNT

Wordt door de beeldscherm-editor gebruikt als teller voor de lengte van de logische regel, waarop de cursor staat.

108,109 $6C,$6D BUFSTR

Tijdelijke opslag. Geeft het karakter waarnaar door BUFCNT wordt gewezen.

110 $6E BITMSK

Wordt als bit-masker gebruikt door de OS-display-driver bij bit-mapping-routines. Wordt door de display-driver ook gebruikt als tijdelijke opslagplaats.

111 $6F SHFTAMT

Pixel-instelling. Berekent de plaats van een pixel binnen een byte grafische informatie.

112,113 $70,$71 ROWAC

Accumulator voor de regel-besturing bij het plotten.

114,115 $72,$73 COLAC

Accumulator voor de kolom-besturing.

116,117 $74,$75 ENDPT

Eindpunt van de lijn, die getrokken moet worden. Bestuurt het PLOTten van punten van een lijn.

118 $76 DELTAR

Regelverschil = het absolute resultaat van NEWROM – ROWCRS.

119,120 $77,$78 DELTAC

Kolomverschil = het absolute resultaat van NEWCOL – COLCRS. DELTAR en DELTAC worden gebruikt, om de stijging te berekenen van de lijn, die gePLOT moet worden.

121,122 $79,$7A KEYDEF

Wijzer naar de definitie-tabel voor het omzetten van de toetscodes in ATASCII-waarden. (#: Zie 760 en 761 = $2F8 en $2F9.)

123 $7B SWPFLG

Vlag voor de cursor-besturing bij een gesplitst beeldscherm (met tekstraam).

124 $7C HOLDCH

Hier wordt de waarde van een karakter neergezet, voordat de CONTROL- of SHIFT-logica daarop wordt toegepast.

125 $7D INSDAT

Tijdelijke geheugenplaats, die door de display-driver benut wordt voor het teken onder de cursor en de regel-einde-herkenning.

126,127 $7E,$7F COUNTR

Begint met de grootste waarde uit DELTAR of DELTAC. Bepaalt het aantal punten dat gePLOT moet worden, om een lijn te trekken. Wordt na ieder gePLOT punt met 1 verminderd. Is deze byte 0 geworden, dan is de lijn klaar.

128,129 $80,$81 LOMEM

Wijzer naar het laagste adres, dat voor een BASIC-programma ter beschikking staat. De eerste 256 bytes dienen als token-uitvoer-buffer. Tokens zijn numerieke uitdrukkingen voor commando's, operatoren en functies, en een byte groot.

De waarde hier wordt gebruikt door MEMLO bij het initiëren van de computer of bij een NEW-commando, maar niet bij RESET. Als MEMLO wordt veranderd, moet dus ook de waarde in LOMEM worden aangepast.

Wordt een programma geSAVEd, dan worden eerst de zeven BASIC-wijzers van LOMEM t/m STARP weggeschreven. De waarde van LOMEM wordt daarbij van al deze 2-byte registers afgetrokken. De eerste twee bytes die worden weggeschreven, zijn dus 0-en. Na de wijzers worden de tabellen met de namen en de waarden van de variabelen weggeschreven en dan het in tokens veranderde BASIC-programma.

Bij LOAD wordt de waarde van MEMLO bij elk van de zeven wijzers opgeteld en dan worden deze waarden op de adressen van de wijzers op de zero-page neergezet. Daarna worden boven MEMLO 256 bytes voor de token-uitvoer-buffer gereserveerd en vlak achter deze buffer wordt het BASIC-programma neergezet.

Met de zeven BASIC-wijzers is natuurlijk een hoop onheil aan te richten. Als vreemde troepen het land binnen marcheren, dan worden alle wegwijzers en straatnaamborden gedemonteerd of verdraaid. Wanneer gevreesd wordt voor de invasie van snuffelaars in het eigen waardevolle programma, dan kunt u deze wijzers veranderen, om de indringer bij de neus te nemen. Maar juich niet te vroeg, want een echte 'hacker' zal zich door zo'n eenvoudige truc niet laten misleiden. Veranderingen hier bieden slechts een beperkte bescherming.

Zoals hiervoor al werd verklaard, worden bij SAVE de waarden van de wijzers mee opgeslagen. Maar niet bij LIST. Daarom bieden veranderingen van de BASIC-wijzers in eerste instantie een LIST-bescherming. Het programma kan weliswaar met SAVE en LOAD nog

probleemloos worden gecopieerd, maar LISTings nemen een min of meer exotische vorm aan.

Het volgende programma toont aan, hoe met een enkele BASIC-regel enige verwarring kan worden gesticht:

```
0 REM ADR128.BEC
1 REM ********************************
2 REM *                              *
3 REM *     LIST-VERWARRING          *
4 REM *                              *
5 REM ********************************
10 A=A+10
20 B=10:C=5:D=7.5
30 F=A+B:G=A*C:H=A/D
40 ? F;G;H
50 GOTO 10
100 REM XIO33,#1,0,0,"D:GEHEIM"
110 REM Y=PEEK(128)+PEEK(129)*256+3:POKE128,Y-INT(Y/256)*256:POKE129,INT(Y/256)
:SAVE"D:GEHEIM"
120 REM RUN"D:GEHEIM"
```

```
0 REM ADR128.BEC
1 REM ********************************
2 REM *                              *
3 REM *      LIST-VERWARRING         *
4 REM *                              *
5 REM ********************************
10 '-D='-D+10
20 F=10:G=5:H=7.5
3DRT20vBEC:
='-D#G:%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
='-D/H
4DRT28.BEC
,%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

50 GOTO 10
100 REM XIO33,£1,0,0,"D:GEHEIM"
110 REM Y=PEEK(128)+PEEK(129)*256+3:POKE128,Y-INT(Y/256)*256:POKE129,INT(Y/256):
SAVE"D:GEHEIM"
120 REM RUN"D:GEHEIM"
```

programma ADR128.BEC

10 t/m 50: bevatten het absoluut waanzinnige BASIC-programma, dat in ieder geval tegen nieuwsgierige blikken beschermd moet worden, omdat het programmeertechnieken gebruikt, die op het gebied van de BASIC-programmering revolutionair genoemd mogen worden.

110: Verwijder hier het regelnummer en de REM en geef de regel als directe invoer. Er wordt dan een file op de diskette geschreven met de mooie naam 'SECRETE'. Daarbij wordt zoals verklaard van alle wijzers de gekke waarde LOMEM afgetrokken.

120: Verwijder ook hier regelnummer en REM voor een directe invoer. Natuurlijk kunt u ook eerst LOAD en dan RUN invoeren, om het wat degelijker te doen. Het resultaat op het beeldscherm laat ondubbelzinnig zien, dat het programma foutloos werkt. Waarom ook niet?! Maar vraag de computer nu eens om een LIST! Is dat een LIST?

Hetzelfde kan overigens ook worden geprobeerd met LIST'C: of LIST'D:. En de printer is heel blij met een L.'P:'!

100: Om het vreemde programma van de waardevolle diskette te verwijderen hoeft alleen deze regel direct ingevoerd te worden. (Regelnummer en REM verwijderen en RETURN.) Deze regel bespaart ons de moeite, om naar DOS te moeten gaan en optie D te moeten oproepen en – als er geen MEM.SAVE op de diskette staat – na optie B-, ook nog het programma opnieuw met 'D:' in de computer te moeten laden.

130,131 $82,$83 VNTP

Deze wijzer wijst naar de eerste byte in de tabel met de variabelen-namen. Deze worden in de volgorde opgeslagen zoals ze door de gebruiker worden benoemd. Ook iedere keer dat er een typefout wordt gemaakt en de typefout heeft toevallig een gedaante die een variabele ook kan hebben, dan belandt deze typefout in de variabelen-tabel.

De variabelen-namen worden in ATASCII-formaat opgeslagen. Geïndiceerde variabelen worden samen met het haakje openen opgeslagen en zijn daar dus aan te herkennen. String-variabelen eindigen met het $-teken. Variabelen-namen kunnen in principe zo lang zijn als een logische regel. De ATARI-computer is in staat al deze variabelen van elkaar te onderscheiden. Hij kijkt dus niet, zoals de meeste home-computers, alleen naar de eerste twee of drie karakters van een variabele. Om te herkennen, waar een variabelen-naam ophoudt en de volgende begint, wordt bij het laatste teken van een variabele-naam het MSB (most significant bit = bit 7) aangezet. Dit bit heeft de decimale waarde 128. Voor de beeldscherm-editor betekent dat, dat hij het karakter invers (negatief) weergeeft.

In de tabel kunnen maximaal 128 variabelen-namen worden ingevoerd. Als we een heel lang programma schrijven, kan dat wel eens moeilijk worden. Daarom volgen hier een paar waardevolle tips.

1: Kies alle variabelen-namen zo kort mogelijk. Lange variabelen-namen gebruiken veel geheugenruimte en maken het programma langzamer. Maar langere variabelen-namen zijn bij het schrijven van het programma gemakkelijker te herkennen. Dus we schrijven het programma met gewone volledige woorden als variabelen-namen en we veranderen deze namen pas als het programma af is. Nee! Natuurlijk niet in het programma zelf, regel voor regel om er hier en daar een over het hoofd te zien en dan urenlang met de ERRORs te knoeien, maar direct in de tabel met de variabelen-namen.

2: Als er geen plaats meer is in de tabel met de variabelen-namen, dan helpt vaak de volgende truc. De kans is namelijk groot, dat in de ijver van het programmeren de een of andere naam in de tabel belandt, die uiteindelijk helemaal niet wordt gebruikt. Zet het programma met LIST op diskette, geef het commando NEW, dat ook de tabel uitwist en

ENTER het programma weer. Nu zijn alle ongebruikte variabelen-namen uitgewist. We moeten wel LISTen. SAVE slaat namelijk de hele tabel ook op en laadt hem weer bij LOAD.

3: Gebruik hulp-variabelen meerdere keren. Er is geen enkele reden, om in de eerste FOR-NEXT lus een bepaalde variabele te gebruiken en in de volgende FOR-NEXT lus een andere. We kunnen zonder enig bezwaar voor alle FOR-NEXT lussen, die niet genest zijn en dus niets met elkaar te maken hebben, dezelfde variabele gebruiken als lusteller.

4: Als het programma niet zoveel variabelen heeft, dan is het nuttig om voor alle numerieke waarden die vaker dan drie keer gebruikt worden een variabele te kiezen. De variabele gebruikt een keer plaats in de namen-tabel en een keer in de tabel met de variabelen-waarden. In het eigenlijke BASIC-programma gebruikt een variabele slechts zoveel bytes als de naam lang is, dus in het gunstigste geval één. Iedere numerieke waarde neemt echter in principe zes bytes in beslag. Schrijf dus GOTO J in plaats van GOTO 300, enzovoorts. Hoe vaker een numerieke waarde in een programma voorkomt, des te meer geheugenruimte bespaard kan worden door een variabele te gebruiken. Alleen als in zo'n programma naderhand de regelnummers met een renumber-functieprogramma worden veranderd, geeft het problemen, omdat geen enkel renumber-programma niet-numerieke sprongen omrekent.

Laten we dan nu eens de variabelen-tabel gaan bekijken:

```
0 REM ADR130.BEC
1 REM ********************
2 REM *                  *
3 REM * VARIABLEN-TABEL  *
4 REM *                  *
5 REM ********************
10 A=PEEK(130)+PEEK(131)*256:E=PEEK(132)+PEEK(133)*256:FOR I=4 TO E-A-1:O=P
EEK(A+I):IF O)128 THEN O=O-128:GOTO 30
20 ? CHR$(O);:GOTO 40
30 ? CHR$(O),:GOTO 40
40 NEXT I:END
100 B=1:C=2:D=3:DIM F$(1),G(2),H$(3)
200 REM V=PEEK(130)+PEEK(131)*256+4:POKE V+X,CHR+128
300 REM X=0:CHR=64
```

programma ADR130.BEC

10: In de variabele A wordt de wijzer opgeslagen naar het begin van de tabel met de variabelen-namen (adressen 130,131 = $82,$83). In E wordt de wijzer opgeslagen naar het einde daarvan (adressen 132,133 = $84,$85). De volgende FOR-NEXT lus leest nu alle bytes, die tussen A en E liggen. Omdat dit kleine programma voor het uitlisten van de variabelen-namen zelf variabelen bevat (A,E, 1 en 0), begint de lus met de waarde 4. Zet hier eens een 1 of een 0 (!) neer. De IF-voorwaarde controleert, of het gelezen byte het laatste teken van een variabele-naam is. Alleen als dat het geval is, wordt naar regel 30 gesprongen, waar de komma voor een tabulator-sprong zorgt en daarmee de variabelen scheidt.

20: Omdat de variabelen-namen in de tabel in ATASCII-code zijn opgeslagen, is het CHRS-statement voldoende, om de variabelen leesbaar te maken. De puntkomma rijgt de tekens direkt aan elkaar.

30: Als regel 20, maar met tabulator-sprong.

100: stelt het BASIC-programma voor, dat hier slechts uit zes variabelen bestaat.

300: Wanneer nu de exacte plaats van een variabele in de tabel bekend is, dan kan door het POKEn van de desbetreffende ATASCII-waarde de inhoud van de geheugenplaats en daarmee de variabele-naam worden veranderd. De eerste (eigenlijke nulde) variabele van het BASIC-programma is B in regel 100. Deze variabele moet nu de naam T krijgen. De ATASCII-waarde van T is 84. Verwijder nu in deze regel het regelnummer en de eerste REM en geef de regel als direkte invoer. X geeft de plaats aan, die veranderd moet worden. Daarbij moet ieder teken, dus niet alleen iedere variabele, geteld worden. B is het nulde teken in de tabel, daarom krijgt X de waarde 0. In CHR wordt de ATASCII-waarde van het nieuwe teken opgeslagen.

200: Hier vindt de omwisseling van de variabelen plaats. In V wordt de wijzer (VNTP) opgeslagen naar het begin van de variabelen-tabel. Hier wordt nog de waarde 4 erbij opgeteld, omdat het voorgaande gedeelte van het programma in de regels 10 t/m 40 vier variabelen bevat, zodat daar overheen wordt gesprongen. Het eerste byte, waarnaar VNTP wijst, is overigens altijd 0. Als we het door de hiervoor gegeven routine laten afdrukken, verschijnt eerst een hartje. (Dat is toch leuk, nietwaar?)

Op adres V (het eerste teken van de eerste variabele in de tabel) + X (offset van het karakter, dat veranderd moet worden) wordt de ATASCII-waarde CHR van het nieuwe teken gePOKEd. Omdat dit teken in het voorbeeld het laatste teken van de variabele is, moet ter herkenning van het einde van de variabele bit 7 nog worden aangezet (CHR + 128).

Verwijder nu ook in deze regel het nummer en de REM voor direkte invoer en geef dan een RUN, om de variabelen-namen-tabel nog eens af te drukken. En zie daar...

Als we vervolgens F$ in V$ willen veranderen, dan moeten we voor X de waarde 3 en voor CHR de waarde 86 invullen, RETURN en klaar.

Op basis van deze programma-regels is eenvoudig een functie-programma op te stellen, dat met het simpel invoeren van de waarden voor X en CHR en de vraag, of het om het laatste teken gaat of niet, het veranderen van variabelen-namen mogelijk maakt. Het is zinvol, zo'n hulpprogramma met heel hoge regelnummers (bijvoorbeeld van 32700 t/m 32767) te schrijven, zodat het, indien nodig, onder een bestaand BASIC-programma ge-

laden kan worden. Daarbij moet er wel op gelet worden, dat in het bron- en het functie-programma niet dezelfde variabelen voorkomen.

Alleen met grote moeite is het probleem, om de naam van een variabele te verlengen, op te lossen, want dat betekent dat alle variabelen, die daaronder in de tabel staan, in het geheugen naar onderen verplaatst moeten worden. Daarentegen is het gemakkelijk, variabelen-namen te verkorten. POKE gewoon 0-en op de desbetreffende plaatsen, respectievelijk 128 als het laatste teken van een variabele geschrapt moet worden. Daardoor alleen wordt weliswaar nog niet het aantal tekens in de variabelen-tabel verminderd, er zijn immers nog evenveel bytes bezet, maar ook dat is op te lossen met LIST'D:', NEW en ENTER'D:'.

Als het nodig is, uw programmeerkunsten tegen nieuwsgierige LIST-blikken te beschermen, dan kan ook de variabelen-tabel u een goede dienst bewijzen. Wat denkt u bijvoorbeeld van een LISTing, waarin alle variabelen hetzelfde zijn:

```
0 REM ADR131.BEC
1 REM ********************************
2 REM *                              *
3 REM * VARIABELEN-TABEL VERSTOREN   *
4 REM *                              *
5 REM ********************************
10 A=PEEK(130)+PEEK(131)*256
20 Z=PEEK(132)+PEEK(133)*256
30 FOR J=A TO Z:POKE J,129:NEXT J
100 B=10:C=30:D=2.5
110 F=C-B:G=B/D:H=D*C
120 ? F,G,H
```

programma ADR131.BEC

10: De wijzer VNTP naar het begin van de variabelen-namen-tabel wordt in A opgeslagen.

20: De wijzer VNTD naar het eind van de variabelen-namen-tabel wordt opgeslagen in Z.

30: Nu moeten alleen nog alle bytes tussen A en Z met een en dezelfde ATASCII-waarde worden gevuld.

Kies daarvoor een ATASCII-waarde plus 128, om alle variabelen in de LISTing door hetzelfde teken te vervangen. Erg leuk is ook, om de waarde 128 te POKEn, dan lossen namelijk alle variabelen in de LISTing in het niets op. Ook waarden kleiner dan 128 zijn grappig, omdat er dan geen variabelen-einde meer is. Kijk zelf maar eens hoe vreemd zo'n LISTing eruit ziet! Maar de mooiste van de variabelen-substituten is wel het getal 155. Dat is de ATASCII-waarde voor RETURN. Iedere variabele wordt door RETURN vervangen. Dat geeft zelfs bij een klein programma heerlijk lange LISTings zonder een enkele variabele-naam. Veel plezier bij het verwarringsspel!

132,133	$84,$85	VNTD

Deze wijzer wijst het eind aan van de tabel met variabelen-namen. Als er 128 variabelen-namen in de tabel zijn ingevoerd, dan wijst hij naar het eerste byte na het laatste teken van de laatste variabele. Staan er minder variabelen in de tabel, dan wijst hij naar een lege byte (0; ATASCII 0 = hartje), die direkt na de laatste variabele ligt, dus naar de plaats, waar de volgende variabele-naam opgeslagen moet worden.

Een verdere doeltreffende LIST-beschermingsroutine bestaat hieruit, de beide wijzers VNTP en VNTD te verzetten. Welke waarde u in de wijzer-bytes POKEt, wordt geheel aan uw eigen fantasie overgelaten. In het volgende voorbeeld is een 0 gekozen:

```
0 REM ADR132.BEC
1 REM ********************************
2 REM *                              *
3 REM * VAR.-TABEL-WIJZER VERSTOREN *
4 REM *                              *
5 REM ********************************
10 FOR J=0 TO 3:POKE 130+J,0:NEXT J
100 A=10:B=30:C=7.5
110 F=A+B:G=B/C:H=C*A
120 ? F,G,H
```

programma ADR132.BEC

10: Deze onschuldig uitziende regel is voldoende om de beide wijzers te verzetten.

100 t/m 120: Dit surrogaat BASIC-programma staat hier alleen, om de indrukwekkende gevolgen van de verwarrende POKEs te laten zien bij een LISTing.

134,135 $86,$87 VVTP

Wijzer naar de variabelen-waarde-tabel. Daarin worden de actuele waarden opgeslagen van alle variabelen, die in de variabelen-namen-tabel zijn ingevoerd. Iedere variabele gebruikt in de waarde-tabel acht bytes. De eerste byte geeft aan, om wat voor soort variabele het gaat.

00 betekent een scalar (niet gedimensioneerde numerieke waarde)

65 betekent een geDIMensioneerd array (geïndiceerde variabele)

128 betekent een ongeDIMde string

129 betekent een geDIMde string

De tweede byte bevat het nummer van de variabele, een getal van 0 t/m 127.

Bij een scalar bevatten de overige zes bytes de BCD-constante (binary coded decimal = binair gecodeerd decimaal getal).

Bij een array geven byte 3 en 4 de offset aan vanaf STARP '140,141 = $8C,$8D), dus de plaats in de string- en array-tabel. Byte 5 en 6 bevatten de eerste DIMensionering + 1, byte 7 en 8 de tweede + 1. Is een variabele slechts enkelvoudig geDIMensioneerd, dan hebben byte 7 en 8 de waarde 0.

Bij string-variabelen geven byte 3 en 4 eveneens de offset aan vanaf STARP. Byte 5 en 6 bevatten de huidige lengte van de string en byte 7 en 8 bevatten de geDIMde lengte.

Er bestaat een eenvoudige techniek om een string te vullen met dezelfde tekens. We hoeven daarvoor alleen op de eerste plaats van de string het gewenste teken neer te zetten, dan op de laatste plaats van de string de string zelf en tenslotte nog op de tweede plaats in de string eveneens de string zelf. Onmiddellijk is dan de gehele string gevuld met het ene, als eerste opgeslagen teken:

```
0 REM ADR135A.BEC
1 REM ********************************
2 REM *                              *
3 REM * STRING IN GROEPEN OPVULLEN   *
4 REM *                              *
5 REM ********************************
10 DIM TX$(264)
20 TX$(1)="!£?&"
30 TX$(261)=TX$
40 TX$(5)=TX$
100 ? TX$
```

programma ADR135.BEC

10: Hier wordt geDIMensioneerd.

20: Het eerste teken wordt opgeslagen.

30: En hier het laatste. Er moet echter wel TX$ worden neergezet, dus niet bijvoorbeeld '?'!

40: Nu nog de tweede plaats vullen en daar is de string.

100: Hier zien we het resultaat.

Het is niet noodzakelijk, de gehele string op deze manier te vullen. We kunnen ook slechts een deel laten volschrijven:

programma ADR135X.BEC

10: Hier wordt de totale hoeveelheid nog beschikbare geheugenruimte (FRE(0) gereserveerd voor TX$. Waarom ook niet?

30: Dan worden echter alleen de eerste 9999 elementen gevuld met '!'.

```
0 REM ADR135X.BEC
1 REM ********************************
2 REM *                              *
3 REM *    GEHEUGEN OVERBELASTEN     *
4 REM *                              *
5 REM ********************************
10 DIM TX$(FRE(0)-1)
20 TX$(1)="!"
30 TX$(9999)=TX$
40 TX$(2)=TX$
100 ? TX$
```

Helaas is er bij dit programma in de gegeven vorm een klein probleem. Omdat TX$ het gehele geheugen in beslag neemt, is een afdruk op het beeldscherm niet meer mogelijk. We moeten ons in regel 10 dus een klein beetje beperken. Maar misschien brengt dit speciale effect de creatievelingen onder de lezers op het idee, hoe men met het opvullen van het vrije geheugen een programma zo kan ontwerpen, dat het gewoon werkt, maar dat eventuele niet gewenste experimenten van de gebruiker met een geheugenoverloop-fout worden gestraft.

Maar laten we weer terug gaan naar de serieuze programmeer-vragen. Het hierboven beschreven effect werkt niet alleen met een enkel teken, maar ook met een hele rij tekens. Daarbij wordt op de eerste plaats van de string de string zelf neergezet en wel zo, dat het laatste teken van de rij de laatste te vullen plaats bezet. Dus als de string volledig opgevuld moet worden: stringlengte min de lengte van de rij tekens. Tenslotte moet dan op de plaats die volgt na de op de eerste plaats opgeslagen tekens, dus op de plaats 1 + lengte van de rij tekens, eveneens de string worden neergezet:

```
0 REM ADR135.BEC
1 REM ********************************
2 REM *                              *
3 REM *      STRING OPVULLEN         *
4 REM *                              *
5 REM ********************************
10 DIM TX$(266)
20 TX$(1)="?"
30 TX$(266)=TX$
40 TX$(2)=TX$
100 ? TX$
```

programma ADR135A.BEC

10: TX$ wordt op 264 geDIMensioneerd.

20: Op de plaatsen 1 t/m 4 wordt de rij tekens neergezet.

30: Dan wordt TX$ op de plaatsen 261 t/m 264 neergezet.

40: En tenslotte nog op plaats 5 respectievelijk 5 t/m 8.

136,137	$88,$89	STMTAB

Wijzer naar de statement-tabel, dus naar het eigenlijke BASIC-programma in de vorm van tokens. De eerste twee bytes van een in tokens omgezette BASIC-regel bevatten het regelnummer (LO, HI*256). Daarna volgt een tel-byte, die aangeeft hoeveel geheugenplaatsen de BASIC-regel in beslag neemt. De waarde van deze tel-byte wordt pas bepaald, wanneer de omzetting in tokens klaar is. Deze omzetting vindt plaats in de tokenuitvoer-buffer (256 na LOMEM). De derde byte bevat dus de offset van (BASIC-)regel tot regel.

Als u wilt weten, op welke geheugenplaatsen uw BASIC-programma zich bevindt, probeert u dan eens het volgende programma:

```
0 ADR136.BEC
1 REM **********************************
2 REM *                                *
3 REM * STARTADRES/BASICREGELNUMMER    *
4 REM *                                *
5 REM **********************************
10 A=10
20 B=30
30 C=7.5
40 F=A*B
50 G=B/C
60 H=C-A
70 ? F,G,H
100 T=PEEK(136)+PEEK(137)*256
110 ZN=PEEK(T)+PEEK(T+1)*256
120 IF ZN>32767 THEN END
130 ? "BASIC-REGEL # ";ZN;" HEEFT STARTADRES   ";T
140 T=T+PEEK(T+2)
150 GOTO 110
```

programma ADR136.BEC

10 t/m 70: bevatten het surrogaat-programma.

100: berekent de numerieke waarde van de wijzer STMTAB

110: Hier wordt uit byte 0 en 1 van de eerste in tokens omgezette BASIC-regel het regelnummer RN berekend.

130: Het regelnummer RN en het begin-adres T worden op het beeldscherm getoond. U kunt hier ook LPRINT neerzetten, als u het overzicht zwart op wit wilt hebben.

140: Nu wordt de derde byte (T+2) gebruikt om het begin-adres van de volgende BASIC-regel te vinden.

150: En de volgende regel kan verwerkt worden.

Wanneer we dit eenmaal hebben doorzien, is het ook geen al te groot probleem meer om een routine te schrijven, waarmee de regelnummers veranderd kunnen worden. Daarvoor hoeven we slechts de nulde en eerste byte van iedere statement-regel op te zoeken en daar het gewenste regelnummer neer te zetten:

```
0 REM ADR137.BEC
1 REM ********************************
2 REM *                              *
3 REM *BASIC-REGELNUMMERS VERANDEREN*
4 REM *                              *
5 REM ********************************
10 A=25
20 B=7.5
30 C=17
40 F=A/B
50 G=B*C
60 H=C-A
70 ? F,G,H
1000 RN=500:SG=25:W=PEEK(136)+PEEK(137)*256
30000 HI=INT(RN/256):POKE W,RN-HI*256:POKE W+1,HI:RN=RN+SG:W=W+PEEK(W+2):GOTO 30
000
```

programma ADR137.BEC

10 t/m 70: bevatten weer het proef-programma.

1000: In RN slaan we het eerste regelnummer op en in SG de stapgrootte van de nummering. W berekent de wijzer naar de eerste BASIC-regel.

30000: Hier vindt de eigenlijke omzetting plaats. HI berekent uit RN het gedeelte van de HI-byte. Dan volgt op adres W de LO-byte rest (RN-HI*256) gePOKEd en op adres W+1 de HI-byte waarde. Vervolgens wordt het regelnummer RN met de stapgrootte SG verhoogd en we hebben het volgende regelnummer. Nu wordt de wijzer W met de offset (PEEK(Y+2)) verzet. Tenslotte volgt de sprong naar 30000, zodat de volgende BASIC-regel verwerkt kan worden.

Het programma eindigt met een foutmelding. Als namelijk het regelnummer van regel 3000 als laatste wordt veranderd, dan vindt GOTO 30000 geen regel meer. Als u nu een LIST vraagt, ziet u dat de laatste regel nu het nummer 850 heeft.

Als u de waarden voor RN en SG bepaalt, moet u erop letten, dat het hoogste toelaatbare regelnummer (32767) niet wordt overschreden. Als u een hernummer-functieprogramma schrijft, dan moet het programma testen of aan deze voorwaarde wordt voldaan, omdat niet altijd te schatten is tot welk regelnummer het veranderde programma zal gaan.

Ook door het veranderen van de regelnummers kunt u uw programma onLISTbaar maken. Nadat het BASIC-programma geschreven is en in tokens in het geheugen zit, krijgen alle regels hetzelfde nummer. Dat gebeurt volgens hetzelfde principe, waarmee ook de regelnummers worden veranderd.

Is het programma eenmaal geschreven en staan de BASIC-regels in de juiste volgorde in het geheugen, dan zijn voor de computer de regelnummers niet meer van belang. Zelfs wanneer alle BASIC-regels hetzelfde regelnummer hebben, werkt het programma nog steeds normaal. Het kan ook gewoon geSAVEd en geLOAD worden. Eenmaal in het geheugen kan het ook op het beeldscherm geLIST worden. Alleen als met LIST op een rand-

apparaat opgeslagen en dan weer geladen wordt, blijft van het totale programma slechts de laatste regel over.

En zo worden de regelnummers gelijk gemaakt:

```
0 REM ADR137X.BEC
1 REM ********************************
2 REM *                              *
3 REM *   REGELNUMMERS VERSTOREN     *
4 REM *                              *
5 REM ********************************
10 A=25
20 B=7.5
30 C=17
40 F=A/B
50 G=B*C
60 H=C-A
70 ? F,G,H
1000 Z=PEEK(136)+PEEK(137)*256
30000 POKE Z,244:POKE Z+1,1:Z=Z+PEEK(Z+2):GOTO 30000
```

programma ADR137X.BEC

10 t/m 70: bevatten weer het dummy-programma.

1000: berekent de wijzer STMTAB

30000: verandert alle regelnummers in 500 (244 + 1*256).
Natuurlijk kunt u hier ieder toelaatbaar regelnummer invullen, bijvoorbeeld ook 0. Wordt een waarde voor de regelnummers ingevoerd, die het toelaatbare bereik overschrijdt, bijvoorbeeld LO=255 en HI=255, dan wordt de omzetting zonder ERRORs uitgevoerd en daarmee verdwijnt het gehele programma. Een bijzonder exclusieve manier, om een programma te wissen.

138,139 $8A,$8B STMCUR

Wijzer naar het BASIC-statement dat wordt uitgevoerd. Wordt gebruikt als toegang tot de tokens van een BASIC-regel, die op een bepaald moment wordt verwerkt. Als BASIC op een invoer wacht, wijst deze wijzer naar de direkte invoer-regel (32768) en dat opent de mogelijkheid voor een bijzondere programma-bescherming:

```
0 REM ADR138.BEC
1 REM ********************************
2 REM *                              *
3 REM *  LIST-BESCHERMING MET RUN    *
4 REM *                              *
5 REM ********************************
32766 POKE PEEK(138)+PEEK(139)*256+2,0:SAVE "D:filenaam.ext":NEW
32767 REM starten met GOTO 32766
```

32766: Zet deze instructie als laatste regel in uw programma en sla het programma op met GOTO 32766. Voordat het programma wordt geSAVEd, wordt met behulp van STMCUR een sprong uit de direkte invoer-regel onmogelijk gemaakt. Dan wordt het programma in token-formaat opgeslagen, dat wil zeggen alle wijzers, ook STMCUR, blijven met hun aktuele waarden bewaard. Daarna wordt het programma met NEW gewist.

Deze bescherming werkt ook met de cassette-recorder. Geef gewoon SAVE 'C:'. Maar wees voorzichtig. Voordat u volgens deze methode een beschermde versie van uw programma maakt, kunt u beter eerst een onbeschermde versie maken, want het bronprogramma, dat zich in de computer bevindt, wordt gewist.

Omdat het programma met SAVE is opgeslagen, kan het alleen met LOAD worden geladen. Na de direkte invoer van LOAD'...' is echter geen verdere direkte invoer meer mogelijk, dus geen LIST, maar ook geen RUN. GeSAVEde programma's kunnen echter met RUN'D:filenaam,ext' of met RUN'C:' geladen en onmiddellijk gestart worden. Het programma kan dus gewoon werken, de gebruiker heeft alleen geen gelegenheid meer om een LIST op te vragen, omdat het programma meteen loopt.

De voorwaarde voor een werkzame bescherming is natuurlijk, dat de gebruiker nooit de gelegenheid krijgt, een BASIC-commando in te voeren, of het nu LIST is of iets anders. Daarom moet het programma absoluut waterdicht worden gemaakt. Het mag geen fout bevatten, die leidt tot een programma-onderbreking met ERROR-, omdat het daarna weer mogelijk is om commando's te geven. Hoe u een programma absoluut onfeilbaar kunt afsluiten voor onderbreking door ERROR-, zal ik u laten zien bij STOPLIN (186,187 = $BA,$BB).

Bovendien moeten RESET en BREAK worden vergrendeld, zodat de gebruiker de loop van het programma niet kan onderbreken. Hoe dat te bereiken is, werd al eerder verklaard. Als u met de hier voorgestelde methode uw programma beschermt, zijn verdere maatregelen eigenlijk overbodig. Uit voorzorg kunt u echter nog de een of andere wijzer verzetten en bijvoorbeeld alle variabelen vervangen door RETURN.

Daarmee heeft u echt een degelijke programma-bescherming voor het gebruik thuis. Copieëren kan men een programma echter altijd en een prof zal het ook geen al te grote hoofdbrekens kosten om deze aardige beschermings-maatregelen buiten werking te stellen. Werkelijke software-bescherming wordt op heel andere slagvelden uitgevochten.

Maar omdat wij ons nu toch met eenvoudige zaken bezig houden, volgt hier nog een methode om geheime boodschappen te versturen, of 'BASIC-programma's als brievenbus'.

U hebt daarvoor een onschuldig programma nodig, waarin u ergens een REM-regel zet (of eventueel meerdere) met uw geheime boodschap.

```
0 REM ADR138S.BEC
1 REM ********************************
2 REM *                              *
3 REM * GEHEIME BOODSCHAP/CODEWOORD *
4 REM *                              *
5 REM ********************************
100 REM PROGRAMMA
200 REM PROGRAMMA
300 REM PROGRAMMA
400 REM Ik houd van je
500 REM PROGRAMMA
600 REM PROGRAMMA
700 REM PROGRAMMA
800 ? "Druk oP <START>"
810 ? :? "om een nieuw sPel te be9innen.":?
820 IF PEEK(53279)=6 THEN RUN
830 IF PEEK(53279)=1 THEN GRAPHICS 0:GOTO 1000
840 GOTO 820
1000 ? CHR$(125):? :? "Na invoer van het codewoord"
1010 ? :? "wordt een onbeschermde versie van het"
1020 ? :? "Pro9ramma oP de diskette 9ezet."
1030 ? :? :? "Het Pro9ramma laadt u met":?
1040 ? "ENTER";CHR$(34);"D:CIA.KGB";CHR$(34)
1050 DIM CW$(5)
1060 TRAP 1060
1070 INPUT CW$
1080 IF CW$="QRYXT" THEN LIST "D:CIA.KGB",0,1999
1090 GOTO 1060
20000 POKE PEEK(138)+PEEK(139)*256+2,0:SAVE "D:BOODSCH.GEK":NEW
```

programma ADR138S.BEC

100 t/m 700: stellen het onschuldige programma voor, bijvoorbeeld een spel, met de geheime boodschap in regel 400.

800 en 810: Het programma eindigt met de optie, een nieuw spel te beginnen door op START te drukken.

820: Wordt op de START-toets gedrukt, dan wordt met RUN opnieuw gestart.

830: Alleen de bevoegde gebruiker weet, dat hij op deze plaats niet op START moet drukken of de computer uit moet schakelen. Hij drukt tegelijk de toetsen OPTION en SELECT in en springt daardoor naar regel 1000.

840: beëindigt de uitlees-lus.

1000 t/m 1050: Hier begint de tweede beschermings-routine, waarbij de invoer van een codewoord wordt gevraagd. Dit programma beperkt zich tot het controleren van het ingevoerde codewoord. Op deze plaats zijn natuurlijk met enige fantasie nog een hele rij verdere beschermingsmaatregelen te bedenken. Zo kan men heel rigoureus bij het verkeerde codewoord het programma wissen (NEW).

1060: De invoer moet tegen (mogelijk opzettelijk) foutieve invoer worden beveiligd.

1080: Was het codewoord correct, dan wordt het programma hier op diskette geLIST, waarbij de laatste regel met de beschermings-routine niet wordt meegeschreven. Natuurlijk werkt dat ook met een cassette.

20000: Hier staat de LIST/LOAD bescherming; u moet het programma dus met LOAD 20000 opslaan, voordat u het brengt naar degene, voor wie het programma en/of de boodschap bestemd is.

140,141	$8C,$8D	STARP

Wijzer naar de eerste byte van de veldvariabelen-tabel (strings en arrays). Arrays worden in 6 byte BCD-formaat opgeslagen. Als u dus een variabele (op 9,9) DIMensioneert, dan worden daarvoor 10 maal 10 maal 6, dus 600 bytes gereserveerd. Strings gebruiken daarentegen slechts een byte voor ieder element, dus DIM G$(20) reserveert twintig bytes.

Vaak kan veel geheugenruimte worden bespaard door getallen, die alleen afgedrukt moeten worden, niet als numerieke waarde, maar als string op te slaan. Het volgende programma geeft een voorbeeld:

```
0 REM ADR140.BEC
1 REM ********************************
2 REM *                              *
3 REM *STRINGS IN PLAATS VAN ARRAYS *
4 REM *                              *
5 REM ********************************
10 DIM A(10),Z$(20)
20 A(0)=28:A(1)=6:A(2)=1961:A(3)=140
30 Z$="28061961ADR140"
40 ? A(0);".";A(1);".";A(2);":";A(3)
50 ? :? Z$(1,2);".";Z$(3,4);".";Z$(5,8);":";Z$(9,14)
```

programma ADR140.BEC

Let op! De listing van regel 30 bevat pseudo-grafische tekens!

10: Ten overvloede (van de tabel) worden arrays en strings door onervaren programmeurs vaak ook nog op voorraad geDIMensioneerd.

20: Hier worden in A(0) t/m A(3) de getallen opgeslagen voor het aangegeven van een datum en een herkenningsgetal. Effectief geheugengebruik: 4 maal 6 = 24 bytes.

30: Hier worden alle benodigde tekens in een string ondergebracht. Geheugengebruik veertien bytes, hoewel nog aanvullende informatie mee opgeslagen is.

40: De datum en het herkenningsgetal worden afgedrukt.

50: Hier wordt hetzelfde bereikt met string-gedeelten. In een numerieke waarde kan geen 0 voorop staan. Als string kan de dag worden aangegeven met 09 (in plaats van alleen 9) en, zoals uit het herkenningsgetal blijkt, kunnen cijfers en letters gemakkelijk door elkaar worden gebruikt. Omdat strings met het BASIC-statement VAL in beperkte mate ook in numerieke waarden omgezet kunnen worden, kunnen met de zo opgeslagen getallen ook berekeningen worden uitgevoerd.

Laten we vervolgens eens een blik werpen op de tabel van de veldvariabelen:

```
0 REM ADR140A.BEC
1 REM ********************************
2 REM *                              *
3 REM *   STRING- EN ARRAY-TABEL     *
4 REM *                              *
5 REM ********************************
10 DIM A(9),Z$(20)
20 REM FOR J=0 TO 9:A(J)=0:NEXT J
30 REM Z$="      "
40 A(0)=28:A(1)=6:A(2)=1961:A(3)=140
50 Z$="28061961ADR140"
60 ? A(0);".";A(1);".";A(2);":";A(3)
70 ? :? Z$(1,2);".";Z$(3,4);".";Z$(5,8);":";Z$(9,14)
80 FOR J=1 TO 1000:NEXT J
100 ? CHR$(125)
110 SP=PEEK(140)+PEEK(141)*256
120 FOR J=0 TO 9
130 FOR I=0 TO 5
140 ? PEEK(SP+J*6+I);" ";
150 NEXT I
160 ?
170 NEXT J
200 FOR J=60 TO 79
210 ? PEEK(SP+J);" ";
220 NEXT J
230 ?
240 FOR J=60 TO 73
250 ? CHR$(PEEK(SP+J));
260 NEXT J
```

programma ADR140A.BEC

10 t/m 70: bevatten het hiervoor reeds verklaarde programma.

100: wist het beeldscherm.

110: berekent met de wijzer STARP het start-adres van de string- en array-tabel.

120: leest de bytes van de tien A-variabelen.

130: leest de zes bytes van iedere A-variabele.

140: PRINT de bytes van een A-variabele met tussenruimtes achter elkaar.

160: begint met de volgende A-variabele op een nieuwe regel, zodat de zes bytes steeds netjes onder elkaar staan.

200: Omdat het array A 60 bytes in beslag neemt, liggen de (ATASCII-) waarden van de Z-string van 60 t/m 73.

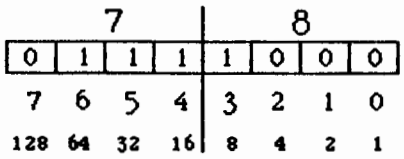
210: Hier worden ze gePEEKt en gePRINT.
Als u het programma start met RUN, zult u zien, dat in de tabel de hel is losgebroken bij variabelen, die wel geDIMensioneerd worden, maar waaraan nog geen waarde is toegekend. Bij arrays kan dat tot noodlottige fouten leiden.
Verwijder nu in de regels 20 en 30 het REM-statement. Vervang in regel 240 het getal 73 door 79 en laat het programma nog een keer lopen. Nu ziet de tabel er veel beter uit. De array-elementen, die nog geen waarde hebben gekregen, zijn met 0-en gevuld. U ziet hier, dat string-elementen in ATASCII-formaat worden opgeslagen, namelijk spaties als 32 en hartjes als 0.

Natuurlijk kunnen waarden van arrays en strings direkt in de tabel worden gePOKEd, als men weet op welke plaats de gezochte variabele staat. De string-elementen worden zoals gezegd in ATASCII-formaat opgeslagen. De numerieke waarden in 6-byte BCD-formaat. BCD (binary coded decimal) codeert ieder afzonderlijk cijfer van een getal in binaire vorm. Dat kost veel geheugenruimte en is voor de rekenkundige bewerkingen veel ingewikkelder, gebruikt dus meer tijd, maar is daardoor wel exacter dan de integer-methode.

Het nulde byte van een BCD-constante bevat in bit 7 het teken (aan -, uit +) en in de bits 6 t/m 0 de exponent met grondtal 100. De exponent is positief, als de bits 6 t/m 0 samen een decimale waarde hebben groter dan of gelijk aan 64. Dan is de grootte van de exponent gelijk aan de waarde van de bits 6 t/m 0 min 64. Dus 64 betekent, dat de exponent 0 is (100 tot de macht 0 = 1, het getal ligt tussen 0 en 99), 65 betekent E = 1 (getal tussen 0 en 9999), enzovoorts. Waarden kleiner dan 64 geven op dezelfde manier negatieve exponenten aan, dus het aantal plaatsen achter de komma.

In de overige vijf bytes zijn de cijfers opgeslagen, waaruit het getal is samengesteld. Daarbij wordt ieder cijfer binair gecodeerd in een nibble opgeslagen. Heeft de variabele bijvoorbeeld de waarde -12, dan heeft de nulde byte de waarde 192 (128+64) en de eerste byte bevat de 12. De hoge nibble bevat de 1 in zijn laagste bit (decimaal 1 = binair 0001), de lage nibble bevat de 2 (decimaal 2 = binair 0010).

Moet het getal 78 worden opgeslagen, dan is de nulde byte 64. De 7 wordt in de hoge nibble als 0111 (= binair 7) gezet, de 8 in de lage nibble als 1000:



In het volgende voorbeeld-programma wordt de waarde van een array-element direkt in de tabel bepaald. Het getal krijgt de cijfervolgorde 1234567890 en de decimale punt wordt door de exponent-byte tussen de zesde en zevende plaats gezet. De 0 op de vierde plaats achter de komma verschijnt niet op het beeldscherm:

```
0 REM ADR141.BEC
1 REM ********************************
2 REM *                              *
3 REM *GEDIMDE VARIABELEN VERANDEREN*
4 REM *                              *
5 REM ********************************
10 DIM Z(9)
20 FOR J=0 TO 9:Z(J)=0:NEXT J
30 SP=PEEK(140)+PEEK(141)*256
40 POKE SP,66:REM = 100^2
50 POKE SP+1,18:REM = 1 / 2
60 POKE SP+2,52:REM = 3 / 4
70 POKE SP+3,86:REM = 5 / 6
80 POKE SP+4,120:REM = 7 / 8
90 POKE SP+5,144:REM = 9 / 0
100 ? Z(0):? :?
110 FOR J=0 TO 5
120 ? PEEK(SP+J);"   ";
130 NEXT J
```

programma ADR141.BEC

10: De grootte van de DIMensionering is willekeurig.

20: Alle elementen van het array worden op 0 gezet. Een maatregel, die na iedere DIMensionering vanzelfsprekend zou moeten zijn.

30: Hier wordt het adres berekend, waarnaar STARP wijst.

40: In de eerste byte (byte 0) worden de exponent en het teken opgeslagen.

50: In byte 1 slaan we de cijfers 1 en 2 binair gecodeerd (0001/0010) op. De byte krijgt daardoor de decimale waarde 18.

60: In byte 2 zetten we de cijfers 3 en 4 (0011/0100 = 52).

70: In byte 3 komen de 5 en de 6 terecht (0101/0110 = 86).

80: In byte 4 de cijfers 7 en 8 (0111/1000 = 120).

90: In byte 5 de cijfers 9 en 0 (1001/0000 = 144).

100: geeft het decimale getal Z en

110 t/m 130: geven de zes data-bytes van de desbetreffende BCD-constante.

142,143 $8E,$8F RUNSTK

Adressen van de runtime-stack (verwerkings-stapelgeheugen), die de gegevens bevat voor alle in werking zijnde GOSUB-routines (ieder vier bytes) en FOR-NEXT lussen (ieder zestien bytes).

De GOSUB-gegevens bevatten in byte 0 een 0 als indicator voor 'GOSUB', in byte 1 en 2 het regelnummer, waarin de GOSUB werd opgeroepen, als integere waarde verdeeld in L0 en HI en in de laatste byte de offset binnen de regel, zodat de computer na RETURN het volgende statement kan terugvinden en uitvoeren.

De FOR-NEXT gegevens bevatten in de bytes 0 t/m 5 de eindwaarde voor de telvariabele (dus de waarde die achter T0 wordt aangegeven) als BCD-constante. In de bytes 6 t/m 11 staat de STEP-grootte, eveneens in BCD-vorm. In byte 12 staat het nummer van de teller-variabele. In de bytes 13 en 14 staat het nummer van de regel, waarin het FOR-statement staat. In de laatste byte, nummer 15, staat de regel-offset van het FOR-statement.

Door RUNSTK wordt tegelijkertijd het eind van de veldvariabelen-tabel gemarkeerd.

144,145 $90,$91 MEMTOP

Wijst naar het einde van de geheugenruimte, die door een BASIC-programma wordt gebruikt. De BASIC-functie FRE(0) berekent hoeveel geheugenplaatsen er nog over zijn tussen MEMTOP en het begin van de display-list en het beeldscherm-geheugen door het verschil te nemen tussen de wijzers SDLSTL en MEMTOP.

Moet er geheugenruimte worden gereserveerd, dan moet het hier en in RAMTOP (106 = $6A) worden ingevoerd.

De geheugenplaatsen 146 t/m 202 ($92 t/m $CA) zijn gereserveerd voor het gebruik door de 8k BASIC ROM.

186,187 $BA,$BB STOPLN

Slaat het regelnummer op, waarbij het programma wordt onderbroken door ERROR, STOP, TRAP of BREAK.

Met dit register kan iedere onderbreking door een van de bovengenoemde oorzaken worden voorkomen, bijvoorbeeld voor de software-bescherming of omdat het de gemakkelijkste manier is, om te voorkomen dat een programma door verkeerde invoer van gegevens onderbroken wordt.

In het volgende programma wordt een dubbel geïndiceerde variabele geDIMensioneerd, waarvan de elementen steeds door willekeurige waarden worden verhoogd. Omdat deze willekeurige waarden groter kunnen uitvallen, dan het array geDIMensioneerd is, wordt de TRAP-truc gebruikt:

```
0 REM ADR186.BEC
1 REM ********************************
2 REM *                              *
3 REM *     ERROR-BEVEILIGING        *
4 REM *                              *
5 REM ********************************
9 TRAP 9:F=F+1:GOTO PEEK(186)+PEEK(187)*256+10
10 REM Regel waarnaar TRAP de eerste keer springt
100 DIM M(6,6):FOR I=0 TO 6:FOR J=0 TO 6:M(J,I)=0:NEXT J:NEXT I
110 X=INT(RND(0)*6)
120 Y=INT(RND(0)*6)
130 AL=4-X:AR=4+X
140 BL=4-Y:BR=4+Y
150 M(AL,BL)=M(AL,BL)+1
160 M(AR,BL)=M(AR,BL)+1
170 M(AR,BR)=M(AR,BR)+1
180 M(AL,BR)=M(AL,BR)+1
190 Z=Z+1:IF Z<50 THEN 110
200 FOR J=0 TO 6
210 FOR I=0 TO 6:? M(J,I);" ";:NEXT I:?
220 NEXT J
230 ? :? :? F
```

programma ADR186.BEC

9: Hier wordt de TRAP neergezet. Hij wijst naar zichzelf, dus steeds wanneer de TRAP in werking treedt, belandt het programma in deze regel, waar de TRAP wordt vernieuwd. De daarop volgende GOTO-verwijzing leest uit de wijzer STOPLN in welke regel de TRAP is opgetreden en zorgt ervoor dat het programma verder gaat met de regel, waarvan het nummer 10 hoger ligt.

Voorwaarde is natuurlijk, dat in het hele programma de regelnummers elkaar consequent in stappen van tien opvolgen en dat in de laatste regel geen fout kan optreden, omdat de GOTO-verwijzing dan zelf tot een foutmelding leidt en daardoor zou het programma in een eindeloze lus terecht komen doordat het steeds weer naar deze regel terugspringt.

Met het oog op software-bescherming geeft zo'n regel de absolute zekerheid, dat de gebruiker het programma niet door de een of andere verkeerde invoer kan onderbreken. Voor dit doel is het dan ook niet van belang, dat het programma na de TRAP ergens verder kan gaan, want dat zou dan ook weer door TRAP worden opgevangen.

Hier is ook een vorm denkbaar als:

```
100 TRAP 20000
20000 NEW:LOAD'D:filenaam. ext'
```

In dit voorbeeld-programma telt de variabele F voor de grap, hoeveel foutmeldingen door de TRAP-truc worden overbrugd.

10: Wanneer het programma voor de eerste maal wordt doorlopen, staat de wijzer STOPLN op 0. In regel 9 volgt dus een GOTO naar 10. Daarom moet het programma beslist een regel 10 hebben. Daar kan dan eventueel ook een REM-statement staan.

100: De array-variabele M wordt op zeven maal zeven elementen geDIMensioneerd. Alle elementen worden gevuld met 0.

110 en 120: X en Y bevatten willekeurige waarden van 0 t/m 5.

130 en 140: Door X en Y krijgen de variabelen AL, AR, BL en BR waarden, die voor een deel niet als rangnummers voor het geDIMensioneerde array gebruikt kunnen worden. Als dat in de regels

150 t/m 180: toch gebeurt, dan leidt dat regelmatig tot de foutmelding 'getallenbereik verkeerd'. Ontoelaatbare waarden van X en Y met IF-voorwaarden testen zou te omslachtig zijn. Dat geldt m.n. voor de gevallen, waarin slechts een van de beide variabelen de waardegrens overschrijdt en de array-elementen, die door de toelaatbare waarde bereikt kunnen worden, toch verhoogd moeten worden.

De TRAP-constructie in regel 9 biedt hier een eenvoudige oplossing. Wordt een poging ondernomen om een niet aanspreekbaar array-element op te roepen, dan leidt de foutmelding tot het vervolgen van het programma op de volgende regel. Het is dus onafhankelijk van de toevalsgetallen. Als we een element van de dubbel geïndiceerde variabele M kunnen bereiken, dan telt M deze gebeurtenis. Als het toevallig niet lukt, dan loopt het programma ongehinderd verder.

190: Het test-programma wordt vijftig keer doorlopen.

200 t/m 220: Dan wordt de veldvariabele gePRINT en in regel

230: laat F zien hoeveel maal een foutmelding is opgetreden.

195 $C3 ERRSAVE

bevat het nummer van de fout, die tot een STOP of TRAP heeft geleid.

201 $C9 PTABW

Bepaalt het aantal kolommen tussen de tabulator-stops, die in een PRINT-statement door het plaatsen van een komma worden veroorzaakt. Dit is dus de afstand tussen het laatste teken van de voorgaande PRINT en het eerste teken van de volgende. Deze waarde is onafhankelijk van de TAB-functie.

De standaardwaarde is 10. De kleinst mogelijke waarde is 3. Bij de in PTABW gePOKEte waarde wordt nog eens 2 opgeteld. Dus POKE 201,1 veroorzaakt een tabulator-sprong-grootte van drie kolommen. De grootst mogelijke waarde is 255. Dit geeft dus een afstand van 257 kolommen.

Wordt in PTABW een 0 gezet, dan blijft het systeem hangen bij de eerstvolgende komma in een print-statement en kan dan alleen nog met RESET teruggehaald worden.

203-209 $CB-$D1 ...

Worden niet gebruikt door BASIC.

210,211 $D2,$D3 ...

Gereserveerd voor BASIC of cartridge.

212-241 $D4-$F1 diversen

Registers voor berekeningen met floating point (verschuivende komma).

242 $F2 CIX

Karakter-index. Offset voor de tekst-invoer-buffer waar INBUFF naar wijst.

243,244 $F3,$F4 INBUFF

Wijst naar de tekst-invoer-buffer, dus de invoer-buffer voor de programma-regels van de gebruiker.

245-250 $F5-$F4 ZTEMP1-3

Tijdelijke registers.

251 $FB RADFLG

Vlag (aanduiding) voor RADians (radialen) of DEGrees (graden). De standaard-waarde 0 voert alle trigoniometrische berekeningen uit in radialen (basis 100). Als hier een zes wordt gePOKEd of wanneer het BASIC-commando DEG wordt gegeven, dan schakelt de computer over op graden (basis 360).

252,253 $FC,$FD FLPTR

Wijzer naar het eerste floating point getal van de gebruiker.

254,255 $FE,$FF FPTR2

Wijzer naar het tweede floating point getal dat in een berekening wordt gebruikt.

256-511 $100-$1FF pagina 1

Dit is het gebied voor de stack (stapel) van het OS, het DOS en BASIC. De machinetaal-instructies JSR, PHA en interrupts zorgen ervoor, dat data op pagina 1 wordt geschreven. RTS, PLA en RTI lezen data van pagina 1.

Na Powerup of Reset wijst de stapelwijzer naar adres 511. De stapel wordt dan naar onderen t/m 256 volgeschreven. In het geval dat de stapel te groot wordt (overflow), springt de stapelwijzer terug naar 511.

512,513 $200,$201 VDSLST

Wijzer voor de NMI (non-maskable interrupt) display-list-interrupt (DLI). Hier wordt het adres opgeslagen, waarnaar tijdens een DLI gesprongen moet worden.

DLIs worden gebruikt, om tijdens de microseconden van een horizontale blanking verdere programma-instructies uit te voeren, bijvoorbeeld tonen van een melodie te produceren. Daardoor kan de indruk worden gewekt, dat een aantal dingen gelijktijdig gebeuren.

Tijdens DLI kunnen ook de grafische modus, de waarden van de kleurenregisters, enzovoorts worden veranderd. Daardoor is het mogelijk, in iedere modusregel een andere kleurwaarde te gebruiken, zodat alle 256 oproepbare kleuren op een beeldscherm gelijktijdig kunnen verschijnen.

Het OS gebruikt geen DLIs. Ze moeten door de gebruiker mogelijk gemaakt, geschreven en opgeroepen worden. VDSLST wordt zo geïnitieerd, dat hij naar adres 59315 ($E7B3) wijst. Om een DLI mogelijk te maken, moet eerst op adres ($D40E) de waarde 192 worden gezet, zodat ANTIC de request (het verzoek om een interrupt) herkent. Dan moet op 512 (L0) en 513 (H1) het adres worden gePOKEd, waar de machinetaal-routine begint, die tijdens de DLI uitgevoerd moet worden. In de display-list moet bij de instructie voor de modus-regel, waama een DLI moet volgen, bit 7 (+128) worden aan gezet. Afhankelijk van de grafische modus staan voor de DLI 14 tot 61 machine-cycli ter beschikking. Als eerste moeten alle 6502 registers op de stapel worden gezet (push to stack) en de DLI moet eindigen met een RTI (return from interrupt). Omdat de DLI in machinetaal moet lopen, kunnen de gewenste veranderingen direkt in de hardware-registers worden gePOKEd. Veranderingen in de parallel-registers, die door BASIC kunnen worden bereikt, wor-

den pas na de VBLANK (verticale blanking) gezien. Daarom zijn bijvoorbeeld veranderingen van de kleurwaarden in de kleurenregisters optisch wel snel te realiseren, maar ze beïnvloeden altijd het hele beeldscherm. Met een DLI is het echter mogelijk om bijvoorbeeld kleurenregister 710 (achtergrond bij GRAPHICS 0) op 140 (helblauw = hemel) te zetten en na enige modus-regels voor de rest van het grafische raam op 182 (meigroen = weide) te zetten.

```
0 REM ADR512.BEC
1 REM ********************************
2 REM *                              *
3 REM * DISPLAY LIST INTERRUPT DLI   *
4 REM *                              *
5 REM ********************************
10 POKE 710,140:POKE 712,0:POKE 709,2
20 X=PEEK(560)+PEEK(561)*256
30 P=1536:FOR DLI=P TO P+10:READ B:POKE DLI,B:NEXT DLI
40 DATA 72,169,182,141,10,212,141,24,208,104,64
50 POKE 512,0:POKE 513,6
60 POKE 54286,192
70 POKE X+6+J,130
80 FOR I=0 TO 300:NEXT I
90 POKE X+6+J,2:J=J+1:IF J<21 THEN 70
100 GOTO 100
```

programma ADR512.BEC

10: Hier worden de kleurwaarden gePOKEd. Register 712 bepaalt bij GR.0 de rand (0 = zwart), 709 de helderheid van de tekst.

20: berekent de wijzer SDLSTL naar de display list.

30: P is het startadres voor de machinetaal-routine. P is het begin van pagina 6 (6*256 = 1536). Hier ligt een klein, beschermd RAM-gebied voor de gebruiker onder LOMEM. De FOR-NEXT lus leest de data-bytes op de voorbij P liggende geheugenplaatsen, die het machinetaalprogramma bevatten.
40: De machinetaal-routine. Naast deze methode om een machinetaal-programma als data-bytes in het geheugen te zetten, is er nog de mogelijkheid, het machinetaal-programma in een string op te slaan. De data-bytes in deze regels kunnen met:

```
FOR J=0 TO 10:READ A:B$(J,J)=CHR$(A):NEXT J
```

in een string worden omgezet. De machinetaal-routine wordt dan in de tabel met veldvariabelen opgeborgen en zal bij interne geheugenverschuivingen door het OS zeker mee verschuiven. Dat ontneemt de gebruiker de zorg te voorkomen dat er over de routine heen wordt geschreven. Het begin-adres van een string in de string- en array-tabel kan met de BASIC-instructie ADR(B$) worden vastgesteld en BASICs USR zet de computer aan het werk: USR(ADR(B$).

50: De DLI-wijzer wordt naar pagina 6 (LO=0,HI=6) gezet.

60: In ANTICs NMIEN (54286 = $D40E) moeten bit 7 en 6 aan zijn (decimaal 192) om de DLI mogelijk te maken.

70: Hier wordt de ANTIC commando-byte in de display-list van 2 (voor een GR.0-regel) veranderd in 130 (2+128 voor een GR.0-regel met DLI).

80: Een kleine pauze voor de ogen van de toeschouwer.

90: Hier wordt het ANTIC commando-byte weer in een 2 veranderd. Dan wordt J met 1 verhoogd zodat, wanneer het programma een volgende keer wordt doorlopen, de DLI in de daarop volgende regel wordt toegevoegd. Hierdoor ontstaat het effect, dat het bovenste deel van het grafische raam (hemel) zich langzaam, modusregel na modusregel, naar onderen uitbreidt.

Als de kleuren u niet bevallen (over smaak valt niet te twisten), kunt u de kleur van de hemel instellen met POKE 710,n; de kleur van de rand met POKE 712,n en de helderheid van de tekst met POKE 709,m. Voor n en m kunt u willekeurige gehele getallen kiezen van 0 t/m 254, voor m zijn echter alleen gehele getallen van 0 t/m 14 interessant, omdat alleen de helderheid beïnvloed kan worden.

Wilt u de kleur na de DLI (dus in het onderste deel van het grafische raam) veranderen, dan moet de desbetreffende data-byte in de machinetaal-routine worden veranderd. Dat is in regel 40 de derde getalwaarde. Ook hier zijn alleen gehele waarden toegestaan.

Omdat er slechts een DLI-wijzer is, moet de wijzer door de voorafgaande DLI-routine zelf worden veranderd, als meerdere DLIs uitgevoerd moeten worden. Bovendien moet het klikken van het toetsenbord worden onderdrukt (register 731 = $2DB) of toetsenbord-invoer onmogelijk worden gemaakt, omdat het de DLI kan storen.

514,515 $202,$203 VPRCED

Wijzer naar de PLA,RTI-routine voor de seriële interface (serial proceed line vector).

516,517 $204,$205 VINTER

Wijzer naar de IRQ-routine voor de seriële interface (serial interrupt vector).

518,519 $204,$205 VBREAK

Wijzer voor de 6502 BRK-instructie ($00). Dit heeft niets met de BREAK-toets te maken.

520,521 $208,$209 VKEYBD

POKEYs toetsenbord interrupt-wijzer. Wordt gebruikt om een interrupt op te wekken door het indrukken van een toets (behalve BREAK). Geïnitieerd op 65470 (keyboard IRQ-routine van het OS).

522,523 $20A,$20B VSERIN

POKEYs wijzer naar de routine voor het ontvangen van seriële data.

524,525 $20C,$20D VSEROR

POKEYs wijzer naar de routine voor het zenden van seriële data.

526,527 $20E,$20F VSEROC

POKEYs wijzer voor het beëindigen van seriële data-overdracht.

528-533 $210-$215 VTIMR1,2,4

Interrupt-wijzers voor de POKEY-timers 1, 2 en 4 (AUDF1, 2, 4). Heeft de desbetreffende POKEY-timer tot 0 afgeteld, dan wordt naar het adres gesprongen, waarnaar de bijbehorende wijzer wijst.

534,535 $216,$217 VIMIRQ

IRQ interrupt-hoofdvector.

536,537 $218,$219 CDTMV1

System-timer 1. Telt iedere 1/50 seconde terug naar 0. Als 0 bereikt is, wordt een sprong uitgevoerd naar het adres, dat in de bijbehorende wijzer CDTMA1 (550,511 = $226,$227) is opgeslagen.

Timer 1 kan beter niet worden veranderd, omdat hij door het OS wordt gebruikt voor I/O-routines.

538-545 $21A-$221 CDTMV2-5

Als CDTMV1.

546,547 $222,$223 VVBLKI

Wijzer voor een onmiddellijke VBLANK-interrupt. Deze wijzer kan worden veranderd, om tijdens de verticale blanking machinetaal-routines te laten uitvoeren, die een lengte van ongeveer 3500 machine-cycli kunnen hebben.

548,549 $224,$225 VVBLKD

Wijzer voor een verlengde VBLANK-interrupt, die tot 20000 machine-cycli lang kan zijn.

550,551 $226,$227 CDTMA1

Sprongadres voor timer 1 (CDTMV1).

552,553 $228,$229 CDTMA2

Sprongadres voor timer 2 (CDTMV2).

554 $22A CDTMF3

Vlag (aanduiding) voor timer 3 (CDTMV3).

555 $22B SRTIMR

Timer, die voor de toetsenbord-besturing wordt gebruikt. Deze veroorzaakt de vertraging, voordat de herhalingsfunctie van de toetsen in werking treedt.

Iedere keer, dat een toets wordt ingedrukt, begint deze timer bij 48. Bereikt SRTIMR de waarde 0 en is de toets nog steeds ingedrukt, dan wordt de waarde van de ingedrukte toets met intervallen van 1/10 seconde naar CH (764 = $2FC) gestuurd.

556 $22C CDTMF4

Vlag voor timer 4 (CDTMV4).

557 $22D INTEMP

Tijdelijk register, dat door de SETVBL-routine wordt gebruikt.

558 $22E CDTMF5

Vlag voor timer 5 (CDTMV5).

559 $22F SDMCTL

Parallel-register van adres 54272. DMA (direct memory access) sturing voor ANTIC. (Zie ook het hoofdstuk over player missile graphics.)

Bit 5 (32) maakt het mogelijk voor ANTIC de display-list-instructies op te halen.
Bit 4 (16) resolutie van een regel van de player bij PM-graphics.
Bit 3 (8) maakt player-DMA mogelijk.
Bit 2 (4) maakt missile-DMA mogelijk.
Bit 1 (2) *): display-breedte.
Bit 0 (1) *): display-breedte.

*) Door de bits 1 en 0 wordt bepaald in welke breedte het tv-scherm door ANTIC wordt aangestuurd.

Bit 1 aan, bit 0 aan: breed display (wordt in de literatuur 'playfield', speelveld, genoemd, met betrekking tot PM-graphics). Bij een breed display passen in GRAPHICS 0 op iedere regel 48 tekens. Daardoor wordt echter alleen de weergave zelf veranderd. De editor werkt verder met 40 tekens per regel. Als u dus door het aanzetten van de beide laagste bits de grootte van het display verandert en dan bijvoorbeeld een programma laat afdrukken, verschijnen de regels in het afwijkende formaat. Dus bij een breedte van 48 tekens begint de editor op de 41e plaats met het eerste teken van de volgende regel, enzovoorts.

Bit 1 aan, bit 0 uit: geeft de gewone breedte van 40 tekens.

Bit 1 uit, bit 0 aan: smal speelveld met 32 tekens. Hier zet de editor in de eerste acht kolommen van de volgende regel de overige tekens neer van zijn 40-teken regel en begint in de 9e kolom met zijn tweede regel.

Bit 1 uit, bit 0 uit: geen display.

Het is mogelijk, de beeldscherm-opmaak door ANTIC helemaal uit te schakelen. Daartoe moeten bit 5, bit 1 en bit 0 op 0 worden gezet. Omdat hierdoor het gehele systeem wordt ontlast, lopen veel processen sneller. Vooral bij lange berekeningen kan het nuttig zijn, het beeld tijdelijk uit te schakelen. Een programma, dat hiervan gebruik maakt, moet de gebruiker echter wel van te voren informeren, dat het beeld voor een moment uitgeschakeld wordt, zodat hij of zij niet denkt, dat het televisietoestel kapot is.

Het eenvoudigste is om eerst de actuele waarde op adres 559 te PEEKen en op te slaan in een variabele. Vervolgens moet op adres 559 een 0 worden gePOKEd. Als het beeld weer aangezet moet worden, dan zet u de waarde van de variabele weer terug in SDMCTL: POKE 559, variabele. Op die manier bent u ervan verzekerd, dat na afloop de oude waarde zich weer in dit register bevindt.

Het uitschakelen van het beeldscherm kan ook nuttig zijn, wanneer in een hoge grafische modus een beeld wordt opgemaakt, hetgeen een paar minuten kan duren. Als u echter het beeld van te voren uitschakelt, het te maken beeld in het beeldscherm-geheugen opslaat en dan op adres 559 het beeld weer aanzet, dan verschijnt het gehele beeld in een keer. Een nog mooiere methode is reeds besproken bij het voorbeeld-programma PAGING.BEC (SAVMSC 88,89 = $58,$59).

560,561 $230,$231 SDLSTL

Parallel-register van 54274 en 54275. Wijzer naar de display list (DL). De DL ligt direkt voor het beeldscherm-geheugen. Het is een kort machinetaal-programma, dat ANTIC stuurt. Het bevat aanwijzingen, waar de beeld-data in het geheugen staan en hoe ze geïnterpreteerd moeten worden. Er zijn ANTIC-instructies voor iedere grafische modus, voor de DLI, voor horizontaal en verticaal scrollen en twee sprong-instructies. (Zie aanhangsel display list.)

Als u de DL wilt bekijken, probeer dan eens dit programma:

```
0 REM ADR560.BEC
1 REM ********************************
2 REM *                              *
3 REM *   DISPLAY LIST AFDRUKKEN     *
4 REM *                              *
5 REM ********************************
10 DIM D(201)
20 ? CHR$(125)
30 ? :? "    Geef een grafische modus"
40 ? :? " waarden van 0 t/m 11 (XL t/m 15)"
50 ? :? "grafische modi zonder tekstraam +16"
60 ? :? "            en <RETURN>"
70 ? :? "------------------------------------"
100 INPUT G
110 GRAPHICS G
120 DL=PEEK(560)+PEEK(561)*256
130 FOR J=0 TO 201
140 D(J)=PEEK(DL+J)              ;
150 Z=Z+1
160 IF D(J)=65 THEN POP :GOTO 180
170 NEXT J
180 J=Z:D(J)=PEEK(DL+J)
190 D(J+1)=PEEK(DL+J+1)
200 GRAPHICS 0
210 POKE 201,1:POKE 83,37
220 FOR J=0 TO Z+1
230 ? D(J),
240 NEXT J
250 GOTO 250
```

programma ADR560.BEC

10: In de veldvariabele D worden de data-bytes van de DL opgeslagen. De langste DL (GRAPHICS 8+16) is 202 bytes lang.

20 t/m 70: maken een tekst-tabel met instructies.

100: Wanneer voor G een waarde wordt ingevoerd van 0 t/m 31, dan geeft dit een zinvol resultaat. Het programma is niet beveiligd tegen verkeerde invoer.

110: De door de gebruiker gekozen grafische modus wordt ingeschakeld.

120: In DL wordt het begin-adres van de display-list opgeslagen.

130: De lusteller wordt op de maximale waarde gezet.

140: Een data-byte van de DL wordt gePEEKt en in D(J) opgeslagen.

150: Z telt het aantal gelezen bytes.

160: 65 (JVB, jump at vertical blank) is de laatste instructie van de DL, waarna alleen nog het sprong-adres (twee bytes) volgt. Als bij het lezen van de DL de waarde 65 wordt gevonden, dan springt het programma met POP uit de FOR-NEXT lus en gaat verder op regel

180: waar de voorlaatste en op regel

190: de laatste byte van de DL wordt gelezen.

200: Voor het weergeven van de DL-data wordt GR.0 ingeschakeld.

210: De print tab-grootte (PTABW) wordt op 1, dus drie kolommen, gezet en de rechter rand (RMARGN) op 37.

220 t/m 240: laten de data-bytes van de DL zien op het beeldscherm.

250: Voorkomt het einde van het programma met READY. De data van GRAPHICS 24 vullen bij het gegeven formaat het hele beeldscherm. Bij het einde van het programma met READY worden van boven twee regels uit het beeld gescrolld. U moet het programma daarom zelf stoppen met BREAK.

Wat de verschillende getalwaarden in de DL betekenen, wordt in het aanhangsel display-list uitgelegd.

562 $232 SSKCTL

Parallel-register van adres 53775. Sturing van de seriële interface.

563 $233 LCOUNT

Byte-teller voor laad-routines.

564 $234 LPENH

Parallel-register van adres 54284. Horizontale positie van de lichtpen.

565 $235 LPENV

Parallel-register van adres 54385. Verticale positie van de lichtpen. De positie-waarden voor de lichtpen komen overeen met de waarden voor PM-graphics. Ze wijken af van de normale kolom- en regel-waarden, die bij de grafische modi horen.

566,567 $236,$237 BRKKY

Wijzer voor de interrupt door de BREAK-toets. Deze kan worden veranderd, om te verwijzen naar een door de gebruiker geprogrammeerde routine die de functie van de BREAK-toets wijzigt (om bijvoorbeeld in verband met software-bescherming bij het gebruik van de BREAK-toets het programma uit het geheugen te wissen of opnieuw te laden).

568,569 $238,$239 vrij

570-575 $23A-S23F diversen.

Interne hulpvariabelen voor de verwerking van seriële data.

576-579 $240-$243 diversen

Interne hulpvariabelen voor het opstarten van de diskette-eenheid.

580 $244 COLDST

Koude-start-vlag. Iedere waarde behalve 0 geeft aan, dat de powerup initiatie-routine loopt. Staat COLDST op 0, dan volgt na RESET een warme start.

581 $245 ...

Reserve-byte

582 $246 DSKTIM

Disk time-out register.

583-622 $247-$26E LINBUF

Regel-buffer van 40 bytes. Wordt gebruikt om tijdelijk een fysieke regel op te slaan als de editor beeld-data aan het verschuiven is.

623 $26F GPRIOR

Parallel-register van adres 53275. Bepaalt bij PM-graphics de prioriteit van de weergave van verschillende beeldelementen, als deze elkaar overlappen. Veroorzaakt optisch een voor-achter effect. Overlappende players kunnen een derde kleur aannemen, de vier missiles kunnen als vijfde player worden gebruikt.

(afkortingen: P=player, PF=speelveld, BAK=achtergrond en rand)

Bit 7 (128) *): GTIA-modi:
Bit 6 (64) *): GTIA-modi.
Bit 5 (32): overlappende Ps krijgen derde kleur.
Bit 4 (16): vijfde P in plaats van vier missiles.
Bit 3 (8): prioriteitsvolgorde: PF 0-1, PF 2-3, BAK.
Bit 2 (4): prioriteitsvolgorde: PF 0-1, P0-3, BAK.
Bit 1 (2): prioriteitsvolgorde: P 0-1, PF 0-3, P 2-3, BAK.
Bit 0 (1): prioriteitsvolgorde: P 0-3, PF 0-3, BAK.

*) Bits 7 en 6 maken de GTIA-modi 9, 10 en 11 mogelijk. Deze grafische modi gebruiken dezelfde display list als GRAPHICS 8. De beeldscherm-data worden alleen anders geïnterpreteerd. In plaats van een bit per pixel te gebruiken en daarmee 320 pixels per regel in twee kleuren weer te geven, worden in deze drie GTIA-modi vier bits per pixel gebruikt. Daarmee zijn maximaal zestien verschillende kleuren mogelijk. Om op hetzelfde aantal data te komen, wordt iedere pixel vier maal zo breed weergegeven, zodat 80 pixels een modus-regel vullen.

Bit 7 uit, bit 6 aan: GRAPHICS 9 (pixels in dezelfde kleur, maar in zestien verschillende helderheden mogelijk).

Bit 7 aan, bit 6 uit: GRAPHICS 10 (pixels in negen vrij definieerbare kleuren mogelijk, maar slechts negen kleurregisters ter beschikking).

Bit 7 aan, bit 6 aan: GRAPHICS 11 (pixels in de zestien standaard-kleuren mogelijk, de helderheid is voor allemaal dezelfde).

Wanneer bit 5 aan is, ontstaat een derde kleur, als player 0 en 1 of player 2 en 3 elkaar overlappen. De twee bijbehorende kleurregisters worden door een logische OR met el-

kaar verbonden. Overlappingen van andere player-combinaties leveren geen nieuwe kleur op. Wanneer bit 5 uit is, dan wordt het gebied op het beeldscherm, dat door twee players bezet is, zwart.

Door het aan zetten van bit 4 wordt het PM-geheugengebied voor de vier missiles behandeld als vijfde player.

Het is heel goed mogelijk, in de bits 0 t/m 3 tegenstrijdige prioriteiten te definiëren. Als daardoor een conflict ontstaat, wordt het betreffende gebied van het beeldscherm zwart.

624 $270 PADDLO

Bevat de waarde van paddle 0. Paddles worden ook vaak potjes (potentiometers, draairegelaars) genoemd.

625-631 $271-$277 PADDL1-7

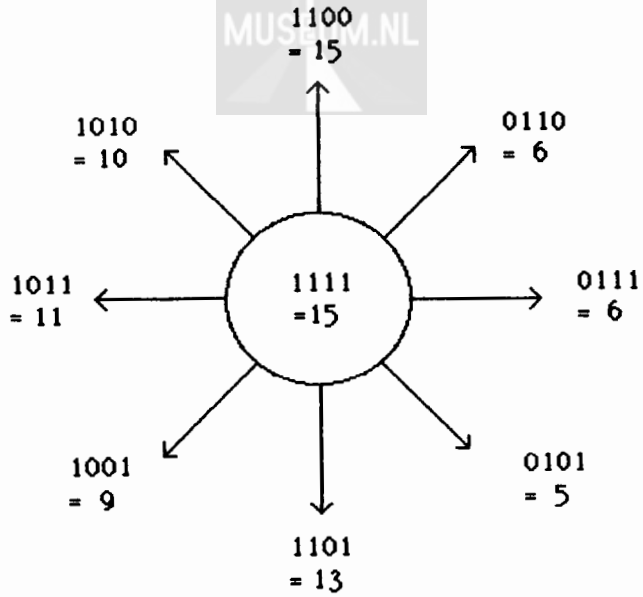
Bij de XL-modellen worden alleen nog de paddles 0 t/m 3 gebruikt.

632 $278 STICKO

Bevat de waarde van een stuurknuppel (joystick) 0. Dit register kan negen verschillende getalwaarden aannemen. Alleen de laagste vier bits worden gebruikt.

Bits 3 t/m 0 aan (0000 1111 = 15) stuurknuppel ruststand.
Bit 3 uit (0000 0111 = 7) stuurknuppel naar rechts.
Bit 2 uit (0000 1011 = 11) stuurknuppel naar links.
Bit 1 uit (0000 1101 = 13) stuurknuppel naar onderen.
Bit 0 uit (0000 1110 = 14) stuurknuppel naar boven.

Bits 3 en 1 uit (0101 = 5) stuurknuppel naar rechtsonder.
Bits 3 en 0 uit (0110 = 6) stuurknuppel naar rechtsboven.
Bits 2 en 1 uit (1001 = 9) stuurknuppel naar linksonder.
Bits 2 en 0 uit (1010 = 10) stuurknuppel naar linksboven.



633-635 $279-$27B STICK1-3

Als stuurknuppel 0. Bij XL-modellen alleen STICK0 en 1.

636 $27C PTRIGO

Geeft aan, of de vuurknop bij draairegelaar 0 is ingedrukt. Bevat een 1, wanneer de knop niet is ingedrukt en een 0 wanneer dit wel zo is.

637-643 $27D-$283 PTRIG1-7

Als PTRIGO. XL alleen 0 t/m 3.

644 $284 STRIGO

Als PTRIGO, maar dan voor stuurknuppel 0.

645-647 $285-$287 STRIG1-3

Als STRIGO. XL alleen 0 en 1.

648-655 $288-$28F diversen

Worden gebruikt door het OS.

656 $290 TXTROW

In de gesplitste scherm-modi (grafisch met tekstraam) wordt het grafische raam door de display-driver ('S:') bestuurd, terwijl de editor ('E:') het tekstraam bestuurt. Er worden gescheiden IOCBs gebruikt en er zijn twee onafhankelijke cursors. De grafische data voor het tekstraam worden bovendien in een gereserveerd geheugengebied opgeslagen, dat van het overige beeldscherm-geheugen is gescheiden. (Zie hoofdstuk beeldscherm-geheugen.)

TXTROW bevat de regel van de tekstraam-cursor. Omdat het tekstraam slechts vier regels groot is, kunnen hier slechts waarden staan van 0 t/m 3. Wordt, door verandering van de display-list, het tekstraam uitgebreid tot vijf of meer regels, dan treden dezelfde problemen met ontoelaatbare cursor-posities op, als we al behandeld hebben bij GRAPHICS 7 1/2 voor register DINDEX (87 = $52).

657,658 $291,$292 TXTCOL

Kolom van de tekstraam-cursor. Bevat een waarde van 0 t/m 39. Bij alle standaard display-lists is de HI-byte (658) dus 0.

De BASIC-instructies POSITION, PLOT en LOCATE hebben alleen betrekking op de cursor in het grafische raam. Deze instructies werken dus niet in het tekstraam en kunnen alleen door overeenkomstige POKEs worden vervangen. (Zie hoofdstuk beeldscherm-geheugen.)

659 $293 TINDEX

Bevat de actuele grafische modus van het tekstraam. Dit register is te vergelijken met DINDEX (87 = $57). Als SWPFLG 0 is, dan is dit register ook 0.

TINDEX wordt geïnitieerd op 0. De grafische modus van het tekstraam kan echter door verandering in de DL willekeurig worden geprogrammeerd. Wordt dat gedaan, dan moet de waarde in dit register worden aangepast.

660,661 $294,$295 TXTMSC

Begin-adres van het beeldscherm-geheugen voor het tekstraam, dat een eigen gebied in beslag neemt dat onafhankelijk is van het beeldscherm-geheugen voor het grafische raam. Daardoor is het ook mogelijk, het tekstraam in en uit te schakelen, zonder dat data verloren gaan van het beeldscherm-gebied, dat afwisselend door het tekstraam en dan weer door het grafische raam wordt gebruikt.

TXTMSC wijst naar de geheugenplaats, die de beeld-data bevat voor de linker bovenhoek van het tekstraam. Dit register is te vergelijken met SAVMSC (88,89 = $58,$59) en kan worden beïnvloed zoals bij SAVMSC (88,89 = $58,$59) staat beschreven.

662-667 $296-$29B TXTOLD

Deze registers voor de cursor-gegevens van het tekstraam zijn te vergelijken met OLDROW (90 = $5A), OLDCOL (91,92 = $5B,$5C), OLDCHR (93 = $5D) en OLDADR (94,95 = $5E,$5F).

668-671 $29C-$29F diversen

Tijdelijke registers.

672 $2A0 DMASK

Masker voor het vastleggen van de bits in een grafische data-byte, die bij een pixel horen. Afhankelijk van de grafische modus kunnen een, twee, vier of acht pixels in een byte worden samengevoegd. Het masker bevat nullen voor alle bits, die niet worden gebruikt voor de besturing van de op een bepaald moment weer te geven pixel. Het masker bevat enen voor de bits, die wel overeenkomen met de aktuele pixel.

Afhankelijk van de grafiche modus wordt het volgende masker aan gezet:

1111 1111 GRAPHICS 0, 1, 2 8 bits/pixel = 1 pixel/byte

1111 0000 GRAPHICS 9,10,11 4 bits/pixel = 2 pixels/byte

0000 1111

1100 0000 GRAPHICS 3, 5, 7, 2 bits/pixel = 4 pixels/byte
0011 0000 12,13,15
0000 1100
0000 0011

1000 0000 GRAPHICS 4, 6, 8, 1 bit/pixel = 8 pixels/byte
0100 0000
0010 0000
t/m
0000 0010
0000 0001

673 $2A1 TMPLBT

Tijdelijk register voor het bit-masker.

674 $2A2 ESCFLG

Escape-vlag. Normaal op 0, maar wordt op 128 gezet (bit 7 aan) als de ESCAPE-toets is ingedrukt. Wordt na de uitvoer van het volgende karakter meteen weer op 0 gezet. Moeten ATASCII control-tekens worden weergegeven zonder ESCAPE, dan kan DSPFLG (766 = $2FE) op een waarde worden gezet ongelijk aan 0.

675-689 $2A3-$2B1 TABMAP

Masker voor de TABulator-stops.

Het TAB-masker heeft betrekking op de logische regel, die bij GRAPHICS 0 drie fysieke regels bevat van ieder 40 kolommen. Dat zijn dus 120 kolom-posities. Het TAB-masker wordt opgeslagen in vijftien bytes (15*8 bits = 120). Ieder bit, die aan is, veroorzaakt een TAB-stop. De TAB-stops zijn voor alle logische regels hetzelfde, zolang ze niet opnieuw worden veranderd. Ze kunnen echter in drie opeenvolgende fysieke regels worden neergezet en van elkaar worden onderscheiden.

Via het toetsenbord kan het TAB-masker worden ingesteld met de toetsen TAB-SET en TAB-CLR. De linker rand van het beeldscherm LMARGN (82 = $52) werkt eveneens als TAB-stop.

De standaardwaarde voor alle bytes van TABMAP is 1, hetgeen TAB-stops veroorzaakt in de kolommen 7, 15, 23, enzovoorts. De standaardwaarden worden neergezet door RESET, iedere GRAPHICS-instructie en OPEN bij 'S:' of 'E:'. Het tabulator-masker werkt ook bij een tekstraam.

Moet het TAB-masker met POKE worden veranderd, dan moeten de gewenste TAB-stops als bits in de verschillende bytes aan worden gezet, hun gewichten moeten worden opgeteld en de zo verkregen waarde moet worden gePOKEd.

Voorbeeld van een TAB-masker:

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5 enz.
00001000	01000010	00010000	10000100	00100001	00001000 enz.
= 8	= 66	= 16	= 132	= 33	= 8 enz.

Om in iedere vijfde kolom een TAB-stop te hebben, moeten in de registers 675 en de volgende de decimale waarden 8, 66, 16, 132, 33, 8, enzovoorts worden neergezet.

Een tabulator-sprong wordt door het indrukken van de toets TAB uitgewist en kan met PRINT 'ESC' TAB worden geprogrammeerd.

690-693 $2B2-$2B5 LOGMAP

Masker voor het begin van de logische regels.

Dit masker bestaat uit vier bytes, waarvan de laatste niet wordt gebruikt. Ieder van de (3*8=) 24 bits staat voor een modus-regel van GRAPHICS 0. Begint in de desbetreffende fysieke regel een logische regel, dan wordt dat bit aan gezet.

Fysieke regels met betrekking tot de bits van de bytes 690 t/m 692:

byte	bit	7	6	5	4	3	2	1	0
690		0	1	2	3	4	5	6	7
691		8	9	10	11	12	13	14	15
692		16	17	18	19	20	21	22	23

694 $2B6 INVFLG

Vlag voor geïnverteerde karakters. Wordt geïnitieerd op 0. Door het indrukken van de toets INVERS (#: 'ATARI'-toets) wordt hier bit 7 (= 128) aangezet.

De display-driver verbindt de waarde van het, via het toetsenbord ingevoerde, karakter met de waarde in dit register door een logische OR. Zolang INVFLG slechts de waarde 0 of 128 aanneemt, blijft de oorspronkelijke waarde van het karakter onaangetast, want de karakterset bestaat slechts uit 128 karakters. Door 128 op te tellen bij de waarde van het karakter, wordt hetzelfde karakter in inverse weergave opgeroepen.

Er kunnen echter ook andere waarden (= bit-patronen) in dit register worden neergezet, waardoor ook de bits 6 t/m 0 aan worden gezet. Daardoor wordt de decimale waarde veranderd en ontstaat er een ander karakter, namelijk met de veranderde waarde en niet

het karakter van de eigenlijk ingedrukte toets. Daarbij moet er wel op worden gelet, dat de invertering gebaseerd is op de toets-code en niet op de ATASCII-waarde!

Het bit-patroon van de toets-code van het opgeroepen karakter wordt dus geORd met het bit-patroon in INVFLG. De zo ontstane waarde wordt dan omgezet in het overeenkomstige ATASCII-karakter en op het beeldscherm weergegeven. INVFLG werkt alleen bij direkte invoer. Het register moet voor de invoer worden ingesteld. Met dit programma kunt u uw ATARI veranderen in een schrijfmachine met geheimschrift:

```
0 REM ADR694.BEC
1 REM ********************************
2 REM *                              *
3 REM *   GEHEIMSCHRIFT-SCHRIJVER    *
4 REM *                              *
5 REM ********************************
10 DIM T$(1):? CHR$(125)
20 TRAP 20
30 ? :? "Decimale waarde voor INVFLG:"
40 ? :? "(van 1 t/m 127)":?
50 INPUT J
60 IF J<1 THEN 20
70 IF J>127 THEN 20
80 POKE 694,J
90 TRAP 90
100 ? :? "Probeer nu het toetsenbord maar eens!":? :?
110 INPUT T$
```

programma ADR694.BEC

10: De dimensionering van T$ is willekeurig, omdat er met T$ niets wordt gedaan.

30 t/m 70: In J wordt de decimale waarde opgeslagen, die in INVFLG wordt gePOKEd. De waarde 128 is niet interessant, omdat deze de negatieve weergave van hetzelfde karakter oproept. Waarden groter dan 128 hebben dezelfde werking als dezelfde waarde min 128. De bit-patronen zijn hetzelfde, alleen in het ene geval is bit 7 aan en in het andere geval niet. Het aan- of uitzetten van bit 7 kan echter alleen nog gebeuren door op de toets INVERS te drukken.

80: Hier wordt J gePOKEd.

110: Nu kunt u willekeurige toetsen indrukken en u verbazen over de karakters die op het beeldscherm verschijnen. Probeert u ook eens de toetsen SHIFT, CONTROL en INVERS. RETURN beëindigt het programma.

695 $2B7 FILFLG

FILL-aanduiding voor de DRAWTO-instructie. Is het lopende proces een DRAWTO, dan staat dit register op 0. Is het een FILL (XIO 18-instructie), dan staat hier een waarde ongelijk aan 0.

696-698 $2B8-$2BA TMPROW/COL

Tijdelijke registers voor ROWCRS en COLCRS.

699 $2BB SCRFLG

Aanduiding voor het scrollen per regel. Telt het aantal fysieke regels min 1, die aan de bovenkant van het beeldscherm worden gewist. Omdat een logische regel maximaal 3 fysieke regels bevat, kan SCRFLG de waarden 0 t/m 2 aannemen.

Als het tekstraam wordt gescrolld, dan is het alsof het totale GRAPHICS-0-beeldscherm in het geheugen naar boven wordt verplaatst (800 bytes!). Dat kan leiden tot het over data heen schrijven, bijvoorbeeld PM-graphics data, karaktersets (#), die boven RAMTOP staan. Bij twijfelgevallen is het veiliger om een niet gebruikt gebied vrij te houden van 800 bytes boven RAMTOP, waar het scrollen kan uitrollen.

700,701 $2BC,$2BD HOLD4,5

Tijdelijke registers tijdens het FILL-proces.

702 $2BE SHFLOK

Aanduiding voor de SHIFT- en de CONTROL-toets.

Bit 7 aan: CTRL-toets werd ingedrukt, CONTROL-codes en pseudo-grafische tekens worden weergegeven. Voorwaarde voor de weergave van lettertekens (hoofdletters en kleine letters) is, dat bit 7 uit is.

Bit 6 aan: SHIFT-toets ingedrukt. Omdat SHFLOK op 64 wordt geïnitieerd, staat de SHIFT-functie in de normale toestand voortdurend aan. Vandaar de naam van het register: SHIFT-lock (vergrendeling). Met de toets CAPS (#: CAPS LOWR) kan bit 6 op 0 worden gezet. In deze schrijfmachine-modus moet dan op SHIFT worden gedrukt, om een hoofdletter op te roepen.

703 $2BF BOTSCR

Geeft het aantal mogelijke tekstregels aan voor PRINT.

24 is de normale waarde voor GRAPHICS 0.
4 is de waarde voor het tekstraam.
0 staat hier bij alle grafische modi zonder tekstraam.
De gekleurde karakter-modi (GRAPHICS 1, 2, 12, 13) zijn hierbij inbegrepen.

De display-driver test in dit register, of een gesplitst beeldscherm (grafisch/tekst) is opgeroepen. Daardoor is het mogelijk om ook in GRAPHICS 0 een tekstraam in te richten (POKE 703,4). De bovenste 20 regels zijn dan alleen met een PRINT-statement in het grafische raam (PRINT #6) te beschrijven. Dit bovenste deel blijft ook staan, wanneer het tekstraam scrollt.
Deze techniek wordt gebruikt bij het DOS-menu.

704 $2C0 PCOLRO

Kleurenregister voor player 0 en missile 0.

De vier kleurenregisters PCOLRO-3 worden alleen uitgelezen bij PM-graphics en in de GTIA-modus GRAPHICS 10. De BASIC-instructie SETCOLOR beïnvloedt de registers 704 t/m 707 niet. Kleurwaarden kunnen hiernaar toe alleen worden gePOKEd.

ATARI kan zestien kleuren weergeven, die met de getallen 0 t/m 15 zijn gekenmerkt. In tegenstelling tot andere homecomputers uit dezelfde prijsklasse kunnen zestien verschillende helderheden worden ingesteld van 0 = donker t/m 15 = helder. Op die manier ontstaat dus een totaal van 256 verschillende kleurwaarden, waaruit er, afhankelijk van de grafische modus, slechts een tot maximaal zestien gelijktijdig op het beeldscherm zijn weer te geven (zonder DLI te gebruiken).

De kleurwaarde is te bepalen volgens de formule:

kleurwaarde = kleur * 16 + helderheidsgraad

Dat betekent, dat de kleur (van 0 = 0000 t/m 15 = 1111) in de hoge nibble staat, en de helderheidsgraad (eveneens van 0 = 0000 t/m 15 = 1111) in de lage nibble van een kleurenregister.

Omdat bit 0 van een kleurenregister niet wordt uitgelezen, zijn eigenlijk maar acht helderheids-gradaties (alleen even waarden) beschikbaar. Alleen in de GTIA-modus GRAPHICS 9 kunnen alle zestien helderheids-gradaties worden gebruikt.

Als in register GPRIOR (623 = $26F) bit 5 aan wordt gezet (derde kleur bij overlapping), dan ontstaat de kleurwaarde voor deze overlappingskleur door de verbinding via een logische OR:

eerste kleurwaarde oud-rose = decimaal 52 = binair 0011
0100
tweede kleurwaarde olijfgroen = decimaal 226 = binair 1110
0010

OR-verbonden reebruin = decimaal 246 = binair 1111
0110

De verbinding via de logische OR zorgt ervoor, dat de kleur in het overlappende gebied altijd een hogere kleurwaarde heeft dan de beide afzonderlijke kleuren, dus een kleur met een hoger rangnummer en een grotere helderheid.

705-707 $2C1-$2C3 PCOLR1-3

Kleurwaarde voor player 1 t/m 3 en missile 1 t/m 3.

708 $2C4 COLOR0

De volgende vijf kleurenregisters worden in de verschillende grafische modi uitgelezen. Ze kunnen met de BASIC-instructie SETCOLOR beschreven worden. Deze instructie heeft het formaat:

SETCOLOR r, k, h

r kan waarden aannemen van 0 t/m 4 en heeft betrekking op het kleurenregister COLOR0 (708 = $2C4) t/m COLOR4 (712 = $2C8).

k kan waarden aannemen van 0 t/m 15 en heeft betrekking op de zestien standaard kleuren.

h kan even waarden aannemen van 0 t/m 14 en bepaalt de helderheid.

De SETCOLOR-instructie is waarschijnlijk de meest overbodige instructie van ATARI-BASIC, omdat hij meer tijd kost dan een POKE-instructie, die hetzelfde doet. Omdat men de waarden voor k en h toch experimenteel moet bepalen, kan men ook gelijk de kleurwaarde (k*16+h) in het register zetten. Vooral wanneer meer dan één kleurwaarde gedefinieerd moet worden, kunnen met POKE enige geheugenplaatsen in een BASIC-programma worden bespaard.

709-712 $2C5-$2C8 COLOR1-4

Kleurenregisters 1 t/m 4.

In de verschillende grafische modi vervullen de kleurenregisters verschillende taken en wordt COLOR door verschillende BASIC-instructies aangesproken. Een overzichts-tabel hiervoor vindt u in het hoofdstuk beeldscherm-geheugen.

Bij het inschakelen krijgen de COLOR-registers verschillende waarden, die na technische overwegingen uitgekozen zijn vanwege hun optimale onderscheidingsvermogen, dus niet uit esthetisch oogpunt (wat eenvoudig te constateren is):

register	kleurwaarde	kleur	helderheid
708 COLOR0	40	2	8
709 COLOR1	202	12	10
710 COLOR2	148	9	4
711 COLOR3	70	4	6
712 COLOR4	0	0	0

Het volgende programma geeft een eenvoudig voorbeeld, welke effecten door het gebruik van de kleurenregisters mogelijk zijn:

```
0 REM COLPOK.BEC
1 REM ********************************
2 REM *                              *
3 REM * VLIEGENDE KLEURENWISSELAAR   *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 10
20 FOR J=1 TO 8
30 POKE 704+J,J*4
40 NEXT J
50 FOR C=0 TO 8
60 COLOR C
70 FOR I=0 TO 7
80 PLOT C*8+I,0:DRAWTO C*8+I,191
90 NEXT I
100 NEXT C
200 A=PEEK(705)
210 FOR J=0 TO 6
220 POKE 705+J,PEEK(706+J)
230 REM FOR W=0 TO 200:NEXT W
240 NEXT J
250 POKE 712,A
260 GOTO 200
```

programma COLPOK.BEC

Vanwege vele mogelijkheden wordt de grafische modus 10 gebruikt, die naast de kleurwaarde voor de achtergrond nog acht vrij definieerbare kleuren toelaat.

In dit programma roteren de kleurwaarden in de registers 705 t/m 712. 704 bepaalt de kleur voor de achtergrond en is op 0 (= zwart) gezet.

20 t/m 40: De FOR-NEXT lus zet in de kleurregisters 705 '704+1) t/m 712 (704+8) kleurwaarden van 4 (1*4) t/m 32 (8*4). Het register 704 hoeft niet te worden beschreven, omdat zijn standaardwaarde 0 overeenkomt met de gewenste kleurwaarde. Natuurlijk kan ook ieder kleurregister afzonderlijk een willekeurige waarde worden gegeven. Voor het gewenste effect komen de in helderheid afnemende kleuren echter goed van pas.

50 t/m 100: vullen het GRAPHICS 10 beeldscherm met brede evenwijdige kleurbalken in de volgorde van de kleurenregisters.

200: Hier wordt de kleurwaarde van register 705 in A opgeborgen.

210: Dan worden de kleurwaarden van de zeven registers (J=0 TO 6) omgewisseld.

220: Waarin in het voorgaande register de waarde van het volgende register wordt geschreven.

250: Tenslotte krijgt het laatste register (712) de waarde van het eerste (705), die in A is opgeslagen.

Wanneer u het programma zo laat lopen, ontstaat de indruk van een draaiende trommel. De draaisnelheid kunt u laten afnemen, door in regel 230 de REM weg te halen. Hoe hoger hier de lusteller wordt gedefinieerd, des te langzamer 'draait' de trommel.

Een ander indrukwekkend effect is de kleurwaarde in een register met korte tussenpozen te veranderen:

```
260 X=X-2:IFX<0 THEN X=254
270 POKE 704,X
280 GOTO 200
```

Voeg deze drie regels toe aan de voorgaande LISTing en de achtergrond begint in verschillende kleuren te knipperen. Als u eens wilt zien, hoe snel het veranderen van de kleurwaarden met BASIC-POKE's gaat, verandert u dan de sprong in regel 280 van 200 in 260.

729 $2D9 KEYDEL

Bepaalt de tijdsduur, tot de herhalingsfunctie van het toetsenbord in werking treedt. De standaard waarde is 40. Met de waarde 0 wordt de herhalingsfunctie uitgeschakeld. Met stijgende waarden van 1 t/m 255 wordt de vertraging groter. Bij de waarde 1 is deze zo kort, dat het onmogelijk is, door het indrukken van een toets slechts één teken op het beeldscherm te krijgen. (Alleen bij XL.)

730 $2DA KEYREP

Als de herhalingsfunctie in werking treedt na het voorbijgaan van de tijd in KEYDEL, dan bepaalt KEYREP de herhalings-frequentie. De standaardwaarde is 5. Hoe kleiner de waarde hier is, hoe sneller de herhaling is. Wordt hier een 0 neergezet, dan is slechts een eenmalige herhaling mogelijk. (Alleen bij XL.)

731 $2DB NOCLIK

Aanduiding voor de pieptoon die klinkt bij het indrukken van een toets. 0 onderdrukt de pieptoon, alle andere waarden niet. (Alleen bij XL.)

732 $2DC HLPFLG

Aanduiding voor de HELP-toets. De standaardwaarde is 0. Door het indrukken van de HELP-toets worden de bits 0 en 4 (=17) aan gezet. Wordt de HELP-toets tegelijk met SHIFT ingedrukt, dan wordt bovendien bit 6 aan gezet. Het register bevat dan de waarde 81. HELP en CONTROL zet naast de bits 0 en 4 ook bit 7 aan (=145). Worden SHIFT en CONTROL tegelijk ingedrukt, dan heeft het tevens indrukken van de HELP-toets hier geen gevolgen meer. De bits die aan gezet zijn, worden na het loslaten van de toetsen niet op 0 terug gezet. (Alleen bij XL).

740 $2E4 RAMSIZ

De grootte van de beschikbare hoeveelheid RAM. Bevat alleen de HI-byte, dus geeft de RAM-grootte in pagina's aan. Bevat dezelfde waarde als RAMTOP (106 = $6A).

741,742 $2E5,$2E6 MEMTOP

Wijzer naar de bovengrens van het vrije RAM-gebied. Deze waarde wordt vernieuwd bij powerup of RESET, door iedere GRAPHICS-instructie of iedere OPEN, die betrekking heeft op het display.

743,744 $2E7,$2E8 MEMLO

Wijzer naar de ondergrens van het vrije RAM-gebied. Wordt bijvoorbeeld veranderd door DOS, dat in het onderste geheugengebied wordt ingeladen.

MEMLO wijst naar de eerste vrije geheugenplaats voor een gebruikers-programma.

750,751 $2EE,$2EF CBAUD

Geeft de baud-rate aan voor de cassette. Wordt geïnitieerd op 1484, hetgeen overeenkomt met een baud-rate van 600 (bits/seconde). De baud-rate wordt ingesteld door S10. Iedere cassette-file begint met een patroon van aan/uit-bits, waarmee de cassette-driver de baud-rate aanpast.

752 $2F0 CRSINH

Aanduiding voor de cursor-onderdrukking. Wordt op 0 gezet bij powerup, RESET, BREAK en OPEN 'S:' of 'E:', hetgeen een zichtbare cursor veroorzaakt. Iedere waarde ongelijk aan 0 maakt de cursor na de eerstvolgende cursor-beweging onzichtbaar.

753 $2F1 KEYDEL

Aanduiding voor de toets-vertraging. Heeft de waarde 0, als er geen toets is ingedrukt. Wordt er een toets ingedrukt, dan wordt hier de waarde 3 neergezet. Bij iedere VBLANK wordt hiervan 1 afgetrokken. De volgende toets-invoer is pas weer mogelijk als hier weer een 0 staat. KEYDEL bepaalt dus de minimale tijd tussen twee aanslagen.

754 $2F2 CHI

Bevat de waarde van de op een na laatste toets die werd ingedrukt. Deze waarde komt van CH (764 = $2FC). Als de waarde van de nieuwe toetscode gelijk is aan deze waarde, dan wordt die code pas geaccepteerd als KEYDEL weer op 0 staat.

755 $2F3 CHACT

Register voor de weergave van de karakters. Wordt hier een 0 neergezet, dan worden alle inverse tekens als normale tekens behandeld. Daardoor wordt ook de cursor onzichtbaar. Een 1 zorgt ervoor dat alle inverse tekens als spaties worden weergegeven. De cursor is dan ook onzichtbaar, maar het teken onder de cursor verdwijnt.

De standaardwaarde is 2.

Met een 3 worden alle inverse tekens als inverse spaties weergegeven.

Door het aanzetten van bit 2 (=4) worden alle tekens ondersteboven weergegeven.

In de karakter-modi GRAPHICS 1, 2, 14 en 15 zorgen de hiervoor gegeven waarden er alleen voor, dat de tekens ondersteboven worden weergegeven.
Maar kijk eens naar de uitwerking van de verschillende waarden bij GRAPHICS 0:

```
0 REM ADR755.BEC
1 REM ********************************
2 REM *                              *
3 REM *    VERDRAAIDE KARAKTERS      *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 0:POKE 703,4
20 ? "321 ööÄáCBA ¿¡¡á"
30 FOR J=0 TO 255
40 POKE 755,J
50 ? £6;J;" ";
60 OPEN £1,4,0,"K:"
70 GET £1,D
80 CLOSE £1
90 NEXT J
```

Let op! Listing bevat inverse tekens die beslist noodzakelijk zijn om het programma te kunnen begrijpen!

10: In GRAPHICS 0 wordt een tekstraam gePOKEd.

20: Normale en inverse tekens worden in het grafische raam gePRINT.

30: De FOR-NEXT lus loopt alle mogelijke decimale waarden af, die in een byte neergezet kunnen worden.

40: Hier wordt de waarde naar adres 755 gePOKEd.

50: In het tekstraam verschijnt ter controle de waarde, die zojuist is gePOKEd.

60 t/m 80: Pas wanneer de gebruiker een toets indrukt, gaat het programma verder en wordt de volgende waarde op adres 755 gezet.

Zoals u ziet, werken in dit register alleen de bits 0 t/m 2. Met de acht verschillende instelmogelijkheden is een grote hoeveelheid knipperende tekens te maken, als tussen twee waarden in CHACT heen en weer wordt geschakeld. Het volgende programma laat u alle 64 combinaties zien:

```
0 REM ADR755B.BEC
1 REM ********************************
2 REM *                              *
3 REM *   KNIPPER  8 X 8  KNIPPER    *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 0:POKE 703,4
20 POSITION 2,12
30 ? £6;"KNIPPER! ■^øöö%°ääKNIPPER! ■^øöö%°ää"
40 FOR I=0 TO 7
50 FOR J=0 TO 7
60 ? I;"/";J;?
70 POKE 755,I
80 FOR W=0 TO 100:NEXT W
90 POKE 755,J
100 FOR W=0 TO 100:NEXT W
110 IF PEEK(764)=255 THEN 70
120 OPEN £1,4,0,"K:"
130 GET £1,D
140 CLOSE £1
150 NEXT J
160 NEXT I
```

Let op! Listing bevat inverse tekens die beslist noodzakelijk zijn om het programma te kunnen begrijpen!

10 t/m 30: In het midden van het beeldscherm wordt de tekst weergegeven.

40 en 50: De geneste lussen geven alle acht maal acht combinaties weer, waartussen in CHACT omgeschakeld kan worden.

60: geeft de aktuele combinatie in het tekstraam weer voor controle.

80 en 100: Hier kunt u de knipper-frequentie veranderen.

110 t/m 130: Als u een toets van het toetsenbord indrukt, begint de volgende combinatie te knipperen.

756 $2F4 CHBAS

Wijzer naar het begin-adres van de ATARI standaard karakterset.

CHBAS is alleen een HI-byte wijzer. Om het eigenlijke adres te vinden moet de hier gevonden waarde met 256 worden vermenigvuldigd. De standaardwaarde in dit register is 224. De karakterset bevat 128 karakters. De negatieve (inverse) karakters (ATASCII 128 t/m 255) hebben geen speciale data nodig. Ze ontstaan eenvoudig door invertering van het bit-patroon van het normale karakter. Aanduiding voor de inverse weergave is bit 7 (=128). De ATASCII-waarde van een invers karakter is dus 128 groter dan de waarde van hetzelfde normale karakter.

In de karakter-modi GRAPHICS 1 en 2 staan slechts de helft van de karakterset ter beschikking, namelijk besturingstekens, cijfers en hoofdletters. Door het omzetten van CHBAS naar het begin van de andere helft van de karakterset, kunnen ook kleine letters worden gebruikt. Het is echter niet mogelijk, hoofdletters en kleine letters samen op een beeldscherm weer te geven (zonder de karakterset zelf te veranderen). Met POKE 756,226 wordt de wijzer omgezet.

Natuurlijk gaat het bij de standaard karakterset om een Amerikaanse karakterset. Bij de XL is er ook een Europese karakterset. Na POKE 756,204 kan deze karakterset worden gebruikt. De internationale karakters zijn op te roepen door de CONTROL-toets te gebruiken. (Zie handleiding).

Hoe de karakterset is opgebouwd en hoe men er zelf een kan ontwerpen, staat in het hoofdstuk 'Karakterset'.

760 $2F8 ROWING

Het teken (+1 of -1) van DELTAR (118 = $76). (#:121 = $79)

761 $2F9 COLINC

Het teken van DELTAC (119,120 = $77,$78). (#:122 = $80)

762 $2FA CHAR

Bevat de interne code van het laatst gelezen of geschreven karakter. BASIC is over het algemeen te langzaam om dit register uit te lezen. PEEK(762) geeft daardoor meestal de waarde 128 (zichtbare cursor = inverse spatie) of 0 (onzichtbare cursor = spatie).

763 $2FB ATACHR

Bevat de ATASCII-waarde van het laatst gelezen of geschreven karakter. Wordt gebruikt bij de omzetting van ATASCII in interne code. Bij grafisch gebruik wordt hier de color-waarde van het grafische punt opgeslagen en bij de instructies X10 17 (DRAW) en X10 18(FILL) staat hier de kleurwaarde voor de te trekken lijn.

764 $2FC CH

Bevat de toetscode van de laatst ingedrukte toets. Hier kan met PEEK(764) toetsinvoer worden uitgelezen, zonder dat op RETURN gedrukt moet worden.

Het register bevat de waarde 255, als er geen toets wordt ingedrukt. De toetscode van de overige toetsen kunt u vinden in de volgende tabel. De decimale waarden aan de linker- en de bovenrand moeten bij elkaar worden opgeteld, om de code van het karakter te vinden, dat in het snijpunt staat.

	0	1	2	3	4	5	6	7
00	L	J	;			K	+	*
08	O		P	U	RETURN	I	-	=
16	V		C			B	X	Z
24	4		3	6	ESCAPE	5	2	1
32	,	spatie	.	N		M	/	invers
40	R		E	Y	TAB	T	W	Q
48	9		0	7	BACKSP	8	(	)

Deze karakters gebruiken de bits 0 t/m 5. Onafhankelijk daarvan zet het gelijktijdig indrukken van een toets samen met de SHIFT-toets bit 6 en met de CONTROL-toets bit 7 aan. Het gelijktijdig indrukken van SHIFT en CONTROL heeft geen functie en wordt door het OS genegeerd.

De toetscode van een karakter is de offset van het teken in de toets-definitietabel, die gebruikt wordt voor het omzetten van de toetscode in ATASCII-code. De gebruiker kan een eigen definitietabel opstellen en aan iedere toets willekeurig een nieuwe ATASCII-waarde toekennen. De tabel moet echter altijd in drie stukken onderverdeeld zijn. De eerste 64 bytes worden gebruikt voor de enkele toetsen, de volgende 64 bytes voor toets + SHIFT en de laatste 64 bytes voor toets + CONTROL.

Nadat de nieuwe definitietabel in het geheugen is opgeslagen, moet de wijzer KEYDEF (121,122 = $79,$7A) naar het begin van de nieuwe tabel worden omgezet.

De toetsen RESET, BREAK, SHIFT, CONTROL, de functietoetsen OPTION, SELECT, START, HELP en de toetscombinatie CONTROL + "1" worden niet via deze tabel verwerkt en kunnen hier dus ook niet worden veranderd.

Een toepassing voor CH vindt in het voorbeeld-programma ADR755B.BEC in regel 110

765 $2FD FILDAT

COLOR-waarde voor het door X10 18 (FILL) te gebruiken gebied. Hoe de FILL-instructie toe te passen is, blijkt uit het volgende programma.

programma ADR765.BEC

10: Hier wordt GRAPHICS 15 zonder tekstraam (+16) ingeschakeld. Als u nog een oude ATARI heeft, die deze modus niet via GRAPHICS kan oproepen, kiest u dan GRAPHICS 7+16. De PLOTs moeten dan gedeeltelijk worden aangepast.

20: Kleurwaarden voor COLOR 1, 2 en 3. COLOR 0, de achtergrond, register 712, behoudt zijn standaard waarde 0 (= zwart).

30: In FILLDAT wordt voor XI0 18 de COLOR-waarde 1 opgeslagen.

40: Voor de volgende PLOTs wordt hier kleur 2 opgeroepen.

```
0 REM ADR765.BEC
1 REM ********************************
2 REM *                              *
3 REM *  XIO 18 = FILL-INSTRUCTIE    *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 31
20 POKE 708,4:POKE 709,50:POKE 710,144
30 POKE 765,1
40 COLOR 2
45 REM PLOT80,80
50 PLOT 139,95
60 DRAWTO 109,0
70 DRAWTO 49,0
80 POSITION 19,95
90 XIO 18,#6,0,0,"S:"
95 GOTO 95
100 PLOT 109,191
110 DRAWTO 139,96:REM COLOR 1
120 DRAWTO 19,96
130 POSITION 49,191
140 XIO 18,#6,0,0,"S:"
150 GOTO 150
160 COLOR 3
170 PLOT 159,191
180 DRAWTO 159,0
190 DRAWTO 0,0
200 POSITION 0,191
210 POKE 765,2
220 XIO 18,#6,0,0,"S:"
230 GOTO 230
240 PLOT 159,191:REM COLOR 2
250 DRAWTO 159,0
260 DRAWTO 110,0
270 COLOR 2:POSITION 140,95
280 XIO 18,#6,0,0,"S:"
290 PLOT 159,96
300 DRAWTO 140,96
310 POSITION 110,191
320 XIO 18,#6,0,0,"S:"
330 GOTO 330
```

50 t/m 80: De FILL-functie heeft de volgende markeringspunten nodig: een gePLOTte lijn als rechter begrenzing en een gePLOTte lijn als voorkant. Deze twee lijnen bepalen de drie hoekpunten van het te FILLen vlak: rechtsonder, rechtsboven en linksboven. Als laatste oriëntatie is het punt links onder nodig. Dit wordt als POSITION-instructie gegeven. Met de FILL-instructie kunnen in principe alleen vierhoekige vlakken met een willekeurige vorm worden gevuld. De enige voorwaarde is, dat de hoek links boven hoger ligt dan de hoek rechts boven, want

90: de FILL-instructie doet het volgende: van de hoek links boven naar het door POSITION aangegeven punt wordt een lijn gePLOT in de opgeroepen kleur. Na ieder afzonderlijk gePLOT punt wordt van dit punt met de in FILDAT opgeslagen COLOR-waarde een lijn naar rechts getrokken, totdat een reeds gePLOT punt, de rechter begrenzing, wordt bereikt. Daarna wordt het volgende punt van de linker zijkant van de vierhoek gePLOT, weer de FILL-lijn naar rechts getrokken, enzovoorts.

Om dit effect gedemonstreerd te zien, moet u in de LISTing de regel:

45 PLOT 80,80

toevoegen en een RUN geven. U ziet dan, dat wanneer de FILL-routine in regel 80 belandt, de vierhoek slechts tot kolom 79 wordt opgevuld. Voorbij kolom 80 blijft de regel in zijn oude COLOR-toestand.

95: Verwijder nu deze regel en ook regel 45 weer. Dan gaat het verder.

100 t/m 140: De zeshoek, die op het beeldscherm getekend moet worden, moet in twee trapeziums worden gesplitst. Hier volgt de onderste helft:

110: Als u het programma nu laat lopen, kunt u duidelijk zien, hoe eerst de linker- en de bovenste rand van de vierhoek worden gePLOT. Tijdens de FILL ontstaat ook nog de linkerrand. Aan de onderkant wordt de vierhoek door de randen van het FILL-vlak begrenst. Wanneer u rondom een gekleurde rand wilt hebben, dan moet de onderkant ook nog in de gewenste kleur worden gePLOT. Aan de andere kant ziet u ook, dat de bovenkant van de onderste helft van de zeshoek in de voor PLOT geldende kleur COLOR wordt weergegeven zodat er een scheidingslijn loodrecht door het midden van de zeshoek loopt. Als u deze lijn weg wilt hebben, verwijdert u dan in regel 110 de REM, zodat de COLOR-instructie in werking treedt. Het is altijd mogelijk, voor PLOT en FILL dezelfde COLOR-waarde te kiezen.

150: Nadat u de werking van dit alles hebt gezien, kunt u deze regel verwijderen, om bij de rest van het programma te komen.

160 t/m 220: Hier worden met COLOR 3 de uiterste randen van het grafische raam als begrenzingslijnen voor de FILL-instructie gePLOT. Als echter de XIO 18-instructie met COLOR 2 in werking treedt, herkent deze de reeds aanwezige linkerkant van de rechthoek als rechterbegrenzing. Daardoor wordt dus niet de hele rest van het beeldscherm opgevuld, maar alleen het vlak van de linkerrand tot aan de zeshoek.

230: Als u dat bestudeerd heeft, verwijdert u dan deze regel.

240 t/m 280: Om nu ook nog het laatste vlak rechts naast de zeshoek op te FILLen zal wat meer moeite kosten. Het is namelijk niet mogelijk, dit vlak met een instructie in te kleuren. Worden de linker en de bovenste begrenzing van het vlak (de beeldschermrand) gePLOT en wordt dan de cursor met POSITION op de uiterste plaats links onder gezet, dan wordt alleen de linkerkant getrokken. Deze kant ligt binnen de zeshoek. De FILL-instructie zou binnen de zeshoek beginnen en de rechterkant van de zeshoek als rechter begrenzing herkennen. Een eenmaal geFILLd vlak kan echter niet nog eens door FILL worden veranderd. Het rechter restvlak moet dus in twee gedeelten worden opgevuld. De regels 240 t/m 280 verzorgen de bovenste helft. De storende linkerkant kan door het verwijderen van de REM in regel 240 in kleur worden aangepast.

Het trekken van de rechterbegrenzing (beeldschermrand) is overigens overbodig, omdat het in de regels 100 en 110 bij de vruchteloze poging al gelukt is, het hele restvlak in een keer op te vullen. In de regels 290 t/m 320 wordt daarom terecht daarvan afgezien.

766 $2FE DSPFLG

Display-aanduiding voor de behandeling van de stuurtekens. Stuurtekens als cursor-verplaatsingen, wissen van het beeldscherm, INSERT, DELETE, BACKSPACE of TAB kunnen op twee verschillende manieren in een BASIC-programma worden gebruikt. In de vorm:

PRINT (CHR$(n)

wordt de opgeroepen functie uitgevoerd, wanneer voor n de gewenste waarde wordt ingevuld. Dus als n=125 wordt het beeldscherm geCLEARd. In veel van de in dit boek beschreven voorbeeld-programma's duikt deze uitdrukking op. Deze vorm heeft als voordeel, dat iedere aangesloten printer deze tekens zonder enig probleem op papier zet.

De tweede mogelijkheid bestaat hieruit, de gewenste functie met de bijbehorende toets te programmeren. De BASIC-instructie heeft dan de volgende vorm, waarbij de aanduidingen tussen haakjes betekenen, dat u bij het intypen de desbetreffende toets moet aanslaan:

PRINT '(ESCAPE)(CLEAR)'

Wanneer u dat invoert, verschijnt op het beeldscherm een grafisch symbool, in dit geval een kleine, gebogen, naar links wijzende pijl. Dat kan voor de ongeoefende programmeur gemakkelijker zijn, omdat hij of zij de bij de gewenste functie behorende ATASCII-waarde niet hoeft te onthouden en bij het programmeren gewoon de toetsen gebruikt, die hij of zij ook bij de direkte invoer gebruikt. De ESCAPE-toets werkt overigens alleen voor de direkt daarop volgende toets. Wilt u meerdere besturingstekens achter elkaar invoeren, dan moet u de ESCAPE-toets steeds weer opnieuw indrukken.

Wanneer u uiteindelijk uw mooie programma op de printer wilt laten uitLISTen, zult u deze gemakkelijke methode vervloeken, want de printer herkent alleen de ontvangen waarden, die hij op zijn eigen manier interpreteert, als ze niet in de ASCII-norm zijn vastgelegd. Het is dan ook vanzelfsprekend, dat hij gek gaat doen. Een (ESCAPE)(CLEAR) ziet de printer nog als een recht haakje. Bij de codes groter dan 128 (bijvoorbeeld TAB) kan ieder model printer voor een andere verrassing zorgen.

Hetzelfde geldt voor de pseudo-grafische tekens met de ATASCII=waarden 0 t/m 31. Dat zijn namelijk de belangrijkste besturingstekens voor alle printers (ASCII-norm). En dan be-

gint de printer ineens met TAB-sprongen, het uitspugen van papier (line feed en paper feed), wagenterugloop (carriage return) en iedere keer klinkt de ingebouwde pieptoon of zoemer (bell). Totnutoe is het ATARI nog niet gelukt om een printer te ontwerpen die de hele ATASCII-karakterset op papier kan zetten en er is nog steeds geen enkele interface op de markt die dat kan. Het is daarom aan te bevelen om in programma's die u wilt laten uitPRINTen, al deze tekens te vervangen door de desbetreffende CHR$-instructies, die hetzelfde doen als de als string ingevoerde pseudo grafische of besturings-tekens.

Wanneer in DSPFLG een andere waarde dan 0 staat, dan wordt het desbetreffende besturingsteken op het beeldscherm weergegeven met het overeenkomstige grafische teken, dus alsof de ESCAPE-toets werd gebruikt. 0 is de standaardwaarde in dit register.

767 $2FF SSFLAG

Start/stop-aanduiding voor de beeldscherm-uitvoer. Zolang hier een 0 staat, gaat de uitvoer ongehinderd verder. Met de waarde 255, (invertering van 00000000), wordt de beeldscherm-uitvoer onderbroken en het systeem wacht, totdat hier weer een 0 wordt neergezet. SSFLAG wordt door het gelijktijdig indrukken van CONTROL en '1' geïnverteerd, dus van 0 naar 255 of omgekeerd.

768 $300 DDEVIC

Identificatie-code voor de aanduiding van randapparaten.

768 $301 DUNIT

Apparaat-nummer, kan door de gebruiker worden bepaald.

770 $302 DCOMND

Commando-byte. Bevat het nummer van de uit te voeren bewerking.

771 $303 DSTATS

Apparaat-status.

772,773 $304,$305 DBUFLO/HI

Adres van de data-buffer.

774 $306 DTIMLO

Wachttijd voor de apparaat-driver tot de timeout foutmelding.

775 $307 DUNUSE

Ongebruikte byte.

776,777 $308,$309 DBYTLO/HI

Aantal bytes in de buffer, waar DBUFLO/HI naar wijst.

778,799 $30A,$30B DAUX1/2

Hulpinformatie. Bij diskette-bewerkingen bijvoorbeeld voor de LO- en HI-bytes van het sectornummer.

780,781 $30C,$30D TIMER1.

Baud-rate timer. Zie verder TIMER2.

783 $30F CASFLG

Ongelijk aan 0, als de lopende bewerking aan de cassette gericht is.

784,785 $310,$311 TIMER2

Interval-timer 2. TIMER1 en TIMER2 worden gebruikt, om de baud-rate bij cassette-bewerkingen in te stellen. Ze vergelijken de standaardwaarde met het bit-patroon aan het begin van een cassette-file. Het verschil tussen de twee timers wordt gebruikt, om in een tabel de correctie-waarde te vinden, waarna dan de juiste baud-rate aan CBAUDL/H (750,751 = $2EE,$2EF) wordt afgegeven.

791 $317 TIMFLG

Timeout-aanduiding

792 $318 STACKP

S10 stapelwijzer (stack pointer).

793 $319 TSTAT

Tijdelijk register voor SIO status-informatie.

794-831 $31A-$33F HATABS

Tabel met de driver-adressen. Dient voor de verbinding van apparaten met hun drivers. Er kunnen maximaal 38 verbindingen worden gemaakt, die ieder uit drie bytes bestaan. Het eerste byte bevat de apparaat-namen (C,D,E,K,P,S,R) in ATASCII-code. Bytes twee en drie bevatten de LO- en HI-byte van het driver begin-adres. Niet gebruikte bytes staan op 0.

832-847 $340-$34F IOCBO

I/O-controleblok 0. Wordt normaal gebruikt voor de beeldscherm-editor ('E:'). In GRAPHICS-modi is kanaal 0 voor het tekstraam geopend. Als geen tekstraam wordt gebruikt en kanaal 0 wordt geopend, dan gaat het hele beeldscherm over op de editor-modus (GRAPHICS 0). Dat gebeurt bijvoorbeeld om de READY weer te geven, als een grafisch programma zonder tekstraam is afgelopen.

De BASIC-commando's NEW en RUN sluiten alle kanalen behalve 0. Wordt een kanaal naar 'S:' of 'E:' geopend, dan wordt het hele beeldscherm-geheugen gewist.

Als in een programma door invoer van de gebruiker bepaald moet worden, of data op het beeldscherm of op de printer weergegeven moeten worden, kan de dubbele programmering van alle mogelijke uitvoer met PRINT- en LPRINT-instructies worden vermeden. Het is voldoende, de wijzer op de adressen 838 (= $340, LO) en 839 (= $34F, HI) te veranderen. Normaal bevat deze wijzer de waarden 163 en 246. Worden hier de waarden 166 (LO) en 238 (HI) neergezet, dan wordt naar de printer gestuurd, wat normaal naar de editor zou gaan.

842 $34A ICAX1

Dit adres is van grote betekenis, omdat hierdoor het lezen van het beeldscherm mogelijk is. De standaardwaarde hier is 12. In deze toestand wordt naar het beeldscherm geschreven. Wordt hier de waarde 13 neergezet, dan wordt de RETURN-toets-modus ingeschakeld, die het lezen van het beeldscherm mogelijk maakt.

Dat werkt zo:

```
0 REM ADR842.BEC
1 REM ****************************
2 REM *                          *
3 REM * LEZEN VAN HET BEELDSCHERM *
4 REM *                          *
5 REM ****************************
10 GRAPHICS 0:POSITION 2,4
20 ? 1000
30 FOR W=0 TO 500:NEXT W
```

```
40 ? 2000
50 FOR W=0 TO 500:NEXT W
60 ? 3000
70 FOR W=0 TO 500:NEXT W
80 ? "4000 REM*
90 FOR W=0 TO 500:NEXT W
100 ? "CONT"
110 FOR W=0 TO 500:NEXT W
120 POSITION 2,0
130 FOR W=0 TO 500:NEXT W
140 POKE 842,13:STOP
150 FOR W=0 TO 500:NEXT W
160 POKE 842,12
170 FOR W=0 TO 500:NEXT W
1000 REM * DEZE REGEL WORDT GEWIST
2000 REM * DEZE REGEL WORDT GEWIST
3000 REM * DEZE REGEL WORDT GEWIST
4000 REM * DEZE REGEL WORDT GEWIST
```

programma ADR842.BC

10: De GRAPHICS-instructie wordt gegeven, om het beeldscherm te wissen. Uiteindelijk moet alleen van het beeldscherm worden gelezen, wat het programma daar naartoe schrijft. Het is belangrijk, ver genoeg naar onderen met het beschrijven van het beeldscherm te beginnen, zodat de systeem-melding 'STOPPED...' niet over het geschrevene heen komt te staan.

20: Hier wordt het getal 1000 op het beeldscherm weergegeven. Later, als van het beeldscherm wordt gelezen, herkent BASIC dit als regelnummer en omdat achter het regelnummer geen statement volgt, wordt regel 1000 in het geheugen gewist.

30: De wachtlus is alleen bedoeld om het verloop van het programma beter te kunnen volgen. Bij de uiteindelijke toepassing moet deze natuurlijk worden weggelaten.

40 t/m 70: Als 20 en 30.

80: Het is echter ook mogelijk om een volledige programma-regel te PRINTen, die door het lezen van het beeldscherm in het programma wordt opgenomen.

100: Vervolgens moet een 'CONT'worden geschreven, zodat het programma na het lezen verder loopt.

120: Nadat alles dat gelezen moet worden op het beeldscherm is geschreven, moet de onzichtbare cursor daar overheen op het beeldscherm worden gezet.

140: Nu wordt naar lezen omgePOKEd en het programma wordt gestopt.

160: Nadat het commando CONT van het beeldscherm is gelezen, loopt het programma verder en ICAX1 wordt weer teruggePOKEd naar beeldscherm-uitvoer.

Als u het programma nu eens een RUN geeft, kunt u op uw gemak toekijken, hoe de regels 1000, 2000, 3000 en 4000 worden geschreven. Dan springt de cursor naar kolom 2 in regel 0 en zet daar de mededeling 'STOPPED IN LINE 140' neer. Dan gaat de computer van het beeldscherm lezen en uiteindelijk volgt dan onderaan het scherm de mededeling 'READY', als het programma is geëindigd. En vraagt u nu eens om een LIST!

848-863 $350-$35F IOCB1

I/O-controleblok 1.

864-879 $360-$36F IOCB2

I/O-controleblok 2.

880-895 $370-$37F IOCB3

I/O-controleblok 3.

896-911 $380-$38F IOCB4

I/O-controleblok 4.

912-927 $390-$39F IOCB5

I/O-controleblok 5.

928-943 $3A9-$3AF IOCB6

I/O-controleblok 6.

De GRAPHICS-instructie opent kanaal 6 naar het beeldscherm ('S:'). Daarom kan kanaal 6 niet worden gebruikt als GRAPHICS 0 wordt verlaten, of er moet een CLOSE-instructie voor kanaal 6 worden gegeven. Daarna kunnen echter de instructies PLOT, DRAWTO en LOCATE niet meer worden gebruikt.
PRINT-instructies naar het grafische raam in de grafische modi 1, 2, 12, 13 en 0 met tekstraam moeten daarom ook naar kanaal 6 worden gestuurd.

944-959 $3B0-$3BF IOCB7

I/O-controleblok 7.

De instructie LPRINT gebruikt kanaal 7. Als dit kanaal naar een ander randapparaat is geopend en de instructie LPRINT wordt gegeven, dan volgt een foutmelding. Ook het LIST-commando gaat via kanaal 7, zelfs als dit reeds voor een ander doel is geopend. LIST sluit het kanaal na gebruik.

960-999 $3C0-$3E7 PRNBUF

Printer-buffer. Hier worden de data tijdelijk opgeslagen, voordat ze na EOL (end of line) naar de printer worden gestuurd.

1001 $3E9 STRTFLG

Deze aanduiding geeft een waarde ongelijk aan 0, als bij powerup de START-toets is ingedrukt.

1021-1151 $3FD-$4FD CASBUF

Cassette-buffer.

1152-1791 $480-$6FF user-RAM

Dit gebied staat ter beschikking voor korte machinetaal-routines. Het omvat 640 bytes. De floating point routine gebruikt echter wel de adressen 1406 t/m 1535 (= $57E-$5FF).

Werkelijk veilige geheugenruimte in dit lage gebied geeft alleen pagina 6 (1536-1791 = $600-$6FF).

1792-5377 $700-$1501 FMS

Het FMS (file management system) is een gedeelte van DOS. Het legt een verbinding tussen BASIC of DUP (zie hierna) en de diskette-eenheid.

5440- $1540- DUP

Het DUP (disc utility package) bestuurt verschillende DOS-functies zoals bijvoorbeeld 'Copy'.

De benodigde geheugenruimte varieert. Het loopt tot het door MEMLO aangewezen adres, maximaal t/m 13062 (= $3306).

Wanneer DUP-utilities uit het DOS-menu worden opgeroepen, kan het gebeuren dat over programma's heen wordt geschreven in het onderste gedeelte van de gebruikers-RAM.

Als op de diskette een MEM.SAV staat, dan worden de data uit dit gebied op de diskette gezet en na het verlaten van DOS daar weer neergezet. Dat is weliswaar heel prettig, maar MEM.SAVE gebruikt wel een aanzienlijk gedeelte van de diskette-sectoren en de MEM.SAV-functie verbruikt enige tijd. Het is daarom aan te bevelen om van een MEM-.SAV af te zien en in plaats daarvan diskette-bewerkingen met de XIO-instructies uit te voeren.

De volgende DOS-bewerkingen kunnen gemakkelijk worden vervangen:

Optie D: files wissen.
Optie E: filenamen veranderen.
Optie F: files beveiligen
Optie G: beveiliging verwijderen.
Optie I: diskette formatteren.

Optie A: Weergave van de directory kan ook redelijk eenvoudig worden vervangen. U heeft daarvoor slechts 2 regels nodig, die achter ieder ander BASIC-programma kunnen staan:

```
0 REM DIRCPRINT.BEC
1 REM ******************************
2 REM *                            *
3 REM * DISK-DIRECTORY AFDRUKKEN   *
4 REM *                            *
5 REM ******************************
32766 END
32767 CLR:DIM FLN$(18):CLOSE #1:OPEN #1,6,0,"D:*.*":FOR FLS=0 TO 64:INPUT #1,FLN
$:? FLN$:NEXT FLS
```

programma DIRCPRNT.BEC

32766: Scheidt het BASIC-programma, dat voor deze regel staat, af van het directory-programma.

32767: CLR wist de gedimensioneerde variabelen. Dit is nodig, omdat er dan meerdere keren naar deze regel gesprongen kan worden.

FLN$ wordt op 18 karakters gedimensioneerd en slaat de filenamen en het aantal gebruikte sectoren op. Dit zijn samen 18 karakters.

De CLOSE-instructie is een veiligheidsmaatregel voor het geval dat datakantaal 1 al is geopend.

Dan wordt kanaal 1 geopend. De 6 geeft de soort bewerking over het kanaal aan, namelijk het lezen van de directory. De '*' worden wild cards genoemd. Ze vervangen de gehele of gedeeltelijke filenaam. '*.*' vervangt iedere mogelijke filenaam, dus alle files worden gelezen.

De FOR-NEXT lus loopt alle files door. Er kunnen maximaal 64 files op een diskette worden gezet. Ook als er dan nog sectoren vrij zijn, kan er verder geen file worden opgeslagen, omdat de directory vol is. Als laatste wordt dan nog het aantal vrije sectoren gelezen.

Wordt in plaats van ?FLN$ een LPRINT FLN$ neergezet, dan gaat de uitvoer naar de printer.

Als de diskette met minder dan 58 files is gevuld, dan eindigt de uitvoer met de foutmelding ERROR- 136. Om dit te voorkomen kunt u eventueel nog een TRAP-instructie tussenvoegen.

Player-Missile-Graphics

53248-53505 $D000-$D0FF GITA

53248 $D000 HPOSPO

(W) Hier wordt de horizontale positie van player 0 opgeslagen.

Player en missile zijn grafische objecten, die in vorm, kleur en beweging onafhankelijk zijn van de normale (playfield) graphics. Gewoonlijk worden zulke objecten sprites genoemd. ATARI's players bedekken als smalle band in vertikale richting het beeldscherm in de gehele hoogte. De horizontale positie van deze player-band wordt opgeslagen in HPOSPn, waarbij n een getal is van 0 t/m 3.

De positie van het PM-object heeft betrekking op de fysieke indeling van de beeldbuis. Daarom zijn horizontale waarden mogelijk van 0 t/m 227. Een player komt echter pas links in beeld, als hij ongeveer de horizontale positie 45 heeft en verlaat het beeldscherm rechts al bij een waarde van circa 210. De exacte waarden kunnen van apparaat tot apparaat iets verschillen.

De vertikale positie van een player wordt bepaald door de toestand van zijn data. Ieder PM-object heeft een eigen afgescheiden geheugengebied. Hoe het totale PM-geheugengebied is ingedeeld, wordt verklaard bij PMBASE (53279 = $D407). Een player is altijd een byte breed. Iedere byte kan echter twee, vier of acht beeldpunten weergeven. Dit is afhankelijk van de definitie in het bijbehorende register SIZEO (53256 = $D008).

Dat geeft dus verschillende display-breedtes van de player. Omdat de player de totale hoogte van het beeldscherm gebruikt, omvat hij 128 bytes, als iedere byte twee beeldlijnen weergeeft (resolutie van twee lijnen) of 256 bytes (bij een resolutie van een lijn). Moet een player niet de gehele hoogte van het beeldscherm opvullen, dan worden de desbetreffende bytes op 0 gezet. Hoe van een byte een hoeveelheid beeldpunten wordt gemaakt, kunt u lezen in het hoofdstuk 'Karakterset'.

Dit register en veel van de registers hebben diverse functies voor lezen (R) en schrijven (W). Nadat hier de horizontale positie van player 0 is opgeslagen, kan niet onmiddellijk met PEEK(53248) deze positie weer worden gelezen, want (R) geeft aan, of een 'collision' (overlapping) heeft plaatsgevonden tussen missile 0 en een speelveld.

Met welk speelveld een botsing mogelijk is, wordt aangegeven door de onderste nibble. Met de verschillende speelvelden worden de verschillende COLOR-waarden bedoeld.

bit	7	6	5	4	3	2	1	0
decimale waarde					8	4	2	1
speelveld	niet gebruikt				3	2	1	0

Na een botsing van missile 0 met speelveld (kleur) 2 vindt PEEK(53248) een 4.

53249-53251 $D001-$D003 HPOSP1-3

(W) Horizontale positie van player 1 t/m 3.

(R) Missile 1 t/m 3 / speelveld – collision.

53252 $D004 HPOSMO

(W) Horizontale positie van missile 0. Missiles bewegen zich horizontaal op dezelfde manier als players.

(R) Collision tussen player 0 en het speelveld.

bit	7	6	5	4	3	2	1	0
decimale waarde					8	4	2	1
speelveld	niet gebruikt				3	2	1	0

Na een botsing van player 0 met speelveld (kleur) 3 vindt PEEK (53252) een 8.

53253-53255 $D005-$D007 HPOSM1-3

(W) Horizontale positie van missile 1 t/m 3.

(R) Player 1 t/m 3 / speelveld - collision.

53256 $D008 SIZEPO

(W) Bepaalt de breedte van player 0.

Een player is altijd een byte (acht bits) breed. Ieder afzonderlijk bit produceert een pixel, als het aan is. SIZEPO bepaalt, hoeveel beeldpunten een pixel breed is. Een 0 of een 2 in dit register geeft de normale breedte van twee beeldpunten per pixel. Een 1 maakt de pixels vier beeldpunten breed, de player is dan 32 beeldpunten breed. Een 3 maakt een pixel 8, een byte 64 beeldpunten breed.

De hoogte van de pixels wordt in SDMCTL (559 = $22F) in bit 4 vastgesteld. Normaal, bit 4 uit, is iedere player-pixel twee beeldlijnen hoog. Door het aan zetten van bit 4 op adres 559 wordt de hoogte van de pixel een beeldlijn.

bitpatr. normale breedte dubbele breedte

10000001 1100000000000011 11110000000000000000000000001111
01000010 0011000000001100 00001111000000000000000011110000
00100100 0000110000110000 00000000111100000000111100000000
00011000 0000001111000000 00000000000011111111000000000000
00001000 0000000011000000 00000000000000001111000000000000
00111000 0000111111000000 00000000111111111111000000000000
11101111 1111110011111111 11111111111110000111111111111111

Weergave van een player bij bit-patroon met achtereenvolgend de decimale waarden 129, 66, 36, 24, 8, 56 en 239. Bij een resolutie van een beeldlijn stelt iedere 1 een beeldpunt voor. Bij een resolutie van twee beeldlijnen wordt het puntenpatroon, dat een byte produceert, op twee beeldlijnen weergegeven. De pixels worden daardoor twee maal zo hoog.

(R) Gelezen wijst dit register botsingen van missile 0 met een player aan.

bit	7	6	5	4	3	2	1	0
decimale waarde					8	4	2	1
player	niet gebruikt				3	2	1	0

Als missile 0 player 0 vernietigt, dan vindt PEEK(53256) een 0.

53257-53259 $D009-$D00B SIZEP1-3

(W) Breedte van player 1 t/m 3.

(R) Missile 1 t/m 3 / player – collisions.

53260 $D00C SIZEM

(W) Per twee bits in dit register wordt de breedte van een missile bepaald.

bit	7	6	5	4	3	2	1	0
gewicht	128	64	32	16	8	4	2	1
missile					-3-	-2-	-1-	-0-

En de volgende bit-paren bepalen de breedte van de missile:

00 normale breedte, twee beeldpunten per pixel.
01 dubbele breedte, vier beeldpunten per pixel.
10 normale breedte, twee beeldpunten per pixel.
11 viervoudige breedte, acht beeldpunten per pixel.

Als u dus missile 3 in viervoudige breedte wilt weergeven, dan moet u bit 7 en 6 aan zetten. Bit 4 zorgt ervoor dat missile 2 de dubbele breedte heeft. Als missile 1 en 0 de normale breedte moeten houden, moeten de bits 3 t/m 0 uit zijn. Zo ontstaat het bit-patroon 11 01 00 00 met de waarden 128+64+16+0+0+0+0 = 208. U moet dus bij SIZEM de waarde 208 POKEn, om de hiervoor gegeven breedtes van de verschillende missiles te krijgen.

(R) Met PEEK kunt u hier controleren, of er een botsing heeft plaatsgevonden tussen player 0 en een andere player.

bit	7	6	5	4	3	2	1	0
decimale waarde					8	4	2	1
player niet gebruikt					3	2	1	-

53261 $D00D GRAFPO

(W) Hier kan voor player 0 een grafische vorm worden gekozen. Dat werkt echter alleen als de DMA in GRACTL (53277 = $D01D) voor de player is uitgeschakeld, omdat anders dit grafische register door de directe geheugentoegang (DMA) op de voor de player in het PM-geheugengebied opgeslagen data worden geladen.

Wordt de DMA uitgeschakeld, dan kan in BASIC hier alleen een byte worden opgeslagen, waarvan het bit-patroon de vorm van de player in de totale hoogte bepaalt, dus er ontstaan vertikale lijnen. Wordt in GRAFPO bijvoorbeeld 255 gezet, dus alle acht bits aan, dan verschijnt de player in acht bit brede balken, dus bij normale breedte 16 beeldpunten. Door de beperking zijn de toepassingen van de GRAFPn-registers beperkt.

(R) Player 1 / player – collisions.

53262,53263 $D00E,$D00F GRAFP1, 2

(W) Vorm van player 1 en 2

(R) Player 2 / player – collisions.

53264 $D010 GRAFP3

(W) Vorm van player 3.

(R) Aanduiding voor vuurknop van stuurknuppel 0. Wordt in BASIC via het parallel-register STRIGO (644 = $284) beschreven.

De vuurknop van stuurknuppel 0 is aangesloten op pen 6 van de stuurknuppel-bus 1. Als de vuurknop niet is ingedrukt, staat in bit 0 een 1, die door het vuren op 0 wordt gezet. Als bit 2 van GRACTL (53277 = $D01D) aan wordt gezet, lezen alle TRIG-registers een 0 totdat GRACTL weer wordt teruggezet. Daarmee is het continue vuren van alle vuurknoppen instelbaar.

53265 $D011 GRAFM

(W) Vorm voor alle missiles. Werkt als GRAFPn.

Ieder bit-paar heeft betrekking op een missile:

bit	7	6	5	4	3	2	1	0
missile			-3--	--2--		--1--		--0--

Ieder bit dat aan is, produceert een vertikale lijn van een pixel breed over de totale hoogte van het beeldscherm.
Natuurlijk kan voor iedere missile apart de gedaante worden gedefinieerd, als bepaald wordt welke bits aan moeten zijn en de decimale waarde van dit bit-patroon in GRAFM wordt opgeslagen.

(R) Vuurknop van stuurknuppel 1. Het parallel-register hiervan is STRIG1 (645 = $285).

53266 $D012 COLPMO

(W) Kleurwaarde van player en missile 0. Missiles hebben altijd dezelfde kleurwaarde als de bijbehorende player. De kleurwaarde voor player/missile 0 wordt in BASIC via het parallel-register PCOLRO (704 = $2C0) geprogrammeerd.

Als de vier missiles worden samengevoegd tot een vijfde player (door het aan zetten van bit 4 in GPRIOR 623 = $26F), dan krijgt deze vijfde player zijn kleur van register COLOR3 (711 = $2C7), het kleurregister voor speelveld 3 (dat in de meeste grafische modi niet wordt gebruikt).

(R) Vuurknop van stuurknuppel 2. Het parallel-register hiervan is STRIG2 (646 = $286).

53267 $D013 COLPMI

(W) Kleurwaarde van player/missile 1. Het parallel-register hiervan is PCOLR1 (705 = $2C1).

(R) Vuurknop van stuurknuppel 3. Het parallel-register hiervan is STRIG3 (647 = $287).

53268 $D014 COLPM2

(W) Kleurwaarde van player/missile 2. Het parallel-register hiervan is PCOLR2 (706 = $2C2).

(R) Wordt gebruikt om vast te leggen of de computer PAL- (bits 1 t/m 3 op 0 gezet) of NTSC- (bits 1 t/m 3 op 1 gezet = decimaal 14) compatibel is.

NTSC is de Noord-Amerika geldende televisienorm, terwijl PAL in de meeste Europese landen en in Israël als norm wordt gebruikt. Een derde standaard, het SECAM-systeem, wordt gebruikt in Frankrijk, de DDR en de USSR.

PAL is gekoppeld aan de 50 hertz frequentie. Een PAL VBLANK volgt dus iedere 1/50 seconde. Dit is 12% langzamer dan bij NTSC met 1/60 seconde. Met 2,217 MHz loopt de 50 Hz versie.

Het PAL-systeem werkt bovendien met 312 in plaats van 262 beeldlijnen (scan lines). Dit verschil wordt gelijk getrokken, doordat de display-list van de PAL-versie met 24 lege regels begint. Met een PAL-ATARI kan men dus eigenlijk een groter gedeelte van het beeldscherm gebruiken. Hoe dat gaat, kunt u nalezen in het hoofdstuk 'Display-list.' De kleurwerking van de PAL-versie is hardwarematig aangepast.

53269 $D015 COLPM3

Kleurwaarde voor player/missile 3. Het parallel-register hiervan is PCOLR3 (707 = $2C3).

53270-53273 $D016-$D019 COLPF0-3

Kleurwaarde van speelveld 1 t/m 3. De parallel-registers hiervan zijn COLOR0 t/m 3 (708-711 = $2C4-$2C7).

53274 $D01A COLBK

Kleurwaarde van de achtergrond (BAK). Het parallel-register hiervan is COLOR4 (712 = $2C8).

53275 $D01B PRIOR

(W) Hier wordt vastgelegd welke grafische elementen (players, missiles en speelvelden) bij het snijden andere overlappen. Kan in BASIC alleen worden beschreven via het parallel-register GPRIOR (623 = $26F), omdat een BASIC-POKE naar PRIOR bij de volgende machinecyclus wordt overschreven, terwijl de in het schaduwregister opgeslagen waarde voortdurend in PRIOR wordt vernieuwd.

Het aan zetten van de diverse bits geeft de volgende prioriteiten (PL-Player, PF-speelveld, BAK=achtergrond):

Bit 3	Bit 2	Bit 1	Bit 0
PF 0	PF 0	P 0	P 0
PF 1	PF 1	P 1	P 1
P 0	PF 2	PF 0	P 2
P 0	PF 3/P 5	PF 1	P 3
P 0	P 0	PF 2	PF 0
P 0	P 1	PF3/P 5	PF 1
PF 2	P 2	P 2	PF 2
PF 3/P 5	P 3	P 3	PF 3/P 5
BAK	BAK	BAK	BAK

Worden in de bit 0 t/m 3 tegenstrijdige prioriteiten ingesteld, dan worden de grafische objecten in het overlappingsgebied zwart.

Bit 4 aan voegt de vier missiles samen tot een vijfde player.
Bit 5 aan past de OR-functie toe op de overlappende kleurwaarden.
Bits 6 en 7 bepalen de GTIA-modi 9 t/m 11. (Verdere informatie bij GPRIOR (623 = $26F).

53267 $D01C VDELAY

(W) Maakt het mogelijk de players en missiles een beeldlijn te verplaatsen als de ingeschakelde resolutie twee beeldlijnen is. Als een bepaald bit aan is, beweegt het bijbehorende object een beeldlijn naar onderen. Als DMA is geactiveerd, ontstaat een verplaatsing van meer dan een beeldlijn, doordat het bit-patroon op een andere plaats in het PM-geheugengebied wordt gezet (zie voorbeeld-programma PMGRPLAY.BEC).

bit 7	(=128)	player 3
bit 6	(= 64)	player 2
bit 5	(= 32)	player 1
bit 4	(= 16)	player 0
bit 3	(= 8)	missile 3
bit 2	(= 4)	missile 2

bit 1 (= 2) missile 1
bit 0 (= 1) missile 0

53277 $D01D GRACTL

(W) De bits 0 t/m 2 kunnen ingesteld worden.

Wordt bit 2 aangezet, dan worden alle vuurknoppen van stuurknuppels en paddles omgeschakeld op doorlopend vuren, als een enkele vuurknop wordt ingedrukt. Het is echter niet mogelijk om een enkele vuurknop op voortdurend vuren te zetten. Het vuren houdt pas op als bit 2 hier weer op 0 wordt teruggezet.

Met bit 1 worden de players en met bit 0 de missiles van PM-graphics geactiveerd.

53278 $D01E HITCLR

(W) Een willekeurige waarde die in dit register wordt gezet, wist alle PM-collision registers. Omdat een collision-register pas weer een nieuwe collision kan registreren als het tussendoor wordt gewist, moet HITCLR regelmatig worden beschreven.

53279 $D01F CONSOL

Aanduiding voor de functietoetsen OPTION, SELECT en START.

Wordt geen van deze toetsen ingedrukt, dan zijn de bits 0 t/m 2 aan, dus in CONSOL staat dan de waarde 7. Het indrukken van een toets zet het desbetreffende bit terug op 0.

toets		bit toestand van de bits							
OPTION	2	0	0	0	0	1	1	1	1
SELECT	1	0	0	1	1	0	0	1	1
START	0	0	1	0	1	0	1	0	1
decimale waarde		0	1	2	3	4	5	6	7

Hoe PM-graphics nu eigenlijk geprogrammeerd moet worden, kunt u zien in de beide volgende voorbeeld-programma's (ook al worden enkele registers pas verderop verklaard):

```
0 REM PMGRPLAY.BEC
1 REM ********************************
2 REM *                              *
3 REM *   ROMANTISCHE ZONSOPGANG     *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 23
20 POKE 708,226:POKE 709,208:POKE 710,6:POKE 712,128
30 COLOR 1:FOR X=0 TO 5:PLOT X,0:DRAWTO X,95:NEXT X
40 FOR X=152 TO 159:PLOT X,0:DRAWTO X,95:NEXT X
50 FOR Y=0 TO 69:READ R:PLOT 6,Y:DRAWTO R,Y:NEXT Y
60 FOR Y=0 TO 53:READ L:PLOT L,Y:DRAWTO 151,Y:NEXT Y
70 DATA 37,38,39,40,40,41,41,41,40,40,38,36,35,34,33,33,32,33,35,36,38,37,37,36,
35,34,28,25,24,23,22,23,24,24
80 DATA 26,27,26,24,22,20,17,18,19,20,20,18,16,14,11,8,7,8,7,9,8,10,13,15,15,14,
14,10,9,11,10,10,9,7,8,7
90 DATA 148,146,144,141,138,139,143,147,148,143,141,147,145,143,140,137,134,131,
128,129,130,134,138,143,141
100 DATA 139,137,138,140,136,132,128,123,124,125,128,131,135,139,143,149,147,145
,142,139,135,131,126,121,119
110 DATA 120,122,125,131
120 COLOR 3:Z=Z+1
130 X=INT(RND(0)*80)+40:Y=INT(RND(0)*70):PLOT X,Y
140 IF Z<30 THEN 120
150 COLOR 2:FOR Y=0 TO 7:PLOT 6,81+Y:DRAWTO 52,84+Y:DRAWTO 80,80+Y:DRAWTO 135,86
+Y:DRAWTO 150,82+Y:NEXT Y
160 FOR Y=86 TO 95:PLOT 6,Y:DRAWTO 150,Y:NEXT Y
200 POKE 704,34
210 POKE 623,8
220 POKE 559,42
230 P=PEEK(106)-20
240 POKE 54279,P
250 PMBASE=P*256
260 POKE 53256,3
270 POKE 53277,2
280 X=30:Y=91
300 POKE 53248,X
310 FOR J=PMBASE+512 TO PMBASE+639:POKE J,0:NEXT J
320 FOR J=PMBASE+512+Y TO PMBASE+512+31+Y:READ D:POKE J,D:NEXT J
330 DATA 24,60,60,60,126,126,126,126,126,255,255,255,255,255,255,255
340 DATA 255,255,255,255,255,255,255,126,126,126,126,126,60,60,60,24
400 S=STICK(0)
410 IF S=15 THEN 400
420 IF S=7 THEN X=X+1
430 IF S=11 THEN X=X-1
440 POKE 53248,X
450 IF S=13 THEN GOSUB 500
460 IF S=14 THEN GOSUB 600
470 GOTO 400
500 IF Y>92 THEN RETURN
510 FOR J=32 TO 0 STEP -1
520 POKE PMBASE+512+Y+J,PEEK(PMBASE+511+Y+J)
530 NEXT J
540 C=C-1:IF C<0 THEN C=0
550 CO=INT(C/15)*2
560 POKE 704,34+CO:POKE 712,128+CO/2:POKE 709,208+CO
570 Y=Y+1
580 RETURN
600 IF Y<16 THEN RETURN
610 FOR J=0 TO 32
620 POKE PMBASE+511+Y+J,PEEK(PMBASE+512+Y+J)
630 NEXT J
640 C=C+1:IF C>75 THEN C=75
650 CO=INT(C/15)*2
660 POKE 704,34+CO:POKE 712,128+CO/2:POKE 709,208+CO
670 Y=Y-1
680 RETURN
```

programma PMGRPLAY.BEC

10: De GRAPHICS-instructie heeft alleen betrekking op het 'speelveld'. De PM-objecten zijn daarvan volledig onafhankelijk.

20: Voor de grafische punten ('speelvelden') en de achtergrond worden kleurwaarden in de desbetreffende kleurregisters neergezet.

30 t/m 110: tekenen met COLOR 3 links het silhouet van een loofboom en aan de rechter beeldschermrand een spar.

120 t/m 140: PLOTten willekeurig dertig sterren aan de avondhemel.

150 en 160: toveren een schilderachtige bergkam op het tv-scherm.

200: Nu gaan we aan de slag met PM-graphics. Voor player 0 wordt in het bijbehorende kleurregister de kleurwaarde neergezet.

210: De prioriteit wordt zo vastgelegd, dat bomen en bergen voor player 0 liggen. De inktzwarte hemel dient als achtergrond en ligt achter player 0.

220: Normale breedte voor het speelveld (bit 1 = 2), de players (bit 3 = 8) en de directe geheugentoegang DMA (bit 5 = 32) worden ingeschakeld.

230: Bij RAMTOP wordt gecontroleerd waar de geheugengrens ligt. GRAPHICS 23 gebruikt met zijn display bijna 16 pagina's. Omdat de PM-graphics met een resolutie van twee lijnen moet werken, zijn 4 pagina's daarvoor genoeg. Het basis-adres voor de PM-graphics moet dus 20 pagina's onder RAMTOP liggen. Deze waarde wordt hier in P vastgelegd.

Als u hier wilt proberen om minder pagina's te gebruiken en bijvoorbeeld 8 aftrekt van RAMTOP, dan kunt u later op het beeldscherm zien dat de player zich in het gebied van het beeldscherm-geheugen bevindt en hoe deze zich daarin beweegt.

240: Registers 54279 is PMBASE, het begin-adres van het PM-geheugengebied. Omdat PMBASE net als RAMTOP alleen de HI-byte aangeeft, krijgt men het uiteindelijk

250: startadres door deze waarde te vermenigvuldigen met 256.

260: player 0 krijgt de viervoudige breedte. Elk van de acht bits wordt acht beeldpunten breed weergegeven. De player beslaat in totaal 64 beeldpunten op het scherm.

270: schakelt player in.

280: De begin-coördinaten voor de player.

300: In HPOSPO wordt X=30 als horizontale positie voor player 0 neergezet. Het is een plaats aan de uiterste (zichtbare) rand van het tv-scherm. Wordt de player nog verder naar links gezet, dan verdwijnt hij uit het beeld, maar het programma wordt pas onderbroken door ERROR- als de waarde voor X kleiner is dan 0 of aan de rechter kant groter dan 255.

310: De geheugenplaatsen 412 t/m 639 achter PMBASE zijn bij een resolutie van twee lijnen voor player 0 gereserveerd. Hier wordt dit geheugengebied gewist, hetgeen niet absoluut noodzakelijk is. Een onderbreking van het programma met BREAK wist het PM-geheugen echter niet, zodat bij een nieuwe programma-start in bepaalde omstandigheden ineens twee players op het beeldscherm verschijnen. De ene, onbeweeglijke, bevindt zich op de vertikale positie, waar hij ook was toen het programma onderbroken werd. De tweede player bevindt zich daar, waar hij door het opnieuw doorlopen programma is neergezet, dus bij zijn start-positie.

Als u dit programma laat lopen, dan met BREAK onderbreekt en vervolgens een nieuwe RUN geeft, kunt u zien hoe de player eerst op de oude vertikale positie verschijnt, dan door deze regel wordt gewist en tenslotte op zijn uitgangs-positie weer wordt opgebouwd.

Het is dus beter om eerst het PM-geheugengebied te wissen en dan pas in register 53277 de player in te schakelen.

320: Hier worden de data-bytes, die de player weergeven, in het geheugengebied voor player 0 ingelezen. Een player is altijd een byte breed en bij een resolutie van twee lijnen 128 bytes hoog. Voor deze 128 bytes staan de geheugenplaatsen PMBASE+512 t/m PMBASE+639 ter beschikking. De eerste byte wordt met de waarde van de vertikale positie Y=91 adressen lager dan PMBASE+512 neergezet en de volgende 31 bytes daarna.

330 en 340: bevatten de DATA, die de player weergeven. Iedere decimale waarde representeert een bit-patroon. Ieder bit dat aan is, produceert een pixel, waarvan de breedte op adres 53256 ($D008) is ingesteld (hier viervoudig = acht beeldpunten) en de hoogte twee beeldlijnen bedraagt bij een resolutie van een lijn. Dit is vastgelegd in register 559 door het aan of uit zetten van bit 4.

400: Hier begint de uitlezing van stuurknuppel 0. Het is zinvol om de in STICK(0) (632 = $278) gevonden waarde in een variabele op te slaan, om de lange uitdrukking STICK(0) niet steeds te hoeven gebruiken. De BASIC-instructie STICK(0) doet overigens niets anders dan het statement PEEK(632), maar STICK(0) gebruikt minder geheugenruimte.

410: Wanneer de stuurknuppel in rust is (STICK(0) = 15), wordt hij onmiddellijk opnieuw uitgelezen.

420: De stuurknuppel wordt naar rechts gedrukt, de player moet zich naar rechts bewegen, dus zijn horizontale positie moet een grotere waarde aannemen.

430: Hetzelfde geldt voor de beweging naar links.

440: De horizontale positie van de player wordt gewoon in het bijbehorende register geschreven. Dat gaat eenvoudig en snel. Players bewegen zich horizontaal heel vlot.

450: Met vertikale bewegingen ligt dat heel anders. Een player beweegt zich in vertikale richting, wanneer zijn data in zijn geheugengebied worden verplaatst. Moet de player dus naar onderen bewegen, dan moeten al zijn data-bytes een geheugenplaats naar onderen worden verschoven. Hoe meer data de player omvat, des te langer deze handeling duurt. U kunt op het beeldscherm volgen hoe de player zich lijn voor lijn naar onderen beweegt. De routine daarvoor staat in het hulpprogramma vanaf regel 500.

460: De beweging naar boven verloopt op dezelfde manier.

470: Nadat alle bewegingsrichtingen zijn uitgelezen, springt het programma terug naar de volgende uitlezing van de stuurknuppel.

500: Hier begint de routine voor de beweging naar onderen. Wanneer de Y-positie 92 is bereikt, mag er geen beweging naar onderen meer plaatsvinden. Zo'n controle is beslist noodzakelijk, omdat de data door de verplaatsings-routine anders onbelemmerd in geheugengebieden gaan rondzwerven, waar ze niets te maken hebben (bijvoorbeeld in het geheugengebied van een andere player).

510: Omdat de player hier 32 bytes hoog is, moeten 33 (0 t/m 32) bytes worden verplaatst. Bij de beweging naar onderen, wordt begonnen met de laagste byte en zet deze een adres lager neer. Dan wordt de byte genomen die een plaats hoger ligt en verplaatst deze naar onderen, enzovoorts. Als u goed oplet, kunt u zien hoe de player zich op een lijn telkens een ogenblik uitrekt, dus quasi naar onderen wiebelt.

Nadat alle 32 bytes een plaats naar onderen zijn verschoven, moet de byte die een plaats hoger ligt, nog naar onderen worden verschoven. Deze byte bevat een 0, omdat de player daar eindigt. Zo wordt de laatste byte, nadat hij naar onderen is verschoven, op zijn oude plaats uitgewist.

540 t/m 560: Als in dit romantische programma de player 0 zon langzaam vanachter de COLOR 2 bergen opkomt en zijn roodgloeiende licht door de COLOR 1 bomen laat schijnen, dan moet het natuurlijk ook lichter worden. De zon verandert langzaam van oranje in wit, de hemel krijgt een stralend blauwe kleur en overstraalt de RANDOM-sterren, de weiden krijgen een frisse meigroene kleur. Dan is zelfs oma enthousiast over de computer.

In deze regels wordt de kleurwaarde veranderd, wanneer de player (zon) naar onderen wordt bewogen, dus bij een zonsondergang. De kleuren worden donkerder, de helderheidswaarde wordt kleiner.

570: Nu moet de Y-waarde nog worden vernieuwd en dan

580: RETURN naar het hoofdprogramma.

600 t/m 680: zijn op dezelfde manier opgebouwd voor de beweging naar boven.

Hoe met de collision-registers gewerkt moet worden, toont de player/missile-lichtshow:

```
730 IF S=11 THEN X1=X1-1
740 POKE 53249,X1
750 IF S=13 THEN GOSUB 800
760 IF S=14 THEN GOSUB 900
770 IF PEEK(53261)=1 THEN GOSUB 1000
800 IF Y1>92 THEN RETURN
810 FOR J=32 TO 0 STEP -1
820 POKE PMBASE+640+Y1,PEEK(PMBASE+639+Y1+J)
830 NEXT J
840 Y1=Y1+1:RETURN
900 IF Y1<16 THEN RETURN
910 FOR J=0 TO 32
920 POKE PMBASE+639+Y1+J,PEEK(PMBASE+640+Y1+J)
930 NEXT J
940 Y1=Y1-1:RETURN
1000 SOUND 0,3,6,8
1010 FOR Q=0 TO 6:POKE 712,Q*2:NEXT Q
1020 POKE 53278,255
1030 POKE 712,0
1040 SOUND 0,0,0,0
1050 RETURN

0 REM PMGRCOLL.BEI
1 REM ********************************
2 REM *                              *
3 REM * PLAYER/MISSILE LIGHT-SHOW    *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 19
200 POKE 704,50:POKE 705,152
210 POKE 623,8+32
220 POKE 559,42
230 P=PEEK(106)-8
240 POKE 54279,P
250 PMBASE=P*256
260 POKE 53256,3:POKE 53257,3
270 POKE 53277,2
280 X0=30:Y0=92:X1=190:Y1=16
300 POKE 53248,X0:POKE 53249,X1
310 FOR J=PMBASE+512 TO PMBASE+767:POKE J,0:NEXT J
320 FOR J=PMBASE+512+Y0 TO PMBASE+512+31+Y0:READ D:POKE J,D:NEXT J:RESTORE
330 FOR J=PMBASE+640+Y1 TO PMBASE+640+31+Y1:READ D:POKE J,D:NEXT J
340 DATA 24,60,60,60,126,126,126,126,255,255,255,255,255,255,255,255
350 DATA 255,255,255,255,255,255,255,126,126,126,126,126,60,60,60,24
400 S=STICK(0)
410 IF S=15 THEN 700
420 IF S=7 THEN X0=X0+1
430 IF S=11 THEN X0=X0-1
440 POKE 53248,X0
450 IF S=13 THEN GOSUB 500
460 IF S=14 THEN GOSUB 600
470 IF PEEK(53260)=2 THEN GOSUB 1000
480 GOTO 700
500 IF Y0>92 THEN RETURN
510 FOR J=32 TO 0 STEP -1
520 POKE PMBASE+512+Y0+J,PEEK(PMBASE+511+Y0+J)
530 NEXT J
540 Y0=Y0+1:RETURN
600 IF Y0<16 THEN RETURN
610 FOR J=0 TO 32
620 POKE PMBASE+511+Y0+J,PEEK(PMBASE+512+Y0+J)
630 NEXT J
640 Y0=Y0-1:RETURN
700 S=STICK(1)
710 IF S=15 THEN 400
720 IF S=7  THEN X1=X1+1
```

10: Hier wordt GRAPHICS 3 (+16) ingeschakeld, omdat deze modus heel weinig geheugen gebruikt. Verder is de modus willekeurig, omdat alleen de achtergrondkleur (0= zwart) wordt gebruikt.

200: Kleurwaarde voor player 0 en player 1.

210: Naast de prioriteit voor players en speelveld wordt deze keer ook de derde, geORde kleur bij de overlapping geactiveerd (+32). Natuurlijk kunt u ook meteen de waarde 40 POKEn.

220: DMA net als in het vorige programma.

230: Vanwege het geringe geheugengebied van GRAPHICS 3 (respectievelijk 19), hoeft de PM-basis hier slechts acht pagina's onder RAMTOP te liggen.

260: Beide players krijgen hier de viervoudige breedte.

280: Player 0 begint links onder, player 1 rechts boven.

310: Deze keer wordt het geheugengebied van player 0 en 1 gewist, dan worden in regel

320: de data voor player 0 ingelezen en weggezet en in regel

330: de data voor player 1. Omdat ze allebei dezelfde gedaante moeten hebben, worden voor player 1 na RESTORE dezelfde

340 en 350: DATA gebruikt als voor player 0.

400 t/m 460: lezen STICK(0) uit en laten het programma dan verder gaan in regel 700, waar STICK(1) aan de beurt is.

470: Als PEEK(53260)=2, dan is player 0 bij player 1 terecht gekomen en springt het programma naar de subroutine in regel 1000.

500 t/m 640: regelen de vertikale verplaatsing van player 0.

700 t/m 760: lezen STICK(1) uit.

770: Als op adres 53261 een 1 wordt gelezen, dan is player 1 bij player 0 aangeland.

800 t/m 940: verzorgen de vertikale verplaatsing van player 1.

1000: Als u de beide players naar elkaar toe laat gaan, dan breekt hier de hel los.

Maar dat moet u zelf maar eens bekijken.

1010: In geen geval mogen we echter vergeten het collision-register weer te wissen, zodat het er heftig toe kan gaan.

De beide cirkelronde players zien eruit als een schijnwerper. Player 0 straalt een tamelijk helderrode kleur uit en player 1 bijna cyaan (DIN blauw). Als beiden over elkaar liggen, krijgt het snijvlak een groengele kleur. Het klopt bijna met de fysische resultaten (rood en groen schijnwerperlicht geeft gemengd een gele kleur!).

Hier is nog eens de berekening van de overlappingsvlakken:

rood	(kleur 2, helderheid 2) 00110010 (= 3*16+ 2 = 50)
cyaan	(kleur 9, helderheid 8) 10011000 (= 9*16+ 8 = 152)
meigroen	(kleur 11, helderheid 10) 10111010 (= 11*16+10 = 186)

Het is overigens heel normaal, dat op de plaats van de player(s) knipperende strepen blijven staan, wanneer u een PM-programma met BREAK onderbreekt. Meestal helpt een RESET wel, om dit storende effect uit te schakelen.

SOUND

53760 $D200 AUDF1

De ATARI is uitgerust met vier toongeneratoren, die bestuurd worden door POKEY (POtentiometer en KEYboard chip). Bij iedere toongenerator horen een register voor de frequentie en een besturingsregister voor het volume en de vervorming. De vervorming wordt door de polycounter en de schuifregisters van POKEY geregeld.

(W) AUDF1 slaat de frequentie op voor geluidskanaal 1. Er kunnen waarden in worden gezet van 0 t/m 255. POKEY telt daar een 1 bij op, zodat uiteindelijk een waarde van 1 t/m 256 ontstaat. Deze waarde bepaalt het aantal pulsen, dat binnen moet komen, voordat een puls wordt afgegeven. Hoe hoger de hier opgeslagen waarde, des te minder pulsen worden afgegeven. Een lage frequentie nemen we waar als een lage toon.

Als u meer van muziek weet dan aan welke kant u op een blokfluit moet blazen, kunt u de waarde, die hier neergezet moet worden om een bepaalde toon te produceren, exact berekenen =

INT(31960/(f+2).

waarbij f de frequentie van de gewenste toon in hertz (Hz) voorstelt.
(R) Waarde van paddle (draairegelaar) 0. Het parallel-register is PADDLO (624 = $270).

53761 $D201 AUDC1

(W) De hoogste drie bits worden gebruikt, om het soort vervorming (van het geluid) op te slaan:

bit 7 6 5	vervormingswaarde (BASIC)	modulatieproces (POKEY)
0 0 0	0	5 bits, dan 17 bits polycounter
0 0 1	2	alleen 5 bits polycounter
0 1 0	4	5 bits, dan 4 bits polycounter
0 1 1	6	alleen 5 bits polycounter
1 0 0	8	alleen 17 bits polycounter
1 0 1	10	geen vervorming
1 1 0	12	alleen 4 bits polycounter
1 1 1	14	geen vervorming

In de bits 0 t/m 3 wordt het volume opgeslagen. Waarden van 0 (0000) t/m 15 (1111) zijn mogelijk.

De BASIC-instructie SOUND k,f,d,v opent het geluidskanaal k, waarbij k een waarde is van 0 t/m 3. De waarde f bepaalt de frequentie in register AUDFn, d is de vervorming (even waarden van 0 t/m 14) en v is het volume van 0 t/m 15 in register AUDCn.

(R) Draairegelaar (paddle) 1.

53762 $D202 AUDF2

(W) Frequentie voor geluidskanaal 2.

(R) Draairegelaar 2.

53763 $D203 AUDC2

(W) Besturingsregister van geluidskanaal 2.

(R) Draairegelaar 3.

53764 $D204 AUDF3

(W) Frequentie voor geluidskanaal 3.

(R) Draairegelaar 4.

53765 $D205 AUDC3

(W) Besturingsregister van geluidskanaal 3.

(R) Draairegelaar 5.

53766 $D206 AUDF4

(W) Frequentie voor geluidskanaal 4.

(R) Draairegelaar 6.

53767 $D207 AUDC4

(W) Besturingsregister van geluidskanaal 4.

(R) Draairegelaar 7.

53768 $D208 AUDCTL

(W) Geluidsbesturing. AUDCTL is een keuze-register, dat alle geluidskanalen beïnvloedt. Door het aan zetten van bepaalde bits kunnen de volgende werkingen teweeg worden gebracht:

bit Werking

7 maskeert de 17 bits polycounter tot 9 bits
6 zet op kanaal 1 een klokfrequentie van 2,217 MHz
5 zet op kanaal 3 een klokfrequentie van 2,217 MHz
4 verbindt de kanalen 1 en 2 (16 bits)
3 verbindt de kanalen 3 en 4 (16 bits) .
2 hoge tonen filter in kanaal 1, gestuurd door kanaal 2
1 hoge tonen filter in kanaal 2, gestuurd door kanaal 4
0 schakelt de hoofd-frequentie om van 64 kHz naar 15 kHz

(R) Leest alle acht draairegelaars tegelijk, ieder in een bit.

OVERIGE ADRESSEN

53770 $D20A RANDOM

(R) Uit dit register worden de toevalswaarden gelezen. Het bevat de hoogste acht bits van de 17 bits polycounter. Zo bevinden zich hier voortdurend echte toevalsgetallen tussen 0 en 255. De BASIC-functie RND(0) heeft betrekking op dit register; het rekent alleen het gevonden toevalsgetal om in een decimale waarde tussen 0 en 1, door de waarde uit RANDOM door 256 te delen. De instructies PRINT RND(0) en PRINT PEEK (53770)/256 geven dus hetzelfde resultaat.

Om gehele getallen van 5 t/m 14 te krijgen, moet in BASIC de instructie INT(RND(0)*10)+5 worden gebruikt. RND(0)*10 geeft waarden van 0 t/m 9,99..., waarvan INT een geheel getal maakt (0 t/m 9). De toevoeging van 5 leidt dan tot de gewenste waarden van 5 t/m 14. INT(PEEK(53770)/25.6)+5 heeft dezelfde werking.

53774 $D20E IRQEN

(W) Interrupt-vrijgave. Het parallel-register hiervan is POKMSK (16 = $10). Nadere informatie kunt u daar vinden.

54272 $D400 DMACTL

DMA-besturingsregister. Wordt in BASIC beschreven via het parallel-register SDMCTL (559 = $22F). (Zie aldaar.)

54273 $D401 CHACTL

Karakter-modus besturingsregister. Wordt beschreven via het parallel-register CHACT (755 = $2F3). (Zie aldaar.)

54274,54275 $D402,$D203 DLISTL/H

LO- en HI-byte van de wijzer naar de display-list. (Zie hoofdstuk 'Display-list'.)

54279 $D407 PMBASE

(W) HI-byte van het PM-beginadres.

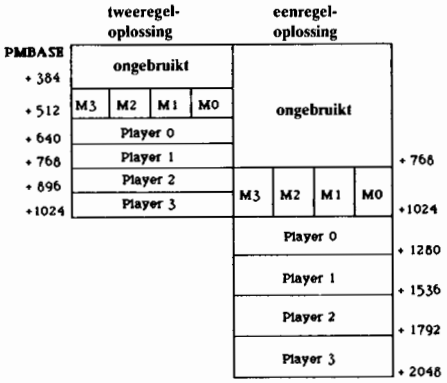
Voor de data van de players en de missiles bestaat er een geheugen-indeling, die ieder element een eigen geheugenplaats toewijst. Een player is altijd een byte breed en bedekt in vertikale richting het hele beeldscherm boven het door de GRAPHICS-modi gebruikte

grafische raam uit. Bij een resolutie van twee beeldlijnen is een player 128 byte hoog, bij een resolutie van een lijn gebruikt hij twee keer zo veel, namelijk 256 bytes, geheugen.

De vier missiles, die ook tot een vijfde player samengevoegd kunnen worden, gebruiken samen net zoveel geheugen als een player. Samen met een niet gebruikte veiligheidszone aan het begin van het PM-geheugen volgt daaruit een totaal gebruik van een Kbyte bij een resolutie van twee lijnen en 2 Kbyte bij een lijn.

Hoe het geheugengebied gewoonlijk is ingedeeld, is te zien op afbeelding 5:

PM-geheugen indeling



Het is beslist noodzakelijk deze indeling aan te houden, omdat bepaalde functies, als bijvoorbeeld de collision-registers op PMBASE, deze geheugenindeling steunen. Het is zinvol om de PM-tabel op te slaan aan het eind van het hoogste RAM-gebied, dus direkt voor de display-list en het beeldscherm-geheugen. Er is echter niets op tegen om een ander (beschermd) geheugengebied te gebruiken. Maar op welke pagina (HI-byte van het adres) het PM-geheugen ook begint, deze waarde moet altijd in PMBASE worden neergezet.

Om een veilig begin-adres te vinden voor de PM-graphics, kan men de geheugenruimte van de geactiveerde grafische modus en de daarbij horende display-list afgerond op hele pagina's plus de lengte van de PM-tabel, 4 of 8 pagina's aftrekken van RAMTOP.

Tot hetzelfde resultaat komt men als, na het inschakelen van de gewenste grafische modus, de wijzer naar de display-list SDLSTL (560,561 = $230,$231) wordt bekeken. Dit adres, afgerond op hele pagina's en afgetrokken van het PM-geheugen, geeft eveneens het PM-beginadres, waarvan de HI-byte in PMBASE neergezet moet worden.

Uitgaande van PMBASE kunt u de waarde voor APPMHI (14,15 = $E,$F) uitrekenen, als u het PM-geheugengebied van vier pagina's bij een resolutie van twee beeldlijnen en acht pagina's bij een lijn, optelt bij PMBASE. APPMHI markeert de onderste grens voor de display-list en het beeldscherm-geheugen.

54281 $D409 CHBASE

(W) Beginadres van de ATARI standaard karakterset, die natuurlijk in het ROM-gebied is opgeslagen. De karakterset bestaat uit 128 karakters van ieder acht bytes en gebruikt dus vier pagina's of een kbyte. De wijzer naar de karakterset kan door de BASIC-programmeur in het parallel-register CHBAS (756 = $2F4) worden veranderd. (Zie aldaar.)

Verdere verklaring over de structuur van de standaard karakterset en de mogelijkheid, vrij definieerbare karaktersets te gebruiken, vindt u in het hoofdstuk 'Karakterset'.

DISPLAY-LIST

De display-list (DL) is een machinetaal-programma voor de microprocessor ANTIC. Dit programma bepaalt, uit welk geheugengebied de grafische data worden gehaald en op welke manier ze op het beeldscherm worden weergegeven. Wanneer in BASIC een GRAPHICS-instructie wordt gegeven, wordt daarmee automatisch het benodigde geheugengebied voor het beeldscherm (SM = screen memory) gereserveerd en voor dit gebied de bijbehorende DL opgeslagen. Als men de opbouw van de DL en de ANTIC-

instructieset kent, is het mogelijk, de DL te veranderen of een geheel eigen DL te maken, waarin alle grafische weergavemogelijkheden van ANTIC gemengd kunnen worden.

Hoe een DL is opgebouwd, kunt u op het beeldscherm bekijken met het programma ADR560.BEC, dat bij SDLSTL (560,561 = $230,$231) is besproken.

ANTIC werkt met de volgende 8 bits instructies:

1 JMP, onvoorwaardelijke sprong-instructie. Direkt achter deze instructie moet het sprongadres volgen in de volgorde LO- en HI-byte. Deze instructie kan bijvoorbeeld worden gebruikt, om vanuit de DL naar een subroutine te springen, bijvoorbeeld een tweede kleine DL, die boven of onder het normale grafische raam een of meer grafische regels maakt, die onafhankelijk zijn van de grote DL.

2 Karakter-modus, wordt in BASIC door GRAPHICS opgeroepen. Een modus-regel is onderverdeeld in 40 kolommen en is acht beeldlijnen (scan lines) hoog. Er kunnen twee kleuren gelijktijdig worden weergegeven.

3 Karakter-modus, 40 kolommen, 10 beeldlijnen, 2 kleurwaarden.

4 Karakter-modus (GRAPHICS 12), 40 kolommen, 8 beeldlijnen, 5 kleurwaarden.

5 Karakter-modus (GRAPHICS 13), 40 kolommen, 16 beeldlijnen, 5 kleurwaarden.

6 Karakter-modus (GRAPHICS 1), 20 kolommen, 8 beeldlijnen, 5 kleurwaarden.

7 Karakter-modus (GRAPHICS 2), 20 kolommen, 16 beeldlijnen, 5 kleurwaarden.

8 Bit-map-modus (GRAPHICS 3), 40 pixels per regel, iedere modus-regel 8 beeldlijnen hoog, 4 COLOR-waarden.

9 Bit-map-modus (GRAPHICS 4), 80 pixels, 4 beeldlijnen, 2 COLOR-waarden.

10 Bit-map-modus (GRAPHICS 5), 80 pixels, 4 beeldlijnen, 4 COLOR-waarden.

11 Bit-map-modus (GRAPHICS 6), 160 pixels, 2 beeldlijnen, 2 COLOR-waarden.

12 Bit-map-modus (GRAPHICS 14), 160 pixels, 1 beeldlijnen, 2 COLOR-waarden.

13 Bit-map-modus (GRAPHICS 7), 160 pixels, 2 beeldlijnen, 4 COLOR-waarden.

14 Bit-map-modus (GRAPHICS 15), 160 pixels, 1 beeldlijn, 4 COLOR-waarden.

15 Bit-map-modus (GRAPHICS 8), 320 pixels, 1 beeldlijn, 2 COLOR-waarden.

(Het aantal pixels per regel heeft betrekking op de normale beeldscherm-breedte.)

0 een lege beeldlijn
16 twee lege beeldlijnen
32 drie lege beeldlijnen
48 vier lege beeldlijnen
64 vijf lege beeldlijnen
80 zes lege beeldlijnen
96 zeven lege beeldlijnen
112 acht lege beeldlijnen

64 opgeteld bij de modus-regel-instructie (dus bit 6 aan) activeert LMS (load memory scan), dat wil zeggen de grafische data worden uit het beeldscherm-geheugen gelezen. Na iedere LMS-instructie moet het begin-adres van de beeldscherm-data als LO- en HI-byte volgen.

65 JVB, van VBLANK afhankelijke sprong. Door deze instructie aan het einde van de DL wordt de synchronisatie tussen computer en beeldbuis bereikt. Aan het eind van de DL aangeland, wacht de computer op het VBLANK-signaal, voordat hij weer naar het begin van de DL springt en zo synchroon met de beeldhuis een nieuwe cyclus begint. Zoals bij iedere sprong-instructie moet ook na JVB direkt het sprongadres (LO, HI) volgen.

128 DLI (display list interrupt), opgeteld bij de modus-regel-instructie maakt de waarde 128 (bit 7) de onderbreking van de DL mogelijk, om tijdens HBLANK een korte machinetaal-routine te verwerken. Als voorbeeld-programma zie ADR512.BEC.

De standaard display-lists gebruiken 192 beeldlijnen. Bij de PAL-versie worden, om de systemen gelijk te trekken, eerst driemaal acht lege beeldlijnen weergegeven, zodat om precies te zijn 192+24, dus 216 beeldlijnen worden weergegeven. Door het veranderen van de DL kunnen de 24 lege regels ook daadwerkelijk benut worden. Een veranderde DL kan ook minder dan 216 respectievelijk 192 beeldlijnen weergeven. Dan blijft aan de onderste rand van het beeldscherm een brede lege (zwarte) balk staan. Door de JVB-instructie wacht de computer op de kathodestraal van de beeldbuis, zodat ook een willekeurig kortere DL altijd synchroon loopt met de beeldbuis.

Omdat het PAL-systeem de beeldbuis verdeelt in 312 beeldlijnen in plaats van 262 lijnen bij het NTSC-systeem, zorgen de 24 lege regels aan het begin van de DL ervoor, dat het grafische raam op het PAL-beeldscherm ongeveer in het vertikale midden van het scherm staat. Dat betekent echter, dat ook onder de standaard-DL bij PAL nog extra regels gebruikt kunnen worden en wel ongeveer 20. Is de DL echter langer dan de beeldbuis lijnen heeft, dan is de DL dus nog niet afgewerkt, als de beeldbuis bij de VBLANK aankomt en

wordt synchronisatie onmogelijk. Het beeld vervormt dan en begint te lopen. Tot beschadigingen van de beeldbuis leidt dit echter niet.

In het volgende programma kunt u zien, hoe een eigen display-list in BASIC geprogrammeerd kan worden:

```
0 REM DLTWUP.BEC
1 REM ********************************
2 REM *                              *
3 REM * GRAPHICS 8 TEKSTRAAM BOVEN   *
4 REM *                              *
5 REM ********************************
10 GRAPHICS 24:POKE 703,4
20 DL=PEEK(560)+PEEK(561)*256
30 FOR J=0 TO 9:READ B:POKE DL+J,B:NEXT J
40 DATA 112,66,96,159,2,2,2,79,80,129
100 COLOR 1
110 PLOT 0,0:DRAWTO 319,191
120 PLOT 319,0:DRAWTO 0,191
130 PLOT 0,0:DRAWTO 100,89
140 PLOT 100,93:DRAWTO 207,191
200 ? "GRAPHICS 8, TEKSTRAAM BOVEN"
```

programma DLTWUP.BWC

10: Of u nu alleen een standaard display een beetje wilt veranderen of een hele nieuwe display-list wilt samenstellen, u kunt er in BASIC niet omheen een grafische modus op te roepen, want de GRAPHICS-instructie verricht een aantal functies, die ook door PEEKs en POKEs niet zonder meer te vervangen zijn.

Een belangrijke functie bestaat eruit dat GRAPHICS geheugenruimte reserveert. Natuurlijk in de mate, waarin de opgeroepen modus geheugen voor de beelddata nodig heeft. Als u een eigen DL maakt, dan moet u de SM-grootte berekenen en een standaard grafische modus inschakelen, die minstens even groot of groter is. De grootte van de verschillende modus-regels vindt u in een tabel in het hoofdstuk 'Beeldscherm-geheugen'.

Het oproepen van een grafische modus door de BASIC-instructie GRAPHICS zet ook de bijbehorende display-list neer op de plaats in het geheugen, die door de wijzer SAVMSC (88,89 = $58,$59) wordt aangegeven. Als het OS deze functies verricht, geeft dat geen problemen, maar als u het allemaal zelf wilt doen, dan moet u goed opletten. Een display-list mag namelijk geen 1k-grens overschrijden zonder een vernieuwde sprongaanwijzing (JMP, zie hiervoor). De langste DL, die van GRAPHICS 8, is 202 bytes lang. Het zal dus altijd wel mogelijk zijn, een DL tussen 1k-grenzen te zetten. Maar men moet er wel op letten.

Desondanks kan een DL een aanzienlijke lengte krijgen. De LMS-instructie voor ANTIC bestaat hieruit, dat in de byte met de instructie voor een modus-regel wordt aangezet, dus

decimaal gezien bij de waarde voor een modus-regel 64 wordt opgeteld. Het is theoretisch mogelijk iedere modus-regel afzonderlijk van een LMS te voorzien en daarmee iedere grafische regel een eigen geheugengebied toe te bedelen. Iedere LMS-instructie moet echter gevolgd worden door de wijzer naar het beeldscherm-geheugen. En dat zijn twee bytes die erbij komen. Hoe langer de DL wordt, des te meer groeit het gevaar, dat de 1k-grens wordt overschreden.

Een tweede beperking is dat het geheugen met de grafisch data (SM) geen 4k-grens mag overschrijden, zonder dat in de DL een LMS-instructie wordt gegeven. Wanneer u de geheugen vretende, lange DLs bekijkt van GRAPHICS 8 t/m 11, 14 en 15 (SM-grootte 7680 bytes), dan kunt u iets onder het midden een nieuwe LMS-instructie vinden, met de daarbij horende wijzer naar het adres iets onder het midden van het beeldscherm-geheugen. De grafische data voor het onderste deel van het beeldscherm gaan daar verder.

De ingeschakelde grafische modus bepaalt ook hoe de data uit het SM worden verwerkt, dus welk bit-masker (DMASK 672 = $2A0) gebruikt wordt. Dat wil zeggen hoeveel bits per pixel gelezen worden en hoeveel kleurenregisters gebruikt worden. Deze functie kan worden beïnvloed, wanneer in DINDEX (87 = $57) een waarde wordt neergezet, die een andere grafische modus simuleert dan degene die met GRAPHICS is geprogrammeerd.

GRAPHICS of de andere in DINDEX opgeslagen waarde beïnvloedt echter wel weer alle instructies die betrekking hebben op het beeldscherm, zoals PLOT, LOCATE, enzovoorts.

Hier in regel 10 van het programma wordt ook nog de grootte van het tekstraam ingesteld op vier (GRAPHICS 0) regels, want er werd alleen GRAPHICS 24 geprogrammeerd, dus 8 zonder tekstraam, omdat de daarbij horende DL beter past bij het doel van dit programma. Het tekstraam moet alleen naar de bovenste beeldschermrand worden verplaatst. Daarom wordt de DL zonder tekstraam gekozen. Aan de onderkant zijn dan geen verdere veranderingen nodig.

20: berekent het begin-adres van de DL en dan begint in regel

30: het veranderen. De FOR-NEXT lus leest tien waarden en POKEt ze in de DL.

40: Hier staan die waarden. 112 maakt 8 lege beeldlijnen. 66 is LMS (64) + een GRAPHICS 0 regel (2). De nieuwe DL begint dus 16 beeldlijnen hoger dan normaal. 96 en 159 zijn LO en HI van de wijzer naar het geheugengebied van het tekstraam.
Daarna volgen nog drie GRAPHICS 1 regel. 79 bevat weer een LMS-instructie (64) + de instructie-waarde 15 voor een GRAPHICS 8 regel, gevolgd door de LO- (80) en HI-byte (129) van de SM-wijzer SAVMSC. De rest van de DL kan onveranderd blijven.

Dat was het. Wanneer u op deze plaats het commando RUN geeft, is de fraaie, nieuwe DL-wereld geheel in orde. Aan de bovenkant van het beeldscherm verschijnt READY en u kunt daar dezelfde dingen doen, die ook normaal in het tekstraam mogelijk zijn.

Wanneer u zich echter in het GRAPHICS 8 gebied verder waagt, dan blijven de problemen niet uit. Voor dit demonstratie-programma werd bewust gekozen voor GRAPHICS 8, omdat bij de hogere modi bepaalde problemen opduiken, namelijk met de hiervoor reeds besproken k-grenzen.

Het nieuwe GRAPHICS 8 gebied begint gelijk onder het tekstraam en werkt in het bovenste gedeelte ook heel mooi met PLOT en alles wat daarbij hoort. Dan komt echter de grens, waar in de DL de nieuwe LMS-instructie staat met de wijzer naar de rest van de SM. Nu is echter het nieuwe grafische raam niet meer 192 modus-regels (GRAPHICS 24!) lang, zoals het OS verwacht en voor alle PLOTs aanneemt, maar slechts 176 regels. Het verschil van 16 modus-regels moet ergens blijven.

100 en 120: PLOT en DRAWTO trekken hier een diagonaal kruis over het gehele grafische raam, en u ziet, in het midden ontbreekt een streep. Deze misère is ook niet zomaar op te heffen door het veranderen van de wijzer na de tweede LMS-instructie, want het OS berekent de PLOTs, die door de DRAWTO-instructie worden gemaakt, in de veronderstelling dat gewoon GRAPHICS 8 (respectievelijk 24) is ingeschakeld.

130 en 140: Er blijft in zo'n geval niets anders over dan alle beeldscherm- instructies voor de beide delen apart te programmeren.

200: Alleen het tekstraam werkt zonder problemen.

De hier voorgestelde problematische DL is ook een schoolvoorbeeld van de weg, die aanvankelijk eenvoudig leek, maar uiteindelijk niet de beste is. Als u namelijk in regel 10 niet GRAPHICS 24 maar GRAPHICS 8 oproept en de DL zo verandert, dat u aan de onderkant het tekstraam weghaalt en er bovenin een maakt, zonder lege beeldlijnen te gebruiken, die immers ook SM nodig hebben, dan klopt het aantal geheugenplaatsen weer precies en krijgen we geen problemen. Probeert u het maar!

In het tweede voorbeeld-programma wordt in het midden van het GRAPHICS 0 beeldscherm een zone ingericht, waarin het schrift sterk vergroot verschijnt als onder een loep:

GRAPHICS 8, TEKSTVENSTER BOVEN

De vergroting van de tekst wordt bereikt door tussengeschoven GRAPHICS 2 regels. De karakters worden in GRAPHICS 2 met dubbele breedte weergegeven en een modus-

regel is twee keer zo groot als in GRAPHICS 0, namelijk 16 beeldlijnen. De beide nieuwe regels gebruiken dus op het beeldscherm de plaats van vier GRAPHICS 0 regels.

Opnieuw worden 16 inactieve regels aan de bovenste rand omgezet. De GRAPHICS 0 DL wordt als het ware naar boven verschoven, om dan in de programmaregels 80 en 90 de beide nieuwe modus-regels tussen te voegen.

Als u dit programma laat lopen, wordt door het GRAPHICS-statement in regel 10 het beeldscherm gewist. U ziet echter onmiddellijk de verandering, omdat de modus-regels van GRAPHICS 2 een zwarte ondergrond hebben.

Geeft u nu eens het commando LIST! De editor-modus (GRAPHICS 0) is aktief. Er kunnen gewoon alle instructies worden gegeven. Als de listing op het beeldscherm verschijnt, ziet u regel 30 in mooie grote oranje letters op een zwarte ondergrond. U kunt in deze uitgelichte regel ook programmeren. Gaat u maar eens met de cursor naar regel 30. He! Waar is de cursor opeens?

De cursor ontstaat door een functie, die de waarde van het karakter, waar de cursor op staat, met 128 verandert.
Daardoor worden de tekens geïnverteerd. In GRAPHICS 2 zijn er echter geen inverse tekens. In plaats daarvan verandert de kleurwaarde.

Zo! Ga nu met de blauwe letter-cursor naar de dubbele punt en verwijder deze evenals de daarop volgende zinloze REM.
Vergeet niet RETURN, we voeren immers een nieuwe BASIC-regel in. Als u het programma nu opnieuw laat uitlisten is regel 30 gecorrigeerd.

Aan de onderste rand van het beeldscherm heeft u waarschijnlijk al enige vreemde karakters ontdekt. Nee, u hebt niets verkeerd gedaan. De door GRAPHICS 0 instructie gereserveerde geheugenruimte is voor deze DL niet groot genoeg. Tenminste voor de laatste regel is geen ruimte meer.
De DL leest wel netjes de inhoud van de geheugenplaatsen, maar bevindt zich al in een gebied, dat niet meer tot het SM behoort. De daar gevonden waarden worden ook in karakters omgezet, maar betekenis hebben ze niet. U kunt ook niet met de cursor naar deze laatste regel toe. De editor verwerkt 24 GRAPHICS 0 regels, daar is hij zeer getrouw in. Als de DL wordt veranderd, dan verandert de editor nog lang niet zijn eigenschappen.

Er zijn ook nog andere problemen. In GRAPHICS 2 is slechts de helft van de karakterset bereikbaar en de nu gemaakte verandering kan alleen een fysieke GRAPHICS 0 regel opnemen, dus geen hele logische regel. Het is echter mogelijk, het gehele GRAPHICS 0 beeldscherm om te zetten in GRAPHICS 2, om dan in GRAPHICS 2 programma's te kunnen invoeren. Voor slecht ziende mensen kan dat een zeer nuttige hulp zijn. Als u zich wilt

beperken tot het weergeven van een fysieke regel, dan is het zinvol, deze aan de bovenste rand van het beeldscherm neer te zetten.

Beeldscherm-geheugen

Als screen memory (SM, beeldscherm-geheugen) wordt het bereik aangeduid, waarin de data opgeslagen worden, die het grafische uiterlijk van het beeldscherm bepalen. De LMS-instructie in de DL verwacht grafische data te vinden. Op welke manier deze data in grafische informatie worden omgezet, wordt bepaald door de instructie voor de modus-regel.

Er zijn twee principieel verschillende besturings-modi: de character mode en de bit-map mode.

In de karakter-modus wordt ieder data-byte, die in de SM wordt gelezen, in een karakter omgezet. Een karakter is een bepaalde rangschikking van beeldpunten. De betrekking tussen de karakters, de waarden, waardoor ze worden vervangen en het puntenpatroon, dat ze weergeeft op het beeldscherm, is vastgelegd in de karakterset (zie hoofdstuk 'Karakterset'). De karakters van de karakterset zijn geordend door waarden van 0 t/m 127. Deze waarden wijken echter af van de ATASCII-code. Men spreekt van de interne code.

In de karakter-mode wordt de waarde uit het beeldscherm-geheugen geïnterpreteerd als de interne code van een karakter. Het karakter wordt uit de karakterset gelezen en de bit-patronen, die het representeren, worden op het beeldscherm weergegeven.

De BASIC-instructie LOCATE x,y,n zoekt in het beeldscherm-geheugen naar de geheugenplaats die overeenkomt met de beeldscherm-positie x,y, leest de daar opgeslagen waarde (interne code) en zet deze om in ATASCII. LOCATE vindt in de karakter-modus de ATASCII-waarde van het teken dat op de aangegeven plaats op het beeldscherm staat.

Overigens werkt de LOCATE-instructie alleen als er een GRAPHICS-instructie aan vooraf gegaan is. Dat betekent, dat ook GRAPHICS 0 extra geprogrammeerd moet worden, wat normaal niet nodig is.

Ook GRAPHICS 1 en 2 zijn karakter-modi. Het onderscheid bestaat hierin, dat alleen de bits 0 t/m 5 worden gelezen, om het karakter in de karakterset te vinden. Daardoor kan alleen de halve karakterset, interne code 0 t/m 63, worden weergegeven. De bits 6 en 7 worden als kleurinformatie beschouwd. Met twee bits kunnen vier verschillende waarden (0 t/m 3) worden weergegeven. Daarom kunnen de karakters in deze beide modi in vier verschillende kleuren worden weergegeven. Een vijfde kleur is voor de achtergrond.

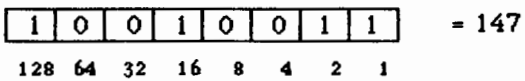
Ook de karakter-modi 12 en 13 kunnen de karakters in meerdere kleuren weergeven. Desondanks kan de gehele karakterset opgeroepen worden, doordat de kleurinformatie op een andere manier wordt verkregen. Bij een standaard karakter representeert ieder bit een beeldpunt, dat dus in twee kleuren kan verschijnen, namelijk in de kleur van het karak-

ter of van de achtergrond. In deze beide grafische modi worden steeds twee bits gebruikt voor een beeldpunt, dat daardoor vier verschillende kleurwaarden (-registers) kan gebruiken. Worden de karakters uit de standaard karakterset in deze modi weergegeven, dan worden ze bijna onleesbaar, omdat de grafische vorm nog slechts gedeeltelijk wordt omgezet, terwijl er onzinnige kleurinterpretatie achteraan komt. (In het hoofdstuk 'Karakterset' wordt dit nog nader verklaard.)

In de bit-map modi worden pixels op het beeldscherm weergegeven, waarvan de grootte afhangt van de resolutie van de gekozen grafische modus.

Als ieder bit een grafisch punt weergeeft, dan kan het grafische punt slechts twee kleuren aannemen, namelijk 0 of 1. Welke kleur (COLOR) 0 of 1 representeert, wordt bepaald in het door COLOR opgeroepen kleur-register (zie tabel) door POKE of SETCOLOR. Acht van zulke grafische bits bevinden zich in een byte van het SM. Al naar gelang de ligging van de bits binnen de geheugen-bytes, respectievelijk de pixels op het beeldscherm, krijgt het bit een gewicht op de bekende manier:

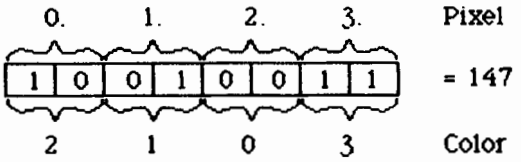
afbeelding twee kleuren



De waarde 147 op een geheugenplaats van het SM produceert de hierboven getoonde instelling van de pixels op het beeldscherm.

De grafische modi 1, 4, 6, 8 en 14 werken met twee kleuren. Zijn er vier kleuren mogelijk, dan zijn voor iedere pixel twee bits informatie nodig:

afbeelding vier kleuren

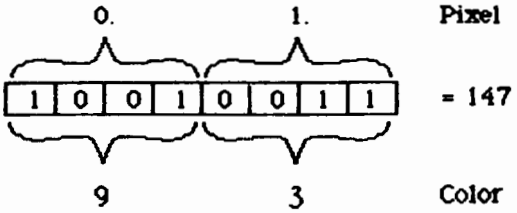


De data-byte 147 produceert dan een rij van vier pixels in de kleuren 2, 1, 0 en 3. In het beeldscherm-geheugen wordt dus de COLOR-waarde van de pixels in binaire vorm opgeslagen. Boven de COLOR-waarde worden de desbetreffende kleurenregisters gevonden. Omdat de PEEK-instructie altijd alleen maar de decimale waarde van een byte kan vinden, vormen de bits van meerdere pixels een gemeenschappelijke decimale waarde.

De grafische modi 3, 5, 7 en 15 werken met twee bits per pixel.

In de GTIA-modi 9, 10 en 11 worden vier bits gebruikt voor een pixel. Met vier bits kunnen zestien kleurwaarden worden onderscheiden. Omdat de ATARI echter slechts negen kleurregisters heeft, kunnen er geen zestien kleuren gedefinieerd worden. In GRAPHICS 9 roepen COLOR 0 t/m 15 daarom verschillende helderheden van een kleur op. In GRAPHICS 11 kunnen de zestien standaard kleuren worden gebruikt, maar de helderheid is voor allemaal gelijk. In GRAPHICS 10 kunnen de kleurwaarden wel vrij worden gedefinieerd, maar omdat er slechts negen kleurenregisters zijn, kunnen ook maar negen COLOR-instructies worden gegeven:

afbeelding zestien kleuren



In een GTIA-modus produceert 147 twee pixels met de COLOR-waarden 9 en 3.

Het beeldscherm-geheugen is lineair ingedeeld. De eerste SM-byte bepaalt de inhoud van de linker bovenhoek van het grafische raam. Daarna volgen de data voor de rest van de bovenste regel tot de rechterrand, dan de data voor de tweede en de volgende regels tot en met de laatste waarde, die het uiterlijk van de hoek rechtsonder bepaalt.

Hoeveel bytes een regel gebruikt is afhankelijk van de grootte van de pixels en het aantal mogelijke kleuren, dus het aantal bits per pixel.

In GRAPHICS 3 staan 40 pixels op een regel. Omdat er vier kleuren mogelijk zijn, gebruikt iedere pixel twee bits. Een GRAPHICS 3 regel gebruikt dus 809 bits = 10 bytes. Om een bepaald punt op het beeldscherm te bereiken, moet worden bepaald, in welke byte de data voor dit punt zijn ondergebracht. Als het punt de coördinaten X,Y heeft, dan liggen zijn data in de

byte = SAVMSC + Y*rpr * INT(X/ppb)

waarbij rpr RAM per regel betekent of bytes per regel en ppb pixels per byte. SAVMSC is het beginadres van het beeldscherm-geheugen. De waarde in deze wijzer duidt op de eerste byte van het SM.

Nadat de juiste byte is gevonden, moet nog worden vastgesteld op welke plaats binnen de byte de data voor het aangestuurde grafische punt liggen.

In GRAPHICS 3 moet op de positie 5,2 een pixel in COLOR 2 worden opgeslagen. De positie 5,2 heeft een offset vanaf SAVMSC van 21 bytes. In de 21e byte onder SAVMSC liggen de pixels met de volgende beeldscherm-coördinaten:

bit	7	6	5	4	3	2	1	0
decimale waarde	128	64	32	16	8	4	2	1
POSITION					-4,2-	-5,2,-	-6,2-	-7,2-
COLOR					0 0	1 0	0 0	0 0

Op de positie 5,2 wordt de COLOR-waarde 2 (binair 10) neergezet. Als de overige drie punten zwart blijven, dan krijgt byte 21 de waarde 32. De BASIC-instructies:

COLOR 2:PLOT 5,2

kunnen worden vervangen door de instructie:

POKE SM+21,32

waarbij natuurlijk eerst SM als beginadres van het beeldscherm uit de wijzer SAVMSC berekend moet worden:

SM = PEEK(88) + PEEK(89)*256

Moeten afzonderlijke grafische punten op het beeldscherm worden afgezet, dan is de PLOT-instructie natuurlijk veel eenvoudiger te gebruiken. Maar ook de POKE in het SM heeft enige voordelen.

POKE is onafhankelijk van cursor-posities, die na het veranderen van de DL door het OS eventueel als ontoelaatbaar beschouwd kunnen worden. Met POKE kan men dus beeldscherm-gebieden beschrijven, die met PLOT en DRAWTO niet meer toegankelijk zijn.

Een andere interessante toepassing bestaat uit het opslaan van beeldscherm-inhouden, wanneer byte voor byte uit het beeldscherm-geheugen wordt gePEEKt en over een datakanaal op de diskette wordt gePUT. Omgekeerd kunnen dan de data van de diskettefile byte voor byte worden geGET en dan in het beelscherm-geheugen worden gePOKEd.

Een toepassing voor de direkte adressering van het beeldscherm geeft het volgende voorbeeld-programma:

```
0 REM SMPOKE.BEC
1 REM ********************************
2 REM *                              *
3 REM *          LICHTKRANT          *
4 REM *                              *
5 REM ********************************
10 S=40040:? CHR$(125)
20 FOR P=0 TO 36:READ B:POKE S+P,B:NEXT P
30 DATA 0,10,0,10,0,36,105,116,0,105,115,0,101,101,110,0,108,105,99,104,116,107,
114,97,110
40 DATA 116,13,116,101,115,116,0,10,0,10,0,10
50 J=PEEK(S):K=PEEK(S+1):L=PEEK(S+2):M=PEEK(S+3)
60 FOR I=4 TO 39
70 POKE S+I-4,PEEK(S+I)
80 NEXT I
90 POKE S+36,J:POKE S+37,K:POKE S+38,L:POKE S+39,M
100 GOTO 50
```

programma SMPOKE.BEC

10: S is een adres van het SM.

20: De FOR-NEXT lus leest voor een beeldscherm-regel (0 t/m 39) de data-bytes en zet ze op de SM-geheugen-adressen, die de gewenste tekst weergeven.

30 en 40: De karakterwaarden moeten overeenkomen met de interne code.

50: De inhoud van de eerste vier adressen wordt in variabelen opgeslagen.

60 t/m 80: dan wordt de inhoud van de overige geheugenplaatsen van de beeldschermregel vier plaatsen naar boven verschoven, dat is op het beeldscherm vier kolommen naar links, en

90: tenslotte worden de vier bewaarde karakterwaarden op de vier achterste plaatsen van de regel gePOKEd.

Door deze bewerking worden de karakters van een beeldscherm-regel bij iedere programma-doorloop vier kolommen naar links verschoven. Het is natuurlijk ook mogelijk in iedere programma-doorloop slechts een karakter te verschuiven, maar BASIC is zo langzaam, dat daarbij geen lopende tekst maar een kruipende tekst ontstaat.

Nog belangrijker is de direkte adressering van het beeldscherm voor het tekstraam, omdat daar de POSITION-instructie niet werkt. Met een SM-POKE kan men karakters willekeurig en betrouwbaar neerzetten en daarmee voortdurend wisselende waarden op steeds dezelfde plaats weergeven, bijvoorbeeld een teller:

```
0 REM TWPOKE.BEC
1 REM ********************************
2 REM *                              *
3 REM *   POKEN IN HET TEKSTRAAM     *
4 REM *                              *
5 REM ********************************
10 DIM Z$(5):P=40897:? CHR$(125):N=25
20 FOR J=0 TO 4:POKE P+J,16:POKE P+J,16:NEXT J
30 Z=Z+1
40 Z$=STR$(Z)
50 L=LEN(Z$)
60 FOR J=1 TO L
70 POKE P+4-L+J,VAL(Z$(J,J))+16
80 NEXT J
90 IF PEEK(P)=N THEN IF PEEK(P+1)=N THEN IF PEEK(P+2)=N THEN IF PEEK(P+3)=N THEN
 IF PEEK(P+4)=N THEN GOSUB 200
100 REM IF Z=99999 THEN GOSUB 200
110 GOTO 30
200 RESTORE 300
210 FOR J=0 TO 8
220 READ D
230 POKE P-10+J,D
240 POKE P+6+J,D
250 NEXT J
260 Z=0:FOR J=0 TO 4
270 POKE P+J,16
280 NEXT J
290 RETURN
300 DATA 47,54,37,50,38,44,47,55,1
```

programma TWPOKE.BEC

10: In Z$ wordt de getallenvolgorde van de tellerstand opgeslagen. De linker bovenhoek van het tekstraam staat bij alle standaard-DLIs op adres 40800.

20: Op de vijf plaatsen van de teller worden 0- en geschreven, interne code = 16.

30: Z telt.

40: Om de tellerstand op het beeldscherm te POKEn, moet Z in afzonderlijke cijfers worden opgesplitst. De eerste stap bestaat uit het omzetten van de numerieke variabele in een string Z$, omdat uit een string de afzonderlijke elementen gemakkelijk gelezen kunnen worden.

50: Opdat de afzonderlijke cijfers op de juiste plaats in het weergegeven getal belanden, moet eerst worden vastgesteld hoeveel plaatsen het getal en hoeveel elementen Z$ heeft.

60 t/m 80: De FOR-NEXT lus leest uit de cijferstring ieder afzonderlijk cijfer (Z$(J,J)), zet ze om in een numerieke waarde (VAL) en verandert de cijfers in hun interne karaktercode door er 16 bij op te tellen. De interne code van 1 is 17, van 2 18, enzovoorts. De zo verkregen karakterwaarde wordt op de plaats P+4-L+J gePOKEd. Zo komt ieder cijfer op de juiste plaats.

90: Ja, ook zo'n lange rij instructies kan de ATARI verwerken! Hij stelt vast of alle cijfers op 9 (interne code = 25) staan.

100: Natuurlijk is dat ook eenvoudiger te bepalen.

110: sluit de lus af door het programma terug te laten springen naar regel 30.

200 t/m 300: Wanneeer de teller tot 99999 heeft geteld, wordt hier...nee, dat verklap ik niet, dat moet u zelf uitproberen. Als u de teller helemaal laat doortellen, heeft u in ieder geval veel geduld nodig.

Bij het verspringen van 99999 op 00000 kunt u zien, dat de teller bij de kleine getallen veel sneller loopt. Dat komt, doordat de cijferstring korter is en er daardoor minder programmastappen nodig zijn. BASIC is echter zo traag, dat men zulke kleinigheden met het blote oog kan zien.

Met het tekstraam is nog iets bijzonders aan de hand. Het beschikt namelijk over een eigen, onafhankelijk beeldscherm-geheugen. De wijzer TXTMSC (660,661 = $294,$295) wijst naar het begin-adres, dat bij alle standaard DLIs 40800 is.

Onverschillig of u een grafische modus met of zonder (+16) tekstraam oproept, voor het grafische raam wordt altijd geheugenruimte gereserveerd, die data voor het gehele beeldscherm-volume kan opnemen. Daarachter ligt het 160 bytes grote geheugen voor het tekstraam. Deze onafhankelijkheid is ook de reden, waarom PLOTs en POSITIONs hier niet werken.

Aan de andere kant opent deze scheiding de mogelijkheid het geheugen van het tekstraam te beschrijven, hoewel dit helemaal niet is ingeschakeld. Het gehele tekstraam in het

grafische raam is in en uit te schakelen, zonder dat data van het tekst- of grafische raam daarbij verloren gaan. Dit is echter alleen mogelijk als bij het geven van de GRAPHICS-instructie ervoor gezorgd wordt, dat de beeldinhoud in het geheugen niet zoals gewoonlijk wordt gewist. Dit gebeurt door het aanzetten van bit 5 van het GRAPHICS-rangnummer:

bit 7 (=128) niet gebruikt
bit 6 (= 64) niet gebruikt
bit 5 (= 32) beeldinhoud niet wissen
bit 4 (= 16) zonder tekstraam
bit 3 t/m (=15 t/m 0) grafische modi 0 t/m 15 tekstraam
bit 0

De programma-stappen voor het in- en uitschakelen van het tekstraam staan in het volgende voorbeeld:

```
0 REM TWINOUT.BEC
1 REM ********************************
2 REM *                              *
3 REM *  TEKSTRAAM IN/UITSCHAKELEN   *
4 REM *                              *
5 REM ********************************
10 GOSUB 300
20 Z=Z+1:Z$=STR$(Z):L=LEN(Z$)
30 FOR J=1 TO L:POKE P+6-L+J,VAL(Z$(J,J))+16:NEXT J
40 IF PEEK(764)=255 THEN 20
100 OPEN £1,4,0,"K:":GET £1,T:CLOSE £1
110 YA=Y:Y=Y-((T=28) AND (Y>0))
120 Y=Y+((T=29) AND (Y<23))
130 XA=X:X=X-((T=30) AND (X>0))
140 X=X+((T=31) AND (X<39))
150 IF T=73 THEN GRAPHICS 3+32:POKE 708,36:POKE 709,38:POKE 710,208:POKE 712,208
160 IF T=85 THEN GRAPHICS 3+16+32:POKE 708,50:POKE 709,52:POKE 710,208:POKE 712,
208
200 COLOR 0:PLOT 0,YA:DRAWTO 39,YA:PLOT XA,0:DRAWTO XA,23
210 COLOR 1:PLOT 0,Y:DRAWTO 39,Y:COLOR 2:PLOT X,0:DRAWTO X,23:GOTO 20
300 DIM Z$(7):P=40896
310 GRAPHICS 19:POKE 708,50:POKE 709,52:POKE 710,208:POKE 712,208
320 FOR J=0 TO 6:POKE P+J,16:NEXT J
330 COLOR 2:PLOT 0,0:DRAWTO 0,23
340 COLOR 1:PLOT 0,0:DRAWTO 39,0
350 RETURN
```

10: Het is niet altijd zinvol, het programma chronologisch op te bouwen, dus in de eerste regel bijvoorbeeld met de initiatie te beginnen. Waarom niet? Lees maar verder:

20 en 30: bevatten dezelfde statements als de hiervoor besproken teller, die echter tot zeven plaatsen is uitgebreid en in minder logische regels is opgeborgen. Een BASIC-programma loopt sneller als het minder logische regels heeft, vooral als het veel spronginstructies bevat, omdat BASIC bij iedere sprong van boven begint en dan alle regelnummers doorloopt, totdat het eindelijk de gezochte regel heeft gevonden. Daarom kunnen de regels, waar vaak naartoe wordt gesprongen, het beste bovenin en weinig aktieve of slechts eenmaal te doorlopen delen het beste onderin het programma worden gezet.

40: In CH (764 = $2FC) staat de toetscode van de laatst ingedrukte toets. Hier staat de waarde 255, als geen toets is ingedrukt. Deze regel zorgt ervoor dat het programma ongehinderd verder loopt, want:

100: de GET-instructie onderbreekt het programma en wacht op een gebruikers-invoer. Daarbij zou echter ook de teller stilstaan.

110 t/m 140: In het grafische raam bevindt zich een draadkruis. Met de vier cursor-toetsen kunt u het hier in alle richtingen bewegen. XA en YA bewaren de oude waarden van X en Y, om later op de oude positie het draadkruis te kunnen wissen. De vier Boolse uitdrukkingen stellen vast of de betreffende toets (T) is ingedrukt en of de grenswaarde van X of Y nog niet is bereikt en vernieuwen dan overeenkomstig de waarde van X of Y.

150: Wordt de toets "E" ingedrukt, dan wordt hier GRAPHICS 3 (dus met tekstraam) ingeschakeld. De +32 zorgt er tegelijkertijd voor, dat de beeldinhoud niet wordt gewist. Kortom, het tekstraam wordt ingeschakeld. U kunt natuurlijk ook meteen GRAPHICS 35 schrijven in plaats van 3+32.

In ieder geval zet iedere GRAPHICS-instructie weer de standaardwaarden in de kleurregisters. Daarom moeten de gedefinieerde kleuren hier worden vernieuwd. Door dezelfde kleurwaarde voor de achtergrond (712) en COLOR 3 (710) wordt overigens bereikt, dat tekst- en grafisch raam een optische eenheid vormen, hetgeen de inschakeling van het tekstraam indrukwekkender maakt.

Nog indrukwekkender zou het ongetwijfeld zijn om het tekstraam smaller te maken. Voor dit doel zou hier zelfs een enkele regel voldoende zijn. Omdat daarvoor de DL veranderd zou moeten worden, zou deze in te lassen regel ook nog iedere willekeurige positie op het beeldscherm kunnen innemen.

160: Wordt de toets 'A' ingedrukt, dan wordt met het GRAPHICS-rangnummer 3+16+32 (=51) het tekstraam weer uitgeschakeld. Omdat bij het in- en uitschakelen de kleurenregisters steeds opnieuw beschreven moeten worden, krijgt de cursor licht afwijkende kleurwaarden, als de teller tussengevoegd is.

Belangrijk is hier, dat zelfs bij ingeschakeld tekstraam het centrum van het draadkruis helemaal tot aan de onderste rand van het beeldscherm bewogen kan worden. Dus daarheen, waar op het moment het tekstraam staat, zonder dat de computer een foutmelding geeft wegens ontoelaatbare cursor-positie. Dat werkt echter alleen, als aan het begin de grafische modus zonder tekstraam wordt ingeschakeld. Als later dan ook het tekstraam wordt ingeschakeld, kunnen instructies als PLOT en dergelijke toch het grafische raam in het door het tekstraam gebruikte gebied beschrijven, zelfs als het tijdelijk niet te zien is.

200: overschrijft het oude draadkruis met de kleur van de achtergrond, dus wist het, en regel

210: PLOT het nieuwe draadkruis.

300 t/m 350: De statements die aan het begin van een programma nu eenmaal nodig zijn, kunnen het beste aan het einde van het programma worden neergezet. Hier worden de grafische en kleurwaarden gedefinieerd, de zeven plaatsen van de teller met 0-en gevuld en de beide assen van het draadkruis gePLOT.

De volgende tabel geeft u een overzicht van de verschillende grafische modi:

tabel uit atari peeks en pokes

[A] verschijnt negatief

	GRAPHICS	ANTIC commando	KOLOM pixel/ regel	MODE regels	bytes per regel	TV-regels per modus-regel	bits per pixel	kleuren-register 708	709	710	711	712**
karakter-mode	0	2	40	20/24	40	8	8		C.1*	C.0*		R
	-	3	40	16	40	10	8		C.1	C.0		R
	12	4	40	20/24	40	8	8	A/[A]	A/[A]	A	[A]	H
	13	5	40	10/12	40	16	8	A/[A]	A/[A]	A	[A]	H
	1	6	20	20/24	20	8	8	A	a	[A]	[a]	H
	2	7	20	10/12	20	16	8	A	a	[A]	[a]	H
bit-map mode	3	8	40	20/24	10	8	2	C.1	C.2	C.3		H
	4	9	80	40/48	10	4	1	C.1				H
	5	10	80	40/48	20	4	2	C.1	C.2	C.3		H
	6	11	160	80/96	20	2	1	C.1				H
	14	12	160	160/192	20	1	1	C.1				H
	7	13	160	80/96	40	1	2	C.1	C.2	C.3		H
	15	14	160	160/192	49	1	2	C.1	C.2	C.3		H
	8	15	320	160/192	40	1	1		C.1	C.0		R
gira-mode	9		80	192	40	1	4	16 helderheids-gradaties				
	10		80	192	40	1	4	9 kleuren naar keuze				
	11		80	192	40	1	4	16 kleuren/vaste helderheid				

* In alle GRAPHICS-modi met tekstraam bepaalt 710 de kleur van de tekst en de achtergrond en 709 de helderheid van de tekst in het tekstraam.
C = COLOR, R = RAND, A = ACHTERGROND.

** De kleurwaarde van register 712 wordt in alle GRAPHICS-modi opgeroepen met COLOR 0.

Karakterset

Om schriftelijke informatie op het beeldscherm weer te kunnen geven, moet de computer weten hoe de schrifttekens eruit zien, waarmee de mensen zich verstaanbaar maken. De computer zelf beperkt zich strikt tot getallen.

Er gelden drie verschillende systemen voor de toewijzing van getalwaarden aan de voorraad karakters (de karakterset) op verschillende gebieden van het systeem.

In de eerste plaats is er de toetscode, die aan iedere toets van het toetsenbord een numerieke waarde toekent van 0 t/m 63. De tweede helft van de karakterset wordt al sinds de schrijfmachine bereikt via een functietoets, waarvoor de naam SHIFT is ingeburgerd, met dit verschil dat bij ATARI het toetsenbord in de uitgangstoestand hoofdletters geeft. Met de CAPS-toets kan de schrijfmachine-modus worden vastgezet. Dan levert het toetsenbord kleine letters en verschijnen hoofdletters alleen, als gelijktijdig de SHIFT-toets wordt ingedrukt.

Met de computer is nog een andere functietoets ingevoerd, om de drievoudige werking van iedere toets mogelijk te maken. Hiervoor is de aanduiding CONTROL of CTRL gebruikelijk.

Een derde toets heeft invloed op het uiterlijk van de karakters en is bij iedere computer te vinden. Het is de INVERS-toets. Wordt deze ingedrukt, dan worden alle volgende tekens geïnverteerd (ze komen dus in negatieve weergave op het beeldscherm).

Sinds de invoering van de telex werden de firma's (die anders iedere gelegenheid voor individuele standaards flink uitbuiten) gedwongen het eens te worden over een norm, om de overdracht van berichten ook tussen verschillende merken te garanderen. Deze standaard heet ASCII-code. Deze code kent aan ieder teken en iedere functie van de telex als regelopschuiving, tabulator, enzovoorts een getalwaarde toe van 0 t/m 127.

Deze ASCII-code vormt tegenwoordig ook de basis voor regeldrukkers en computers. Het is overbodig te vermelden, dat aan deze Amerikaanse norm pas later ook internationale tekens (a umlaut, o umlaut, u umlaut en sz) werden toegevoegd, die echter in de eerste 128 tekens, dus in de eigenlijke karakterset, geen plaats vonden.

De eerste 32 tekens van de ASCII-code zijn voor de besturing van de printerkop. Deze stuurtekens heeft de computer niet nodig, omdat deze via een beeldscherm werkt. De ATARI-constructeurs hebben deze vrije ruimte benut om aanvullende tekens in onder te brengen, die een serieuze telex beslist niet gebruikt, maar voor een hobbycomputer heel nuttig zijn, de zogenaamde pseudo-, semi- of blok-grafische tekens. Om deze code nu

van de norm te kunnen onderscheiden, spreekt men heel toepasselijk over de ATASCII-code. Deze ATASCII-code komt voor een groot gedeelte overeen met de ASCII-norm.

Om het de geïnteresseerde hobbyprogrammeur nu ook weer niet al te gemakkelijk te maken, is er nog een derde systeem dat wordt aangeduid als de interne code. De interne code bepaalt in welke volgorde de data van de 128 karakters in het ROM-geheugen zijn opgeslagen.

Deze interne code onderscheidt zich van de ATASCII-code doordat het totale aantal tekens is verdeeld in drie grote blokken. De relatie tussen ATASCII- en interne code is als volgt:

omzetten van ATASCII-waarden in interne code

0 - 31 en 128 - 159 ATASCII + 64
32 - 95 en 160 - 223 ATASCII - 32
96 - 127 en 224 - 255 gelijk

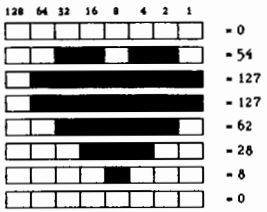
omzetten van interne code in ATASCII-waarden

0 - 63 en 128 - 191 interne code + 32
64 - 95 en 192 - 223 interne code - 64
96 - 127 en 244 - 255 gelijk

Het nulde karakter van de interne code is de spatie (ATASCII 32). De karakterset begint dus met de data voor de spatie.

Ieder karakter wordt weergegeven in een matrix van acht bij acht punten. Ieder afzonderlijk punt licht wel of niet op, wat door het aan of uit zetten van een bit wordt opgeslagen. Acht naast elkaar liggende bits worden samengevoegd tot een byte. Ieder karakter is dus acht bytes groot:

afbeelding het teken ATASCII 0



Het hartje ziet de computer dus als een rij van acht bytes met de decimale waarden 0, 54, 127, 127, 62, 28, 0. Het heeft de ATASCII-waarde 0 en de interne code 64. Dat betekent, dat het in het karakterset-geheugen op de 65e plaats staat. Omdat elk van de 64 karakters daarvoor echter ook acht bytes gebruikt, ligt de eerste byte van het hartje op de 64*8e geheugenplaats voorbij het begin-adres van de karakterset. De interne code bepaalt dus de offset van de data van een teken in het karakterset-geheugen. De HI-byte van het beginadres van de karakterset staat in CHBAS (756 = $2F4). Om de data voor een bepaald karakter te vinden, rekent men:

CHBAS*256 + interne code van het karakter * 8

De data van het gezochte karakter staan op dit adres en op de zeven volgende adressen.

Als men dat eenmaal weet, is het slechts een kleine moeite een eigen karakterset te maken. Daarvoor is het 'alleen' nodig, voor elk van de 128 karakters het patroon van acht maal acht punten in data-bytes om te rekenen en deze 8*128 = 1024 bytes of 1kbyte in het geheugen op te slaan. Laat men dan de wijzer CHBAS naar het begin-adres van deze alternatieve karakterset wijzen, dan verschijnt iedere PRINT, iedere listing op het beeldscherm met de nieuwe karakters.

Hoe dat er dan uitziet, kunt u zien met het volgende programma, dat een cursieve karakterset bevat in de vorm van DATA-regels. Cursief betekent niet dat de karakters rond zijn, maar het betekent dat hun vertikale as naar rechts is gebogen, zoals gebruikelijk is bij een handschrift. De Amerikanen noemen dat 'italic', Joost mag weten waarom.

```
1 REM **********************************
2 REM *                                *
3 REM * CURSIEVE KARAKTERSET (DATA) *
4 REM *                                *
5 REM **********************************
10 POKE 106,PEEK(106)-8
20 GRAPHICS 0
30 A=PEEK(106)+8:C=A*256
40 FOR J=0 TO 1023:READ B:POKE C+J,B:NEXT J
50 ? CHR$(125):POKE 752,1
100 POKE 756,A
110 POSITION 2,7
120 ? "Zo zien de nieuwe karakters eruit."
130 FOR W=0 TO 400:NEXT W
140 POSITION 2,7
150 ? "Zo zien de 'oude' karakters eruit."
160 POKE 756,224
170 FOR W=0 TO 400:NEXT W
180 GOTO 100
30000 DATA 0,0,0,0,0,0,0,0,0,12,12,24,24,0,48,0,0,51,51,102,0,0,0,0
30001 DATA 0,54,127,54,108,254,108,0,12,62,96,56,12,248,48,0,0,108,104,24,16,54,
38,0
30002 DATA 24,52,24,40,74,68,54,0,0,24,24,48,0,0,0,0,0,14,28,24,24,28,14,0
30003 DATA 0,112,56,24,24,56,112,0,0,102,60,255,60,102,0,0,0,8,24,126,24,16,0,0
30004 DATA 0,0,0,0,0,24,24,48,0,0,0,126,0,0,0,0,0,0,0,0,0,24,24,0
30005 DATA 0,6,12,24,48,96,64,0,0,28,50,110,118,76,56,0,0,6,28,12,24,24,48,48
30006 DATA 0,30,50,12,24,48,124,0,0,31,6,8,4,4,40,48,0,6,12,28,40,126,24,48
30007 DATA 0,62,48,56,4,4,76,48,12,16,32,120,100,100,56,0,0,60,4,12,24,48,48,48
30008 DATA 12,18,50,56,76,76,56,0,0,28,54,34,22,12,24,48,0,0,24,24,0,48,48,0
30009 DATA 0,0,24,24,0,48,48,96,6,12,24,48,24,12,6,0,0,0,126,0,0,126,0,0
30010 DATA 96,48,24,12,24,48,96,0,0,28,34,12,24,16,0,32,0,60,54,110,108,64,124,0
```

```
30011 DATA 0,15,27,50,102,252,204,0,0,62,50,124,102,198,252,0,0,30,50,32,96,100,
56,0
30012 DATA 0,60,54,102,102,204,248,0,0,62,48,120,96,192,248,0,0,62,48,120,96,192
,192,0
30013 DATA 0,30,48,96,110,204,120,0,0,51,51,126,102,204,204,0,0,30,12,24,24,48,2
48,0
30014 DATA 0,30,38,12,12,24,152,112,0,51,54,104,120,216,204,0,0,24,24,48,48,96,1
24,0
30015 DATA 0,33,51,127,107,198,198,0,0,51,59,106,110,204,196,0,0,30,51,102,102,2
04,120,0
30016 DATA 0,30,19,54,60,96,96,0,0,30,51,102,102,216,108,6,0,62,51,102,124,216,2
04,0
30017 DATA 0,30,48,60,6,140,120,0,0,63,76,24,24,48,48,0,0,51,99,102,198,204,120,
0
30018 DATA 0,51,35,102,100,104,48,0,0,33,99,67,203,222,228,0,0,34,22,28,24,52,10
0,0
30019 DATA 0,17,35,22,28,24,48,96,0,62,6,24,48,192,248,0,0,30,24,48,48,96,120,0
30020 DATA 0,96,96,48,24,12,12,0,0,30,6,12,12,24,120,0,0,8,28,54,99,0,0,0
30021 DATA 0,0,0,0,0,0,255,0,0,54,127,127,62,28,8,0,24,24,24,31,31,24,24,24
30022 DATA 3,3,3,3,3,3,3,3,24,24,24,248,248,0,0,0,24,24,24,248,248,24,24,24
30023 DATA 0,0,0,248,248,24,24,24,3,7,14,28,56,112,224,192,192,224,112,56,28,14,
7,3
30024 DATA 1,3,7,15,31,63,127,255,0,0,0,0,15,15,15,15,128,192,224,240,248,252,25
4,255
30025 DATA 15,15,15,15,0,0,0,0,240,240,240,240,0,0,0,0,255,255,0,0,0,0,0,0
30026 DATA 0,0,0,0,0,0,255,255,0,0,0,0,240,240,240,240,0,28,28,119,119,8,28,0
30027 DATA 0,0,0,31,31,24,24,24,0,0,0,0,255,255,0,0,0,0,24,24,24,24,255,255,24,24,24
30028 DATA 0,0,60,126,126,126,60,0,0,0,0,0,255,255,255,255,192,192,192,192,192,1
92,192,192
30029 DATA 0,0,0,255,255,24,24,24,24,24,24,255,255,0,0,0,240,240,240,240,240,240
,240,240
30030 DATA 24,24,24,31,31,0,0,0,120,96,120,96,126,24,30,0,0,24,60,126,24,24,24,0
30031 DATA 0,24,24,24,126,60,24,0,0,24,48,126,48,24,0,0,0,24,12,126,12,24,0,0
30032 DATA 0,24,60,126,126,60,24,0,0,0,28,6,62,204,122,0,0,48,48,124,102,204,248
,0
30033 DATA 0,0,60,95,96,192,120,0,0,3,3,62,102,204,124,0,0,0,28,102,124,192,120,
0,
30034 DATA 0,14,24,60,24,48,48,96,0,0,31,54,102,60,12,120,0,48,48,124,102,198,20
4,24
30035 DATA 0,12,0,56,24,48,120,0,0,6,0,12,12,24,24,240,0,48,32,108,120,208,204,0
30036 DATA 0,28,12,24,24,48,120,0,0,0,51,127,122,214,132,0,0,0,62,99,103,198,204
,0
30037 DATA 0,0,30,102,102,204,120,0,0,0,62,102,102,248,192,192,0,0,31,102,102,60
,12,12
30038 DATA 0,0,26,48,48,96,96,0,0,0,31,96,60,12,248,0,0,12,63,24,24,48,48,0
30039 DATA 0,0,51,102,102,204,124,0,0,0,51,102,102,120,48,0,0,0,35,107,87,252,21
6,0
30040 DATA 0,0,51,60,24,124,196,6,0,0,51,38,102,60,24,240,0,0,63,12,24,96,252,0
30041 DATA 0,24,60,126,126,24,60,0,24,24,24,24,24,24,24,24,0,126,120,124,110,102
,6,0
30042 DATA 8,24,56,120,56,24,8,0,255,255,255,255,255,255,255,255
```

programma ITALIC.DAT

10: Het is dringend aan te raden de kostbare karakterset in een beschermend geheugengebied op te slaan. Hoe kostbaar zo'n karakterset is, weet u als u dit programma ingetypt hebt of als u al eens een eigen karakterset hebt geprogrammeerd!

De wijzer RAMTOP (106 = $6A) wordt acht pagina's naar onderen verplaatst. Eigenlijk zijn slechts vier pagina's nodig, maar het is veiliger, net als bij het opslaan van het tekstraam, een bufferzone te reserveren, zodat door het scrollen de data van de karakterset niet worden overschreven. (Alleen #.)

20: De GRAPHICS-instructie zorgt ervoor dat het beeldscherm-geheugen en de display list onder RAMTOP komen te staan. De nieuwe karakterset werkt overigens ook in GRAPHICS 1 en 2 en in alle grafische modi met tekstraam. Bedenk echter wel, dat door iedere GRAPHICS-instructie de standaardwaarde in CHBAS weer wordt hersteld en u dus het

begin-adres van de alternatieve karakterset (alleen de HI-byte) opnieuw op adres 765 moet POKEn.
30: Hier worden de beide benodigde waarden berekend, namelijk de pagina respectievelijk de HI-byte en het volledige adres van de nieuwe karakterset.

40: De FOR-NEXT lus leest nu alle 1024 bytes uit de DATA-regels in en zet ze in het gereserveerde geheugengebied neer. Dat duurt een moment. Maar dan begint het.

50: Het beeldschem wordt gewist en de cursor uitgeschakeld.

100: Hier wordt de wijzer naar de nieuwe karakterset ingesteld en

120: grootvaders ganzeveer schrijft op het beeldscherm.

150: Dan schrijven we een andere tekst en

160: zetten de wijzer weer naar de in ROM opgeslagen standaard karakterset. Opeens is alles weer als normaal.

30000 t/m 30042: Dat is dan de italic-karakterset, byte voor byte.
En zo zien de nieuwe en de standaard ATARI-karakters er op het beeldscherm uit:

tabel uit atari peeks en pokes

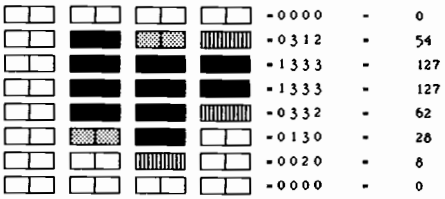
[A] verschijnt negatief

	GRAPHICS	ANTIC commando	KOLOM pixel/ regel	MODE regels	bytes per regel	TV-regels per modus-regel	bits per pixel	kleuren-register 708	709	710	711	712**
karakter-mode	0	2	40	20/24	40	8	8		C.1*	C.0*		R
	-	3	40	16	40	10	8		C.1	C.0		R
	12	4	40	20/24	40	8	8	A/[A]	A/[A]	A	[A]	H
	13	5	40	10/12	40	16	8	A/[A]	A/[A]	A	[A]	H
	1	6	20	20/24	20	8	8	A	a	[A]	[a]	H
	2	7	20	10/12	20	16	8	A	a	[A]	[a]	H
bit-map mode	3	8	40	20/24	10	8	2	C.1	C.2	C.3		H
	4	9	80	40/48	10	4	1	C.1				H
	5	10	80	40/48	20	4	2	C.1	C.2	C.3		H
	6	11	160	80/96	20	2	1	C.1				H
	14	12	160	160/192	20	1	1	C.1				H
	7	13	160	80/96	40	1	2	C.1	C.2	C.3		H
	15	14	160	160/192	49	1	2	C.1	C.2	C.3		H
	8	15	320	160/192	40	1	1		C.1	C.0		R
gita-mode	9		80	192	40	1	4	16 helderheids-gradaties				
	10		80	192	40	1	4	9 kleuren naar keuze				
	11		80	192	40	1	4	16 kleuren/vaste helderheid				

De karakter-modi 12 en 13, die bij de XL-apparaten ook via GRAPHICS opgeroepen kunnen worden, werken in principe net zo. Het onderscheid is, dat telkens twee bits een pixel van het karakter produceren. Daardoor kunnen in totaal vier COLOR-waarden worden onderscheiden, maar een byte bevat nog slechts de informatie voor vier pixels. Om op dezelfde display-grootte uit te komen, zijn de pixels twee beeldpunten groot.

Worden de waarden van de standaard karakterset op deze manier weergegeven, dan ontstaan nauwelijks zinvolle opeenhopingen van gekleurde punten. Deze beide GRAPHICS-modi kunnen pas goed tot hun recht komen met een opnieuw gedefinieerde karakterset.

Het ATARI-hartje ziet er bijvoorbeeld in GRAPHICS 12 of 13 zo uit:



De COLOR-waarden hebben betrekking op de kleurregisters op de adressen 708 (COLOR 1 = 01), 709 (2 = 10), 710 (3 = 11) en op de achtergrond op adres 712 (0 = 00) bij de normale karakters. De inverse karakters krijgen hun kleurwaarde van het kleurregister COLOR 3 op adres 711.

Hier tot slot nog een aardig idee, wat men allemaal met deze gekleurde karakters kan doen:

```
0 REM COLCHR.DAT
1 REM ********************************
2 REM *                              *
3 REM *          CHARACTERS          *
4 REM *                              *
5 REM ********************************
10 GOSUB 100
20 X=INT(RND(0)*39)
30 Y=INT(RND(0)*21)
40 Z=INT(RND(0)*11)
50 J=Y*40+X+1:Z=Z+1
60 P$(J,J)=CHR$(Z,Z)
70 POSITION 0,0:? #6;P$:GOTO 20
100 DIM P$(680),C$(11):C$="#$%&'()*+,-"
110 P$="#":P$(800)=P$:P$(2)=P$
120 POKE 106,PEEK(106)-4
130 GRAPHICS 20
140 A=PEEK(106)*4:C=A*256
150 FOR J=0 TO 111:READ B:POKE C+J,B:NEXT J
200 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
210 DATA 66,66,24,24,36,36,129,129,78,78,116,116,39,39,177,177,112,112,27,2
7,220,220,141,141
220 DATA 74,74,16,16,4,4,161,161,66,66,16,16,132,132,129,129,66,66,28,28,18
0,180,129,129
230 DATA 74,74,28,28,52,52,161,161,74,74,16,16,132,132,129,129,66,66,18,18,
4,4,161,161
240 DATA 66,66,30,30,52,52,161,161,74,74,28,28,180,180,129,129
250 POKE 756,A
260 RETURN
```

10: Het begin van het programma staat weer in de onderste regels, zodat het werkende gedeelte tenminste iets sneller loopt.

100: In P$ moet de beeldinhoud als string worden opgeslagen. Bij 40 karakters per modus-regel worden daarmee 22 regels gevuld. In C$ wordt de regel met karakters opgeslagen, die als beeldelementen worden gebruikt.

110: vult P$ met 880 ruiten (nummer-tekens) volgens de methode die besproken is in het programma ADR135A.BEC. Hier kunt u ook ieder ander in CS opgeslagen karakter neerzetten of een van de overige karakters. Enige van de nieuw gevormde karakters bevatten geringe gedeelten in COLOR 3. Als u deze karakters in hun inverse vorm oproept, bereikt u de vierde of, de achtergrond meegerekend, de vijfde kleur.

120 t/m 150: reserveren geheugenruimte voor de nieuwe karakterset, die deze keer niet zo groot hoeft te zijn, omdat slechts elf nieuwe karakters worden gevormd.

200: De DATA van de nieuwe karakters. In de volgorde van de interne code staat de spatie op de nulde plaats, gevolgd door het uitroepteken en de aanhalingstekens. Omdat aanhalingstekens in een string niet kunnen voorkomen, in ieder geval niet in ATARI-BASIC, zijn voor dit doel hier de eerste drie karakters niet gebruikt. Pas de volgende elf karakters dienen als beeldelementen. Daarom leest de FOR-NEXT lus in regel 150 veertien (14*8=112) data.

250: verandert de CHBAS-wijzer en

260: dan begint het.

20 en 30: Ook al is de inhoud van het beeldscherm als karakters in een lintworm-string ondergebracht, toch kunnen we net doen, alsof we met beeldscherm-posities te maken hebben, die gewoon met X en Y worden aangegeven. Omdat de string altijd op dezelfde plaats beginnend op het beeldscherm wordt gePRINT, belandt het eerste beeldelement ook altijd op dezelfde plaats van het beeldscherm. P$ is hier functioneel niets anders dan een beeldscherm-geheugen waarin rangnummers worden bewaard, die verwijzen naar complexe, gekleurde grafische informatie, namelijk de door ons gevormde karakters, om deze op het beeldscherm weer te geven.

Op dezelfde manier kunt u zich de functies van het beeldscherm-geheugen in de karaktermodus voorstellen. Op de geheugenplaatsen worden de waarden van de karakters in interne code neergezet, waarmee dan de op het beeldscherm weer te geven bit-patronen uit de karakterset-ROM worden gelezen.

Het grote voordeel van de string-methode is, dat het geen problemen geeft met de te reserveren geheugenruimte. Want de gedefinieerde beeldelementen op het beeldscherm brengen, zou ook in GRAPHICS 15 (XL) mogelijk zijn, maar alleen met moeizame afzonderlijke PLOTs, die ook overeenkomstig veel rekentijd zouden verbruiken. Hier wordt met ieder element een vlak gevuld van acht bij acht beeldpunten (4 maal 8 pixels). De beperking is dat het beeldscherm alleen samengesteld kan worden uit een patchwork van voorgevormde patronen. Als u dit programma eens nader bekijkt, zult u echter ook zien hoe klein deze stukjes zijn en dat de afzonderlijke pixels in dit patroon verloren gaan.

De string-methode vereenvoudigt dus het vormingswerk van gekleurde grafische beelden bij een gering geheugengebruik en relatief hoge verwerkingssnelheid. Omdat in totaal 128 beeldelementen gedefinieerd kunnen worden, kunt u een veelzijdige blokkendoos met grafische elementen definiëren, waaruit dan echt snel bijvoorbeeld een landschap of een kaleidoscoop of wat dan ook samengesteld kan worden.

In deze beide programmaregels worden dus willekeurige beeldscherm-posities X en Y bepaald, zoals dat bij een andere weergavevorm, bijvoorbeeld GRAPHICS 3, ook kan gebeuren.

40: En hier wordt een willekeurig getal van 0 t/m 10 geproduceerd, dat later het gevormde element moet uitkiezen.

50: J berekent uit Y en X de positie in de string, zoals men uit Y en X het adres in het beeldscherm-geheugen zou bepalen, namelijk regelpositie (Y) maal het aantal tekens per regel (40) plus kolompositie (X). Omdat we met een string te maken hebben en het eerste teken van een string nu eenmaal niet positie 0 maar 1 heeft, moet bij de gevonden waarde nog 1 worden opgeteld.

Dat geldt ook voor het willekeurige getal Z. Een string heeft geen nulde karakter.

60: Het karakter, dat in P$ op de berekende plaats J staat (P$(J,J), wordt vervangen door het toevallig verkregen karakter Z, dat in C$ op de plaats Z staat.

70: PRINT P$, beginnend in de linker bovenhoek van het beeldscherm en laat het programma opnieuw doorlopen.

Als het doel van zo'n programma alleen hieruit bestaat, een veranderend gekleurd beeld op het beeldscherm te toveren, dan is het natuurlijk niet nodig, de willekeurig aangewezen beeldelementen voortdurend in de string tussen te voegen en bij iedere doorloop de complete P$ weer te geven. Dezelfde werking kan worden bereikt, als met POSITION de toevallig verkregen beeldscherm-positie wordt aangewezen en het beeldelement C$(Z,Z) daarheen wordt gePRINT.

De hier voorgestelde methode slaat de aktuele beeldinhoud voortdurend op. De beeldinhoud is dus onafhankelijk van zijn weergave op het beeldscherm. De verandering zou ook verder kunnen lopen, terwijl op het beeldscherm iets geheel anders wordt getoond. Het programma zou ook ogenblikkelijk kunnen omschakelen tussen P$ en andere strings met andere beeldinhouden en daarmee het probleem van snelle scenewisselingen in een grafische modus met hoge resolutie op een elegante en (geheugen-)ruimte besparende manier oplossen.

Label ABC

APPHMHI	14
ARGOPS	128
ATACHR	763
ATRACT	77
AUDC1	53761
AUDC2	53763
AUDC3	53765
AUDC4	53767
AUDCTL	53768
AUDF1	53760
AUDF2	53762
AUDF3	53764
AUDF4	53766
BFENLO/HI	52
BITMSK	110
BOOT?	9
BOTSCR	703
BPTR	61
BRKKEY	17
BRKKY	566
BUFADR	21
BUFCNT	107
BUFRFL	56
BUFRLO/HI	50
BUFSTR	108
CASFLG	783
CASINI	2
CASSBT	75
CBAUD	750
CDTMA1-2	550
CDTMF3-5	554
CDTMV1-5	536
CFB	570
CH	764
CH1	754
CHACT	755
CHACTL	54273
CHAR	762
CHBAS	756
CHBASE	54281
CHKSNT	59
CHKSUM	49
CIX	242
CKEY	74
COLAC	114
COLBK	53274
COLCRS	85
COLDST	580
COLINC	122???
COLORO-4	708
COLPFO-3	53270
COLPMO-3	53266
COLRSH	79
CONSOL	53279
COUNTR	126
CRETRY	54
CRITIC	66
CRSINH	752
DAUX1-2	778
DBSECT	577
DBUFLO/HI	772
DBYTLO/HI	776
DCB	768
DEGFLG	251
DELTAC	119
DELTAR	118
DINDEX	87
DLISTL/H	54274
DMACTL	54272
DMASK	672
DOSINI	12
DOSVEC	10
DRETRY	55
DRKMSK	78
DSKFMS	24
DSKTIM	582
DSKUTL	26
DSPFLG	766
DSTAT	76
DSTATS	771
DTIMLO	774
DUNIT	769

DUNUSE	775
DVSTAT	746
ENDPT	116
ENDSTAR	142
ERRSAVE	195
ESCFLG	674
ESIGN	239
FEOF	63
FILDAT	765
FILFLG	695
FMZSPG	67
FREQ	64
FTYPE	62
GLBABS	736
GPRIOR	623
GRACTL	53277
GRAFM	53265
GRAPFO-3	53261
HATABS	794
HITCLR	53278
HLPFLG	732
HOLDCH	124
HOLDI	81
HPOSMO-3	53252
HPOSPO-3	53248
ICAX1Z-6Z	42
ICBALZ/HZ	36
ICBLLZ/HZ	40
ICCOMT	23
ICCOMZ	34
ICDNOZ	33
ICHIDZ	32
ICPTLZ/HZ	38
ICSTAZ	35
INBUFF	243
INITAD	738
INSDAT	125
INTEMP	557
INTRVEC	522
INVFLG	694
IOCBO-7	832
IRQEN	53774
KEYDEF	121
KEYDEL	729
KEYDEL	753
KEYREP	730
LCONT	563
LINBUF	583
LINZBS	0
LMARGN	82
LOGCOL	99
LOGMAP	690
LOMEM	128
LPENH/V	564
MEMLO	743
MEMTOP	144
MEMTOP	741
NEWCOL	97
NEWROW	96
NOCKSM	60
NOCLIK	731
OLDADR	94
OLDCHR	93
OLDCOL	91
OLDROW	90
OUTBUFF	128
PADDLO-7	624
PBPNT	29
PBUFSZ	30
PCOLRO-3	704
PMBASE	54279
PRIOR	53275
PRNBUF	960
PTABW	201
PTEMP	31
PTIMOT	28
PTRIGO-7	636
RADFLG	251
RAMLO	4
RAMSIZ	740
RAMTOP	106
RANDOM	53770
RECVDN	57
RMARGN	83

ROWAC	112
ROWCRS	84
ROWINC	760 OF 121?
RTCLOK	18
RUNAD	736
RUNSTK	142
SCRFLG	699
SDLSTL	560
SDMCTL	559
SHFAMT	111
SHFLOK	702
SOUNDR	65
SRTIMR	555
SSFLAG	767
SSKCTL	562
STACKP	792
STARP	140
STATUS	48
STICKO-3	632
STMCUR	138
STMTAB	136
STOPLN	286
STRIGO-3	644
STRTFLG	1001
SWPFLG	123
TABMAP	675
TEMP	80
TIMER1	780
TIMER2	784
TIMFLG	791
TINDEX	659
TMPCHR	80
TMPCOL	697
TMPLBT	673
TMPROW	696
TRAMSZ	6
TSTAT	793
TSTDAT	7
TXTCOL	657
TXTMSC	660
TXTOLD	662
TXTROW	656
VBREAK	518
VDELAY	53276
VDSLST	512
VIMIRQ	534
VINTER	516
VKYBD	520
VNDT	132
VNTP	130
VPRCED	514
VSERIN	522
VSEROC	526
VSEROR	524
VTIMR1-4	528
VVBLKD	548
VVBLKI	546
VVTP	134
WARMST	8
XMTDON	58

Bij de *ATARI computers (600/800 XL en 130 XE)* kunnen met de instructies *peek* en *poke* erg nuttige dingen bereikt worden. Op eenvoudige wijze gaat de schrijver van dit boek langs alle adressen, die met deze twee instructies gewijzigd kunnen worden. Naast de onderwerpen als *bits* en *bytes, memory-map, grafiek modi* en *het produceren van geluid* wordt tegelijkertijd de opbouw en werking van deze ATARI computers verklaard.

ISBN 90 229 3348 2

ATARI Bibliotheek 1

DATA BECKER
NEDERLANDS*